Nonlinear Approaches to Satisfiability Problems

Joost P. Warners

Stellingen

behorende bij het proefschrift

Nonlinear Approaches to Satisfiability Problems

van

Joost P. Warners

1

Niet-lineaire modellen in het algemeen en kwadratische modellen in het bijzonder zijn breed inzetbaar in de ontwikkeling van effectieve *Satisfiability* algoritmes. Door gebruik te maken van de specifieke eigenschappen van binaire variabelen kunnen modellen van verschillende aard worden verkregen, elk met eigen specifieke toepassingen.

Dit proefschrift.

2

We beschouwen Karmarkar's potentiaal reduktie algoritme voor combinatorische optimaliseringsproblemen. Om een dalingsrichting te berekenen dient een lineair stelsel te worden opgelost. In [1] wordt gesteld dat wegens de dichtheid van de betrokken matrix, het gebruik van directe factorisatie methodes niet praktisch is. Derhalve wordt een efficiënte techniek voorgesteld om een dalingsrichting te berekenen die niet noodzakelijk aan de gestelde optimaliteits-eisen voldoet.

Door gebruik te maken van het feit dat de betrokken matrix de som is van een ijle matrix en een dichte matrix van rang een, kan een methode worden ontwikkeld die gebruik maakt van directe factorisaties, efficiënt de ijlheid benut en optimale oplossingen levert.

[1] P.M. Pardalos en M.G.C. Resende. Interior point methods for global optimization. In: T. Terlaky, samensteller. *Interior point methods for mathematical programming.* Kluwer Academic Publishers, 1996.

[2] E.D. Andersen, C. Roos, T. Trafalis, T. Terlaky en J.P. Warners. The use of low-rank updates in interior-point methods. Manuscript, 1999.

3

De mate van effectiviteit waarmee HeerHugo in staat blijkt moeilijke *Satisfiability* problemen op te lossen, geeft aan dat men het in dit vak niet alleen figuurlijk, maar ook wel eens letterlijk in de breedte moet zoeken.

J.F. Groote en J.P. Warners. The propositional checker HeerHugo. CWI rapport SEN-R9905, 1999. Wordt gepubliceerd in de*SAT2000* uitgave van *Journal of Automated Reasoning.*

4

Stelling 5.6.3 van dit proefschrift (pag. 103) kan worden aangescherpt tot

$$\omega(G) \leq |C^*| \leq \vartheta(\overline{G}) \leq \gamma(G).$$

Hier staat $\vartheta(\overline{G})$ voor het Lovász ϑ–getal van de complement graaf van G.

E. de Klerk, D.V. Pasechnik en J.P. Warners. Approximate graph colouring algorithms based on the ϑ-function. Manuscript, 1999.

5

Definieer de familie van functies $\varphi_\alpha(x) = x^T(Q - \alpha I)x - 2q^T x$, met $Q \in \mathbb{R}^{m\times m}$ en q, $x \in \mathbb{R}^m$. De grootste en kleinste eigenwaarde van Q zijn $\lambda_{\max}$ en $\lambda_{\min}$. Beschouw het NP-moeilijke optimaliseringsprobleem

$$\begin{array}{ll} \min & \varphi_0(x) \\ \text{s.t.} & x \in \{-1,1\}^m, \end{array}$$

met optimale waarde $opt(\varphi_0)$. Laat N een bovengrens zijn van de functie $\varphi_0(x)$ over de hoekpunten van de m-dimensionale kubus. Verder is, voor $\alpha > \lambda_{\min}$, $c_\alpha = (Q - \alpha I)^{-1}q$. Met c_α wordt een $\{-1,1\}$ vector $\text{sgn}(c_\alpha)$ geassocieerd. Als α gekozen is zodat $\alpha > \lambda_{\min}$ en $\|c_\alpha - \text{sgn}(c_\alpha)\|^2 \leq m$, dan geldt

$$0 \leq \frac{\varphi_0(\text{sgn}(c_\alpha)) - opt(\varphi_0)}{N - opt(\varphi_0)} \leq \frac{(\lambda_{\max} - \lambda_{\min})m}{N - \lambda_{\min} - \varphi_\alpha(c_\alpha)}.$$

H. van Maaren en J.P. Warners. Bounds and fast approximation algorithms for binary quadratic optimization problems with application to MAX 2SAT and MAXCUT. Rapport 97-35, Faculteit der Technische Wiskunde en Informatica, Technische Universiteit Delft, 1997.

6

Een versterking van de semidefiniete relaxatie voor 3CNF problemen zoals beschreven in Sectie 5.6.3 kan worden verkregen door hem te relateren aan de polynomiale representatie van clauses (zie pag. 25, vergelijking (2.4)). Er geldt dat

$$s_k = \frac{1}{2}\left(1 - \prod_{i \in I_k \cup J_k} a_{ki}x_i\right).$$

Middels deze gelijkheid kunnen additionele toegelaten gelijkheden worden geformuleerd en toegevoegd aan de relaxatie, met als gevolg dat de relaxatie niet langer noodzakelijk de triviale oplossing toelaat (vgl. Lemma 5.6.9). Deze tamelijk natuurlijke methode om de relaxatie te versterken is waarschijnlijk van cruciaal belang om op termijn semidefiniete programmering met succes toe te passen om *Satisfiability* problemen op te lossen.

7

Het feit dat veel algoritmes extreme moeilijkheden ondervinden om te beslissen dat het *pigeon hole* probleem geen toegelaten oplossing heeft, is slecht voor het imago van de kunstmatige intelligentie.

Zie voor een omschrijving van het *pigeon hole* probleem pag. 100 van dit proefschrift.

8

De hedendaagse, vaak zeer realistisch ogende en extreem gewelddadige computerspelen zijn als drugs: geestverruimend, verslavend en potentieel gevaarlijk voor de mens en zijn omgeving.

9

Gezien de enorme belangen die op het spel staan bij het hedendaagse voetbal enerzijds en het feit dat men ter ontspanning wedstrijden bezoekt anderzijds, verdient het aanbeveling een extra scheidsrechter aan te stellen, die, indien nodig, aan de hand van videobeelden direct de scheidsrechter op het veld kan corrigeren.

10

Hij die zichzelf overwint lijdt ook een nederlaag.

Nonlinear Approaches to Satisfiability Problems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr. M. Rem, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 14 september 1999 om 16.00 uur

door

Johannes Pieter Warners

geboren te Borsele

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. J.F. Groote
en
prof.dr. J. van Leeuwen

Copromotor:
dr. H. van Maaren

The research in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics), at the Faculty of Information Technology and Systems of Delft University of Technology, and at the CWI in Amsterdam. It was supported by the Dutch Organization for Scientific Research NWO/SION, under grant 612–33–001.

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Warners, Johannes Pieter

Nonlinear Approaches to Satisfiability Problems / Joost P. Warners. - Ill.
Thesis Eindhoven University of Technology.
With ref. - With summary in Dutch. - xiv, 162 p.
ISBN 90-9012936-7
NUGI 811
Subject headings: satisfiability / propositional logic / mathematical programming

IPA Dissertation Series 1999-08.

Printed by Ponsen & Looijen, Wageningen.

Contents

Acknowledgement

One might say that performing Ph.D. research and writing the corresponding thesis is in fact very similar to solving an instance of satisfiability. There are several 'clauses' to be satisfied: the research should be interesting and enjoyable to perform, as well as to write about and to read. Eventually, the thesis should represent the solution that satisfies all these clauses (or at least as many as possible).

The approach I took certainly had a nonlinear feel to it as I was working at two institutions: Delft University of Technology and CWI in Amsterdam. In addition, I live in Utrecht and spent a few months at the department of Philosophy of Utrecht University as well, while the defense of this Ph.D. thesis takes place in Eindhoven. Thus, during these years time I've enjoyed the company of many different people in many different places. Consequently there are many people who, in one way or another, contributed to the solution of this particular satisfiability problem. Without them, I wouldn't have succeeded. I couldn't possibly mention them all individually, so here's a shortlist.

I thank my supervisor Hans van Maaren with whom I had the pleasure of collaborating intensively. On the numerous drives from Utrecht to Delft and back many a brilliant idea was conceived, discussed in detail and rejected ... well some remained. This lead to a number of papers, most of which are featured in this thesis. Also many thanks are due to Etienne de Klerk, with whom I co-wrote two papers that form the basis of Chapter 5. I also thank my other supervisor and promotor Jan Friso Groote, and the other members of the reading committee for being prepared to read and comment on the concept version of this thesis: my second promotor Jan van Leeuwen, Emile Aarts and John Franco.

I thank the colleagues I had the honour of co-authoring papers with: Kees Roos, Tamás Terlaky, Benjamin Jansen, Theodore Trafalis, Arie Quist, Thomas Schiex, Dima Pasechnik and Erling Andersen. Thanks go to all colleagues at Delft and the CWI. Inevitably failing to mention all, thank you Bernd, Bas, Bert, Izak, Jaco, Jos, Judi, Mark, Michel, Vincent, Yaroslav. I acknowledge NWO/SION for their financial support.

Outside work, and that's just as important to make it all worthwhile, I enjoyed many great hours in the company of family and friends; I thank you all. In particular I thank the Red Cloaked Wizards for the walks and talks, rain or shine; the Doomed Spiders for all the noise and great music; and the other two 'musketiers'.

Finally, special thanks and love go to pa en ma, and Marieta.

Preface

The satisfiability problem of propositional logic (SAT) captures the essence of many difficult problems in disciplines such as artificial intelligence, computer science, electrical engineering and operations research. It concerns the problem of deciding whether or not a logical formula contains a contradiction. In general no (theoretically) efficient algorithms are available to solve it; it is considered unlikely that such algorithms exist. As an exception, various specific classes of SAT allow efficient algorithms. In the nineteen nineties SAT has experienced a growing interest and activity, which has been spurred by several developments. On the one hand, the speed and memory size of contemporary computers has revived the interest for the classical satisfiability algorithms, but just as important is the discovery of new algorithms that in practice seem to suffer less from the combinatorial explosion. The latter include *incomplete* local search algorithms as opposed to *complete* systematic search algorithms. Complete algorithms give a definite answer as to the (un)satisfiability of a formula, while incomplete algorithms are not capable of recognizing contradictions. However, if a formula has a solution, it is often found much faster using an incomplete than using a complete algorithm. Thus many instances can be solved relatively easy by one of the available state-of-the-art solvers, or by recognizing them as being a member of an efficiently solvable class and solving them as such. The problem sizes that can be handled by current algorithms and implementations have increased to such an extent that it is now possible to solve SAT encodings of real-world practical problems. Since many interesting problems can be modelled using propositional logic, an often successful approach is to solve them using a dedicated SAT algorithm rather than by spending much time and effort on developing special-purpose algorithms for each of them individually.

To cope with SAT problems of ever larger sizes, algorithms (both complete and incomplete) and implementations are improved continually. Unfortunately, certain types of problems have remained very difficult to solve. It is commonly expected [116] that for making substantial progress on these problems new approaches are required. Either new algorithms need to be developed, or existing ones need to be enhanced with powerful tools to exploit special structures that are not exploited by current algorithms. The main subject of this thesis is the development and application of such new techniques (both complete and incomplete) for satisfiability (and related) problems. We make use of techniques and methods from the area of *logic*, *artificial intelligence* and *mathematical programming*, including *semidefinite programming*, both to enhance and to extend existing algorithms and to develop new algorithms for specific classes of SAT.

Organization of this thesis

This thesis is subdivided in six chapters. The first and second of these give an introduction to the satisfiability problem, its complexity, some applications, a number of algorithms and models relevant for this thesis. Along the way, classes of difficult satisfiability problems are encountered and some interesting and challenging phenomena in SAT solving are pointed out. These act as a motivation for the research described in the remaining four chapters. These chapters can be read independently, since all treat different aspects of and approaches to satisfiability solving. Where necessary they are cross-referenced. The chapters are based on several papers which are specified below.

Chapter 3 *Enhancing the DPLL algorithm using elliptic approximations.*

J.P. Warners and H. van Maaren. Solving satisfiability problems using elliptic approximations - Effective branching rules. To appear in *Discrete Applied Mathematics.*

H. van Maaren and J.P. Warners. Solving satisfiability problems using elliptic approximations - A note on volumes and weights. Technical Report 98-32, Faculty of Information Technology and Systems, Delft University of Technology, 1998. Submitted.

Chapter 4 *A two-phase algorithm for a class of hard satisfiability problems.*

J.P. Warners and H. van Maaren. Recognition of tractable satisfiability problems through balanced polynomial representations. To appear in *Discrete Applied Mathematics.*

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

Chapter 5 *Semidefinite relaxations of satisfiability problems.*

E. de Klerk and J.P. Warners. Semidefinite programming techniques for MAX-2-SAT and MAX-3-SAT: Computational perspectives. Technical Report 98–34, Faculty of Information Technology and Systems, Delft University of Technology, 1998. Submitted.

E. de Klerk, H. van Maaren, and J.P. Warners. Relaxations of the satisfiability problem using semidefinite programming. Technical Report SEN-R9903, CWI, Amsterdam, 1999. To appear in the *SAT2000* issue of the *Journal of Automated Reasoning*

Chapter 6 *A nonlinear approach to combinatorial optimization.*

J.P. Warners. A nonlinear approach to a class of combinatorial optimization problems. *Statistica Neerlandica*, 52(2):162–184, 1998.

J.P. Warners. Nonconvex continuous models for combinatorial optimization problems with application to satisfiability and node packing problems. Technical Report SEN-R9710, CWI, Amsterdam, 1997.

Appendix A *Representing linear inequalities by CNF formulas.*

J.P. Warners. A linear–time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68:63–69, 1998.

Notation

For easy reference, we list the notation most frequently used in this thesis. Notation used locally in a single chapter only is not included. Also, some of the notation used locally differs slightly; this is indicated where necessary in the chapters themselves.

n	:	number of clauses of a given CNF formula.
m	:	number of variables of a given CNF formula.
$\mathbb{R}^m$	:	m-dimensional Euclidian vector space.
$\mathbb{R}^{n\times m}$	:	space of $(n \times m)$ real matrices.
Φ, Ψ	:	propositional formulas, usually in CNF.
$\vee, \wedge, \rightarrow, \leftrightarrow$	:	or, and, implication, bi-implication.
$(\neg)p_i$	:	(negated) propositional variable.
l_i	:	literal; $l_i \equiv p_i$ or $l_i \equiv \neg p_i$.
$\mathbf{C}_k$	:	clause k; $\mathbf{C}_k \equiv \bigvee_{i\in I_k} p_i \vee \bigvee_{i\in J_k} \neg p_i$.
$\ell(\mathbf{C}_k)$	:	length of clause k; $\ell(\mathbf{C}_k) = \|I_k \cup J_k\|$.
$\sigma(l_i)$	:	number of occurrences of literal l_i in a given formula.
Q_{ij}	:	entry (i,j) of a square symmetric matrix Q.
a_{ki}	:	entry (k,i) of a non-square matrix A.
$A \in \mathbb{R}^{n\times m}$	:	(usually) clause-variable matrix; $a_{ki} = 1, -1, 0$ if $i \in I_k$, $i \in J_k$, otherwise.
$A^T \in \mathbb{R}^{m\times n}$	:	transpose of a matrix $A \in \mathbb{R}^{n\times m}$.
a_k^T	:	row k of matrix A.
$\mathrm{diag}(A)$	:	diagonal matrix containing the diagonal entries of $A \in \mathbb{R}^{m\times m}$.
$\mathrm{diag}(a)$	:	diagonal matrix with the vector $a \in \mathbb{R}^m$ on its diagonal.
$b \in \mathbb{R}^n$	:	$b_k = 2 - \ell(\mathbf{C}_k)$, $1 \le k \le n$.
$r \in \mathbb{R}^n$	:	$r_k = \ell(\mathbf{C}_k)(\ell(\mathbf{C}_k) - 2)$, $1 \le k \le n$.
$w \in \mathbb{R}^n$	:	vector of nonnegative clause-weights.
W	:	diagonal matrix; $W = \mathrm{diag}(w)$.
(IP_{SAT})	:	integer programming formulation of SAT; find $x \in \{-1,1\}^m$ such that $Ax \ge b$.
$\mathcal{E}(w)$	:	weighted elliptic approximation of a given formula; $\mathcal{E}(w) = \{x \in \mathbb{R}^m \mid x^T A^T W A x - 2w^T A x \le r^T w\}$.
e	:	all-one vector of appropriate length.
e_T	:	vector with ones in entries $i \in T$ and zeros elsewhere.
x_T	:	vector x with its entries $i \notin T$ set to zero.
$\mathrm{sgn}(x)$	:	equals $-1, 0, 1$ if x is negative, zero, positive; on vectors and matrices sgn is taken component-wise.
$\mathbf{Tr}\ A$	:	trace of $A \in \mathbb{R}^{m\times m}$; $\mathbf{Tr}\ A = \sum_{i=1}^{m} A_{ii}$.
$\|\mathcal{M}\|$	:	cardinality (number of elements) of the set $\mathcal{M}$.

1

The Satisfiability Problem

A short introduction to the satisfiability problem of propositional logic is given. We discuss its complexity, describe a number of applications, mention the best known algorithms and give an indication of the effectiveness of a subset of algorithms on a set of standard benchmarks. Thus several interesting and challenging issues are revealed, which act as motivation for the development of new algorithms in this thesis.

1.1 Propositional logic, SAT and CNFs

We give a rather informal description of propositional logic. For a more formal treatment the reader is referred to e.g. van Dalen [35]. Given is a set $\{p, p_0, p_1, p_2, \ldots, q, r, \ldots\}$ of *atomic propositions* or *variables*. Each proposition can have one of two *truth values*: *true* or *false*. *Propositional formulas* can be constructed using a number of *connectives*: $\neg$ ('not' or 'negation'), $\vee$ ('or' or 'disjunction'), $\wedge$ ('and' or 'conjunction'), $\rightarrow$ ('implication') and $\leftrightarrow$ ('equivalence' or 'bi-implication'). We assume these connectives to have their 'natural' interpretation, which for the sake of completeness we briefly review:

- $\neg p$ denotes that p is *false*. Furthermore, $\neg(\neg p) \equiv p$.
- $p \vee q$ means that at least one of p and q is *true*.
- $p \wedge q$ implies that both p and q are *true*.
- $p \rightarrow q$ means that if p is *true* then q must be *true* as well (if $\neg p$, then q can have both truth values).
- $p \leftrightarrow q$ implies that either both p and q must be *true*, or both must be *false*.

Note that in the above p and q can both be atomic propositions or propositional subformulas.

A *literal* is an atomic proposition or its negation. Thus, the set of literals is $\{p, \neg p, p_0, \neg p_0, \ldots, q, \neg q, \ldots\}$. Propositional (sub)formulas are denoted by Φ and Ψ.

Example 1.1.1 An example of a propositional formula is

$$\Phi_0 = (p_0 \leftrightarrow \neg p_1) \vee ((\neg p_0 \rightarrow p_2) \wedge p_1).$$

Here p_0, p_1, p_2 are the atomic propositions, p_0, $\neg p_1$, $\neg p_0$ etc. are the literals and Φ_0 has five subformulas: (1) $(p_0 \leftrightarrow \neg p_1)$, (2) $((\neg p_0 \rightarrow p_2) \wedge p_1)$, (3) $(\neg p_1)$, (4) $(\neg p_0 \rightarrow p_2)$ and (5) $(\neg p_0)$. □

Given a propositional formula Φ, the satisfiability problem of propositional logic can be stated as follows.

Find an assignment of truth values $\{true, false\}$ to the propositional variables such that Φ evaluates to true, or prove that no such truth value assignment exists.

Henceforth we often use the abbreviation SAT for satisfiability problem. If the formula is *true* for all assignments to the variables, it is called a *tautology*, while if it is *false* for all assignments it is said to be a *contradiction* or *unsatisfiable*. Otherwise it is said to be *satisfiable*. Given a formula containing m propositional variables, there are 2^m distinct truth value assignments to the variables that can be evaluated via a *truth table*.

Example 1.1.2 For formula Φ_0 we can set up the truth table as given in Table 1.1. All subformulas of Φ_0 are evaluated separately in order to evaluate the full formula. By 't' we denote *true*, by 'f' we denote *false*. Examining the truth table we conclude that

p_0	p_1	p_2	$\neg p_0$	$\neg p_1$	$p_0 \leftrightarrow \neg p_1$	$\neg p_0 \rightarrow p_2$	$(\neg p_0 \rightarrow p_2) \wedge p_1$	Φ_0
t	t	t	f	f	f	t	t	t
t	t	f	f	f	f	t	t	t
t	f	t	f	t	t	t	f	t
t	f	f	f	t	t	t	f	t
f	t	t	t	f	t	t	t	t
f	t	f	t	f	t	f	f	t
f	f	t	t	t	f	t	f	f
f	f	f	t	t	f	f	f	f

Table 1.1: Truth table for evaluating Φ_0.

Φ_0 is neither a contradiction nor a tautology, but it is satisfiable. It has 6 satisfying assignments. □

In this thesis, the satisfiability problem is studied in *conjunctive normal form* (CNF). A CNF formula consists of a *conjunction of disjunctions of literals*. Each of the disjunctions is called a *clause*, in notation:

$$\Phi = \mathbf{C}_1 \wedge \ldots \wedge \mathbf{C}_n = \bigwedge_{k=1}^{n} \mathbf{C}_k,$$

where each clause $\mathbf{C}_k$ is of the form

$$\bigvee_{i \in I_k \cup J_k} l_i \equiv \bigvee_{i \in I_k} p_i \vee \bigvee_{i \in J_k} \neg p_i, \tag{1.1}$$

with l_i a literal. The *length* $\ell(\mathbf{C}_k)$ of a clause is defined as the number of distinct literals occurring in its minimal representation, i.e. tautological clauses reduce to *true* and have no length, while doubled occurrences of identical literals in the same clause are reduced to a single occurrence. Hence we assume that $I_k \cap J_k = \emptyset$, and that no entry occurs more than once in I_k c.q. J_k. Then $\ell(\mathbf{C}_k) = |I_k \cup J_k|$. Clauses $\mathbf{C}_k$ with $\ell(\mathbf{C}_k) = 1$ are called *unit clauses.*

Example 1.1.3 Examples of clauses are $\mathbf{C}_1 \equiv (p_1 \vee \neg p_3 \vee p_4 \vee p_3)$ and $\mathbf{C}_2 \equiv (p_2 \vee p_6 \vee \neg p_4 \vee p_6)$. $\mathbf{C}_1$ is a tautology, while $I_2 = \{2, 6\}$, $J_2 = \{4\}$, hence $\ell(\mathbf{C}_2) = 3$. □

A CNF formula in which the maximum clause length is equal to ℓ is referred to as an ℓCNF or ℓSAT formula. If *all* clauses have length *exactly* ℓ, the formula is called a *pure* ℓCNF (or ℓSAT) formula. The complexity of the SAT problem on CNF formulas is related to clause lengths. Some relevant complexity issues are briefly reviewed in the next section.

Let us now argue that without loss of generality we can restrict ourselves to CNF formulas. First note that the $\leftrightarrow$ and $\rightarrow$ operators can be eliminated using the well known De Morgan's laws to obtain a CNF formula:

$$(p \leftrightarrow q) \equiv (p \rightarrow q) \wedge (p \leftarrow q) \equiv (\neg p \vee q) \wedge (p \vee \neg q).$$

Example 1.1.4 Using De Morgan's laws, Φ_0 can be reduced to the CNF formula $(p_0 \vee p_1) \wedge (p_0 \vee p_1 \vee p_2)$. The first clause *dominates* or *subsumes* the second; i.e. the second clause is always *true* when the first is *true*. Thus $\Phi_0 \equiv (p_0 \vee p_1)$. Note that this is confirmed by the truth table 1.1. □

In general, using these laws, transforming an arbitrary formula to CNF requires a number of operations that is exponential in the length of the formula. However, using auxiliary variables, for any propositional formula a *satisfiability-equivalent* 3CNF formula can be constructed in linear time.

Definition 1.1.5 (Satisfiability-equivalent) *The formulas Φ and Ψ are said to be* satisfiability-equivalent *if either both are satisfiable or both are contradictory.*

Such a construction is believed to be first described by Tseitin [129] and therefore we refer to it as the *Tseitin construction.* We informally describe the Tseitin construction; see also [11, 58, 139]. Suppose we are given a propositional formula Φ. For any subformula Ψ of Φ, introduce a new proposition p_Ψ and construct the following formula:

$$p_\Phi \wedge \bigwedge_{\Psi \equiv (\Psi_1 \oplus \Psi_2) \subseteq \Phi} (p_\Psi \leftrightarrow (p_{\Psi_1} \oplus p_{\Psi_2})) \wedge \bigwedge_{\Psi \equiv \Psi_1 \subseteq \Phi} (p_\Psi \leftrightarrow \neg p_{\Psi_1}).$$

Here $\Psi \subseteq \Phi$ denotes that Ψ is a subformula of Φ, and $\oplus$ denotes one of the binary connectives. Obviously, the number of subformulas is linear in the size of Φ, and since each of the logical expressions involved can be expressed in at most four clauses the construction is linear in the size of the formula. For completeness, we give an example of the result of translating a 'triple' $(p_\Psi \leftrightarrow (p_{\Psi_1} \wedge p_{\Psi_2}))$ and a formula $(p_\Psi \leftrightarrow \neg p_{\Psi_1})$ to 3CNF form. We obtain $(\neg p_\Psi \vee p_{\Psi_1}) \wedge (\neg p_\Psi \vee p_{\Psi_2}) \wedge (p_\Psi \vee \neg p_{\Psi_1} \vee \neg p_{\Psi_2})$ and $(p_\Psi \vee p_{\Psi_1}) \wedge (\neg p_\Psi \vee \neg p_{\Psi_1})$, respectively.

It is easy to verify that the original formula allows a satisfying assignment if and only if its associated 3CNF counterpart allows a satisfying assignment. More precisely, any satisfying assignment of the original formula can be (uniquely) extended to a satisfying assignment of its associated 3CNF formula. Note that to obtain a slightly more concise formulation, the equivalence operators may be replaced by implications [139]. Thus satisfiability is maintained as well, but in case the original formula is satisfiable, the resulting formula possibly allows more satisfying assignments.

Observe that if the original formula is a tautology, the resulting CNF formula is merely satisfiable. Since proving a formula to be tautologous is equivalent to proving its negation to be contradictory the tautology problem can still be approached using the construction.

Example 1.1.6 Let us apply the Tseitin construction to Φ_0 (see Example 1.1.1). As stated before, it has apart from the full formula, five subformulas. Introducing a new propositional variable p_{i+2} for subformula (i), we obtain

$$\begin{array}{lllllll}
(p_{\Phi_0} & \leftrightarrow & (p_3 & \vee & p_4)) & \wedge \\
(p_3 & \leftrightarrow & (p_0 & \leftrightarrow & p_5)) & \wedge \\
(p_5 & \leftrightarrow & \neg p_1) & \wedge \\
(p_4 & \leftrightarrow & (p_6 & \wedge & p_1)) & \wedge \\
(p_6 & \leftrightarrow & (p_7 & \rightarrow & p_2)) & \wedge \\
(p_7 & \leftrightarrow & \neg p_0).
\end{array}$$

This formula can be expanded to a CNF formula with 17 clauses which after addition of the unit clause p_{Φ_0} is satisfiability-equivalent to Φ_0. Given a satisfying assignment of Φ_0, for instance $p_0 \wedge p_1$, this can be expanded to the satisfying (partial) assignment $p_0 \wedge p_1 \wedge p_4 \wedge \neg p_5 \wedge p_6 \wedge \neg p_7$ of its associated 3CNF formula. □

Remark: The terminology used may seem somewhat ambiguous. To avoid confusion we stress that 'SAT', 'the SAT problem', 'SAT instance' and 'SAT formula' all refer to the satisfiability problem of propositional logic, in either general form or in CNF. The latter is also referred to as 'CNF formula'.

1.2 The complexity of satisfiability problems

SAT is considered a difficult problem, as no methods are available that solve it 'efficiently'. To make this notion more precise, we rely on *complexity theory*. For a mathematically rigorous treatment of the complexity-theoretic issues involved, see Garey and Johnson [51]. Here we suffice with an informal discussion.

An *algorithm* is a procedure to solve a (well-defined) problem. Thus, for satisfiability problems, given an instance of SAT, an algorithm should be able to determine whether the instance is satisfiable or contradictory. The *worst-case complexity* of an algorithm is defined as the maximal number of operations the algorithm has to perform to solve any instance of a class of problems, expressed as a function f of the size s of the instance. We say that the complexity of the algorithm is $\mathcal{O}(f(s))$. The size of an instance of SAT is usually measured as the number of distinct propositions occurring in it, or as the number of clauses.

Example 1.2.1 Let Φ be a CNF formula consisting of n unit clauses on m variables. Using the truth table approach of the previous section, 2^m distinct solutions have to be evaluated. The complexity of this approach is $\mathcal{O}(2^m)$. An alternative approach is to go through the clauses in sequence. Each unit clause requires fixing its associated variable to its required truth value. As soon as a contradictory assignment is found for a some variable, the algorithm declares the formula a contradiction and it terminates. If no contradictory assignment is found, the algorithm terminates after the last clause has been processed. Then the fixed variables constitute a satisfying assignment. The complexity of this algorithm is $\mathcal{O}(n)$. □

An algorithm is considered *good* if its runs in *polynomial time*, i.e. its worst-case complexity is a polynomial function of the input size. In the example, the second algorithm runs in polynomial time, while the truth table approach runs in *exponential time*. For sufficiently large problems, a polynomial algorithm is more efficient than an exponential algorithm. A problem is considered *easy* if it allows a polynomial time algorithm; if only exponential algorithms are known it is considered *hard*. This is fundamental for complexity theory.

The class $\mathcal{P}$ is defined as the class of problems for which a polynomial time algorithm exists; accordingly, 1CNF formulas are in $\mathcal{P}$. The class $\mathcal{NP}$ is defined as the class of problems for which, given an instance and a particular solution, it can be verified in polynomial time that indeed the solution solves the instance[1]. It follows that $\mathcal{P} \subseteq \mathcal{NP}$. Note that it is easy to see that SAT $\in \mathcal{NP}$, since given an instance of SAT and a satisfying solution, it can be verified in polynomial time that the solution indeed satisfies all clauses. It is not known whether SAT $\in \mathcal{P}$. Or, more generally, a notorious and challenging open problem is whether or not $\mathcal{P} = \mathcal{NP}$. While this problem is still unsolved, it is commonly conjectured that $\mathcal{P} \neq \mathcal{NP}$; i.e. it is suspected that there are problems in $\mathcal{NP}$ that do not allow a polynomial time algorithm. The motivation for this conjecture relies on the notion of *NP-completeness*. To define the class of NP-complete problems, we require the notion of *polynomial reducibility*. A problem P' is polynomially reducible to a problem P, if for any instance of P' an instance of P can be constructed *in polynomial time*, such that by solving the instance of P, also the original instance of P' is solved. Note that then P is at least as hard as P', since P' is in fact a special case of P. A problem P is called *NP-hard* if any problem P' $\in \mathcal{NP}$ is polynomially reducible to P. It is called *NP-complete* if P $\in \mathcal{NP}$ and P is NP-hard. Cook [26] proved that any problem in $\mathcal{NP}$ can be polynomially reduced to a satisfiability problem. Therefore SAT is NP-complete, implying that

- any problem in $\mathcal{NP}$ can be solved as a satisfiability problem;
- if SAT can be solved efficiently, then any other problem in $\mathcal{NP}$ can be solved efficiently.

Since despite extensive research on many notoriously difficult NP-complete problems (including SAT) no polynomial time algorithms have been found, it is conjectured that such algorithms do not exist for these problems. This implies that it is considered unlikely that an algorithm exists that can solve *any* given instance of SAT in *polynomial time*. There are exceptions; for example, the SAT problem on 1CNF (see above) and 2CNF formulas is solvable in linear time (Aspvall et al. [6]). The SAT problem on ℓCNF formulas, $\ell \geq 3$, is in general a hard problem.

[1] If the problem at hand is an *optimization* problem, a *recognition* problem is associated with it. It then must be verified that a solution indeed has a specified objective value.

1.3 Applications of satisfiability problems

Since SAT is the original NP-complete problem it deserves thorough study. By complexity theory, we know that any instance of an NP-hard problem can be modelled as a SAT formula; but more importantly, many NP-hard practical problems can be expressed as SAT problems *in a natural way.* A large number of applications of SAT are listed and referenced in the overview paper by Gu et al. [62] (section 14). In order to give the reader an impression of the expressive strength of propositional logic, we give a few examples below. These examples are chosen due to their relevance for some of the actual applications considered in this thesis. First the design and testing of Boolean circuits is considered. Then, to support the claim that many combinatorial optimization problems have a quite natural SAT encoding as well, we consider as an example the *Frequency Assignment Problem* from mobile telecommunication. It may be noted that such *optimization* problems usually involve an objective function that does not seem to have a 'direct' natural SAT encoding. By putting a bound on the objective function it can be treated as an additional constraint; the optimization problem is then turned into a *feasibility* problem. In Appendix A a construction is discussed to efficiently obtain a CNF formula representing a linear constraint.

1.3.1 Boolean circuit synthesis

The *Boolean circuit synthesis problem* has applications in artificial intelligence, machine learning and digital integrated circuit design [17]. It arises when one needs to construct a logical circuit describing the behaviour of a *black box system.* Suppose we are given a set of sample inputs $\mathcal{X} = \{x_1, \ldots, x_n\}$ for this system and a set of corresponding sample outputs $\mathcal{Y} = \{y_1, \ldots, y_n\}$, where $x_i \in \{0,1\}^m$ and $y_i \in \{0,1\}^k$. The aim is to synthesize a function f, $f : \{0,1\}^m \to \{0,1\}^k$, that describes the behaviour of the black box system with a prescribed accuracy ε, i.e. it must hold that $f(x_i) = y_i$ for at least $(1-\varepsilon)n$ input/output samples. The function f has a sum-of-products form, where the additions are assumed to be modulo 2. Each of the products is called a *disjunct.* The sums and products correspond to logical gates. Given a function f, its corresponding circuit can be represented via a directed acyclic graph; this is discussed in more detail in the next section. The problem of finding a function f with a specified number of disjuncts that accurately describes the system is NP-complete [17]. Note that the classical problem is to minimize the number of disjuncts of a function f that describes the input/output behaviour without errors (i.e. $\varepsilon = 0$). Here we consider the case where the number of disjuncts is specified and we give as an example the *parity function* as described by Crawford et al. [32]. The general case is handled similarly; we refer to Kamath et al. [83, 84].

The parity function has at most m disjuncts, where each disjunct consists of exactly one element. Given a vector $a \in \{0,1\}^m$, it is defined as $f : \{0,1\}^m \to \{0,1\}$, $f(x) = a^T x$. The parity function f computes the parity of a subset of the components of x, namely the subset of the components for which $a_j = 1$. The parity learning problem is now to determine on which bits the parity of $a^T x_i$ is computed, with sufficient accuracy. Or, more formally, given sets of sample inputs $\mathcal{X}$ and sample outputs $\mathcal{Y}$: *find a vector* $a \in \{0,1\}^m$ *such that* $|\{i : a^T x_i \neq y_i\}| \leq \varepsilon n$.

We can encode this as a satisfiability problem in the following way. The m entries of sample input x_i are denoted as x_{ij}, $1 \leq j \leq m$. Let us associate a propositional variable p_j with each a_j, $1 \leq j \leq m$, q_{ij} with x_{ij}, $1 \leq i \leq n$, $1 \leq j \leq m$, and r_i with y_i, $1 \leq i \leq n$. The p_j's are the unknown propositions; the q_{ij}'s and r_i's are known from the sample input/outputs. We associate the value 1 with *true* and 0 with *false*. The parity of each $a^T x_i$ is computed separately. We introduce 'carry'-propositions c_{ij} representing the parity of the first j terms of $a^T x_i$. For all $1 \leq i \leq n$ we have the logical expressions

$$\begin{aligned} c_{i1} &\leftrightarrow (p_1 \wedge q_{i1}), \\ c_{ij} &\leftrightarrow (\neg c_{i,j-1} \leftrightarrow (p_j \wedge q_{ij})), \quad 2 \leq j \leq m, \\ s_i &\leftrightarrow \neg(c_{im} \leftrightarrow r_i). \end{aligned}$$

The reader may want to verify that c_{ij}, $j \geq 2$, is *true* if and only if either $c_{i,j-1}$ is *true* and $a_j x_{ij} = 0$ or if $c_{i,j-1}$ is *false* and $a_j x_{ij} = 1$. It is easy to verify that s_i holds if and only if $a^T x_i \neq y_i$. Introducing subformulas of the above form for each of the sample inputs, we ultimately have n propositional variables s_i of which at most εn may be *false*. This can be expressed as a linear (inequality) constraint, which in turn can be translated to a CNF formula. We refer to Appendix A. If no errors are allowed, the above gives a complete formulation of the parity learning problem. In this case, unit clauses $\neg s_i$ are added to the formula expressing that all input/output patterns are correct. The parity learning problem without errors is in fact polynomially solvable (see Chapter 4). Thus in polynomial time it can be checked whether a parity function exists that describes the system without inaccuracies. If it turns out that no such function exists (this might be due to inaccuracies in the data or to the restricted expressive power of parity functions), either a more complex function can be assumed and tested or some errors might be allowed.

Example 1.3.1 Let us consider a very small example of the parity learning problem. In Figure 1.1 a system with three inputs and one output is depicted. Three sample inputs/outputs are given. Let us now construct the SAT formula associated with the parity-learning problem on 3 bits without errors. Consider the first input/output sample: $x_1 = (1,1,0)$, $y_1 = 0$. The associated formula simplifies to $p_1 \leftrightarrow p_2$ (using that q_{11}, q_{12}, $\neg q_{13}$, $\neg r_1$, $\neg s_1$). Similarly, the SAT formula for the second input/output sample is $\neg(p_2 \leftrightarrow p_3)$ and for the third one p_3. Thus the unique satisfying assignment is $\neg p_1 \wedge \neg p_2 \wedge p_3$, or equivalently, $a = [0,0,1]$. The reader may want to verify that indeed this is an error-free parity function for the given inputs/outputs. □

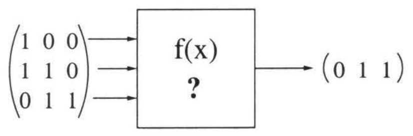


Figure 1.1: Small example of the parity learning problem.

1.3.2 Test pattern generation

To produce a reliable computer system, one must check whether its components are working correctly. Automatic Test Pattern Generation (ATPG) systems distinguish defective components from correct components by generating input sets that – via the corresponding output – indicate that a certain fault is present. Successful ATPG systems are for example PODEM and Socrates [54, 114]. Larrabee [91] proposed to solve the problem of test pattern generation for *single stuck-at* faults in combinational circuits via a SAT problem.

Suppose a piece of hardware implements a Boolean function f with m inputs and k outputs, i.e. $f : \{0,1\}^m \rightarrow \{0,1\}^k$. The topological description of the circuit can be represented via a *directed acyclic graph.* Its internal nodes correspond to the logical gates and fan-out points, its source nodes correspond to the inputs and its sink nodes to the outputs, while its edges correspond to the wiring. Each edge has an associated propositional variable to represent the signal (either 0 or 1) transferred by the corresponding wire. Each node is tagged with a CNF formula describing its input/output behaviour. For example, if an internal node corresponds to an AND gate with two input wires labelled p_i and p_j and a single output wire with the label q_{ij}, it is tagged with the CNF equivalent of $q_{ij} \leftrightarrow (p_i \wedge p_j)$. Taking the conjunction of all CNFs tagged to the internal nodes, a CNF formula describing the input/output behaviour of the circuit is obtained. Assigning truth values to the input variables, the values of the output variables are uniquely determined.

Let us consider systems with a single output. A single stuck-at fault occurs when a single wire is stuck-at a particular value (either 0 or 1). To check whether a specified wire has this fault, a test (input) pattern is required that distinguishes the output of the correct circuit (which is specified by f) and the output of the circuit with the faulty wire. To this end, construct the CNF formula Ψ corresponding to the correct circuit and denote its output variable by r and construct the CNF formula Ψ' corresponding to the circuit with the specified faulty wire; the output variable of the faulty circuit is denoted by r'. Because both circuits will have identical behaviour except at nodes affected by the fault, only the variables that are associated with wires on a path between the faulty wire and the output need to be renamed. Now consider the formula $\Phi = (\Psi \wedge \Psi') \wedge (r \leftrightarrow \neg r')$. A satisfying assignment corresponds to a correct input/output pattern for both circuits, with *different* output. Hence the corresponding input pattern is a test pattern for the specified fault. If the formula is unsatisfiable, the fault cannot be tested.

Example 1.3.2 Suppose we have a simple circuit with four inputs and a single output as shown in Figure 1.2. The function represented by this circuit can be written as $f(x_1, \ldots, x_4) = (x_1 + x_2 + x_1x_2)(x_3 + x_4 + x_3x_4) \pmod 2$. Using the labelling of the wires, the corresponding Boolean formula is

$$\Psi = (q_{12} \leftrightarrow (p_1 \vee p_2)) \wedge (q_{34} \leftrightarrow (p_3 \vee p_4)) \wedge (r \leftrightarrow (q_{12} \wedge q_{34})).$$

Suppose we want to test whether the wire corresponding to the variable q_{12} is stuck-at 1. Then $\Psi' = (r' \leftrightarrow q_{34})$, hence

$$\Phi = \Psi \wedge (r' \leftrightarrow q_{34}) \wedge (r \leftrightarrow \neg r'),$$

which yields the test pattern $\neg p_1, \neg p_2, p_3, p_4$: the correct circuit gives output $\neg r$, while the faulty circuit has output r. Note that this test patterns is not unique; Φ allows more satisfying assignments. □

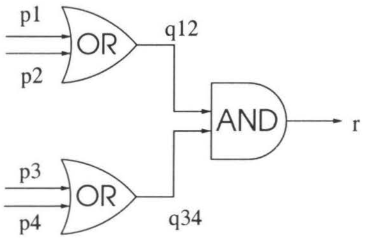


Figure 1.2: Boolean circuit corresponding to the example.

1.3.3 The Frequency Assignment Problem

The Frequency Assignment Problem (FAP) occurs in practice when a network of radio links has been established. Frequencies need to be assigned to the radio links such that communication via these links does not interfere. Interference generally occurs when the same or close frequencies are assigned to links that are situated near each other. Various algorithms have been applied to the FAP; see for overviews Tiourine et al. [125], Cabon et al. [21]. The problem can be stated as follows.

Given a set of radio links $\mathcal{L}$, a set of frequencies $\mathcal{F}$ and a set of interference constraints, assign each radio link a frequency, without violating any interference constraint.

A second objective might be to use as few distinct frequencies as possible in order to minimize usage of the frequency spectrum. An interference constraint is a triple (l, k, d_{lk}), where $d_{lk} \geq 0$ is the minimum distance required between the frequencies assigned to radio links l and k. In [138] various mathematical models for this problem are developed. Let us construct a valid SAT encoding. We introduce proposition letters p_{lf} and q_f for all $l \in \mathcal{L}$, $f \in \mathcal{F}$; p_{lf} holds if and only if frequency f is assigned to link l, q_f expresses that frequency f is assigned to at least one link. Letting $F_{\max}$ be the maximal number of frequencies to be used, the FAP is expressed as follows.

First, each link must be assigned a frequency.

$$\bigvee_{f \in F} p_{lf}, \; l \in \mathcal{L}.$$

The interference constraints can be modelled as implications:

$$p_{lf} \to \neg p_{kg}, \; l, k \in \mathcal{L}, \; f, g \in \mathcal{F} \text{ such that } |f - g| \leq d_{lk}.$$

Finally, to express the objective to use no more than $F_{\max}$ frequencies, we have the implications

$$p_{lf} \to q_f, \; l \in \mathcal{L}, \; f \in \mathcal{F},$$

with the additional constraint that *at most $F_{\max}$ propositions q_f must be true.*

Note that this constraint is not in CNF. However, it can be expressed as a linear inequality constraint, which in turn can be translated to a CNF formula; see Appendix A. In Chapter 6 an algorithm for the FAP is developed that does not require the non-CNF constraint to be transformed to CNF first.

Example 1.3.3 In Figure 1.3 a small FAP is depicted. There are four links that have either three or four frequencies available, which are listed below the link number. If two links are connected, an interference constraint concerning these links must be satisfied. The edge labels specify the minimal required frequency distances. For example, the edge connecting nodes 1 and 2 indicates that the frequencies assigned to links 1 and 2 must be more than four apart. This gives rise to the following implications: $(p_{1,2} \rightarrow \neg p_{2,2}) \wedge (p_{1,2} \rightarrow \neg p_{2,4}) \wedge (p_{1,4} \rightarrow \neg p_{2,2}) \wedge (p_{1,4} \rightarrow \neg p_{2,4}) \wedge (p_{1,4} \rightarrow \neg p_{2,7}) \wedge (p_{1,9} \rightarrow \neg p_{2,7}) \wedge (p_{1,9} \rightarrow \neg p_{2,9})$. To model all the interference constraints, 27 implications are required, involving 14 p_{lf} variables. For more details on this example, see Section 6.5.2. □

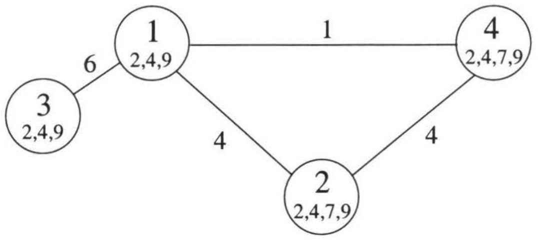


Figure 1.3: Small FAP instance.

1.4 Solving satisfiability problems

We mention a number of algorithms and briefly discuss methods for comparing the performance of algorithms. Then three types of algorithms are evaluated empirically to identify hard classes of SAT.

1.4.1 Algorithms

As stated before, any satisfiability problem can be solved by *brute force*, since via evaluating all distinct truth value assignments (of which there are finitely many) to the variables the problem is solved. For a problem with m variables, 2^m assignments need to be evaluated. Even for small m this is intractable, since this number grows exponentially with m. Unfortunately, SAT being NP-complete, it is unlikely that an algorithm exists that does much better than the brute force algorithm. Still, many SAT problems can often be easily solved when the proper algorithm is applied, even if its worst-case complexity is exponential. Therefore, many SAT algorithms have been developed; for a comprehensive overview, see the article by Gu, Franco, Purdom and Wah [62]. Here we mention a number of the best known algorithms. Later on we try to provide the reader with some global overview as to the strength of a subset of these algorithms.

Most importantly, we can distinguish between *complete* and *incomplete* algorithms.

Complete algorithms give a definite answer as to the unsatisfiability of a given formula. Examples of complete algorithms are

- The 'brute-force method' c.q. truth–tables;
- DPLL (Davis, Putnam, Loveland, Logemann [36, 37]), which is a *depth-first* search algorithm which implicitly enumerates all solutions, enhanced with unit resolution and monotone variable fixing (see also Section 3.3);
- Resolution (Robinson [110]), which is similar to the classic Davis-Putnam algorithm;
- Binary Decision Diagrams (Bryant [18]), using which *all* solutions of a given instance of SAT can be determined;
- Stållmarck's algorithm [122] (see also [59]), which is a *breadth-first* search algorithm (as opposed to the DPLL algorithm);
- Analytic tableaux (see e.g. [34]), which can be interpreted as being a variant of the DPLL method;
- Branch and bound c.q. Branch and cut, which are essentially enhanced versions of depth-first search algorithms (see e.g. Blair et al. [11], Hooker and Fedjki [73]).

Incomplete algorithms do not give a definite answer in all cases. Usually these are designed to find solutions quickly; if a solution is found, the formula is declared satisfiable and the algorithm terminates successfully. However, if it fails to find a solution, no conclusions can be drawn. Well-known incomplete algorithms for satisfiability are

- Local search algorithms in various guises, such as introduced by Selman et al. [115, 117] (GSAT, WalkSAT), Gu [60] and Resende and Feo (GRASP) [109];
- Gu's global optimization algorithms [61];
- An interior point potential reduction approach by Kamath et al. [82, 83, 84];
- Many other local search algorithms such as tabu search, simulated annealing and evolutionary algorithms (see articles in the books [42, 79]).

In the above list, we do not include incomplete algorithms that are used as subroutines in complete algorithms, such as *unit resolution* (which is complete for Horn CNF formulas [41]).

1.4.2 The relative performance of algorithms

Given the amount of different algorithms, it can be a difficult decision as to which algorithm to use for solving a specific SAT instance, in particular when its solution is required at the shortest possible notice. As the very existence of all these different algorithms seems to suggest, there is in general no useful theoretical foundation available to prefer one algorithm to another *in practice*. For example, while theoretically a complete algorithm seems to be preferable over an incomplete one at any time, in practice (incomplete) local search algorithms can be very effective (albeit on satisfiable instances only) and thus they are extremely useful.

In the literature, several methods to analyze and compare the performance of algorithms are proposed. We mention *worst-case* analysis, *probabilistic* analysis and *polynomial simulation*. While we emphasize that each of these methods can give valuable insights into understanding why algorithms do or do not perform well on specific instances, below we argue that they are not sufficient when it comes to practical satisfiability solving.

Worst-case analysis

While all complete algorithms require exponential time in the worst case, in a number of cases some distinction can be made: for instance, Schiermeyer [112] has shown that 3CNF formulas can be solved in $\mathcal{O}(1.497^m)$ operations rather than in 2^m operations (improving on earlier upper bounds by Monien and Speckenmeyer [102] and Kullmann [89]). However, the worst case performance of an algorithm is usually only rarely achieved and algorithms with an extremely bad worst-case complexity might in practice often be very effective. Most notably, incomplete algorithms do not even terminate in the worst case.

Probabilistic analysis

A probabilistic analysis of an algorithm gives rise to the notion of *average-case* performance (as opposed to worst-case performance). Thus it provides an indication of the time in which an instance can be expected to be solved. To this end, assumptions on the structure of the SAT instances under consideration need to be made. Usually, the instances are assumed to be randomly generated using some specific probabilistic model. Finding a representative probabilistic model is not trivial. Using the 'wrong' model may yield misleading results (as argued by Franco and Paull [47]). Also to our knowledge probabilistic analyses have been carried out for substantially simplified algorithms only. It appears that for more involved algorithms the analysis is too complex to handle with the currently available techniques. Specifically, for most algorithms mentioned in the previous section a probabilistic analysis is not available. Most results in this area are of a reassuring nature, stating that 'easy' instances can be easily solved (i.e. in average polynomial time) by a simple algorithm. An overview of results is contained in Gu et al. [62] (section 12).

Polynomial simulation

Polynomial simulation [28] is a tool to qualify the relative complexity of algorithms. By a *proof of size s* we mean the number of proof steps that has to be done to prove a formula contradictory (or satisfiable). Algorithm I is said to polynomially simulate or *p-simulate* algorithm II if there exists a polynomial P such that for every proof of size s using algorithm II, a corresponding proof (of the same formula) of size at most $P(s)$ using algorithm I can be constructed. An approach to prove that algorithm I p-simulates algorithm II is to show that any reasoning step that can be done using algorithm I can be efficiently simulated using algorithm II. If an algorithm p-simulates another it is considered at least as good as the other. Two methods are *polynomially equivalent* if they p-simulate one another. They are said to be *polynomially incomparable* if they mutually do not p-simulate one another. The usual way to prove that algorithm I does not p-simulate algorithm II is by specifying a particular class of formulas that can be proved in polynomial time by II, while I requires exponential time. For an overview of results in this area, see Urquhart [132].

This notion is of particular use in the context of the *existence* of proofs of a certain size. In this thesis, the focus is on effective proof *searching*. Although the notion of polynomial simulation can be used here as well, we consider it less appropriate since the existence of a short proof is no guarantee that it can be found efficiently. In addition, even when an algorithm p-simulates another (and consequently it is considered better), in practice it may perform substantially worse, since it is not clear how to exploit its stronger reasoning capabilities (for example, *extended resolution* [129] is in theory an extremely powerful proof system, but there are no practical implementations available). Finally, observe that an incomplete algorithm can never p-simulate a complete algorithm.

Benchmarking

For lack of a satisfactory theoretical method, we resort to a more *ad hoc* way of comparing algorithms in order to get a better intuition on their performance. The most commonly used evaluation method is to run the algorithms on a set of 'standard' benchmarks. Comparing the results should provide the user with an intuition based on which an algorithm can be chosen for 'new' instances. The choice of benchmarks is obviously rather arbitrary. In the SAT community there seems to be consensus that the DIMACS suite [128] (which contains a collection of benchmarks stemming from various sources, both practical and theoretical) and random 3CNF formulas are reasonably 'representative' benchmarks.

The incentive for using random instances is twofold. It is easy to generate as many instances as considered appropriate, and algorithms performing well on hard random instances will arguably be effective on other instances as well. The question is how to ensure that the randomly generated instances are sufficiently difficult. It has been shown empirically that difficult random instances can be obtained using the *constant clause length* model: 3CNF formulas with m variables and αm clauses are hard when $\alpha \approx 4.3$ (see Mitchell et al. [101], Crawford and Auton [31]). This is known as the *threshold phenomenon.* For the mentioned clause-variable ratio it has been demonstrated empirically that roughly half of the instances are satisfiable and half are contradictory. For theoretical results in this area we refer to [62].

Note that the *size* of the instances plays a crucial role in the overall picture obtained. Small sized instances are often easy for any algorithm, while if instances get too large none of the algorithms might be capable of solving them. On the other hand, this also challenges the researcher to come up with algorithms that *are* capable of handling such large instances. Another issue is that the quality of the implementations of the algorithms and of the hardware they are executed on are of great influence on solution times, and thus should be taken into consideration when evaluating the performance.

Ultimately, it is (merely) *assumed* that the results obtained give a representative view of the effectiveness of the algorithms over *all* instances of SAT. In any case, the results gathered in the next section are used as a guidance for our research, since they reveal certain gaps in our SAT solving capabilities.

1.4.3 Empirically comparing three types of algorithms

In Table 1.2 a *very rough* indication of the effectiveness of three types of algorithms on standard benchmarks is given. As reference algorithms we restrict ourselves to algorithms of three different types: *depth-first*, *breadth-first* and *local* search. In the literature on SAT solving, the first and the third type of algorithm are the most frequently used and thus seem to be considered the most effective[2].

- SATO3.0 (Zhang [141]) for the depth-first state-of-the-art. This implementation includes intelligent backtracking and the addition of newly derived clauses (see also [120]; more details on the DPLL algorithm and enhancements are given in Chapter 3).
- Böhm's solver (see Böhm and Speckenmeyer [12]), which is an implementation of a depth-first search algorithm *without* intelligent backtracking and the addition of newly derived clauses. This solver is an improved version of the solver winning a SAT competition in 1992 [19].
- HeerHugo, which is an implementation of a breadth-first search algorithm by Groote at CWI (see [59]). This implementation is inspired by Stållmarck's algorithm [67, 122]. It is enhanced with many reasoning capabilities. It appears that this type of algorithm is largely overlooked in the literature on practical satisfiability solving. As it appears to be remarkably effective on many sets of benchmarks it is included here.
- A local implementation of a local search algorithm with *adaptive clause-weights*. Let us briefly describe this algorithm. To each individual clause a weight is assigned. Initially, all weights are set to one. Starting from a random truth value assignment, in each iteration a single variable is '*flipped*' (i.e. set from $true$ to $false$ or vice versa). The variable to be flipped is chosen by cycling through the set of clauses. In each iteration the variables of the *first* unsatisfied clause are chosen as candidates (this is similar to the WalkSAT approach [115]). The variable that yields the largest increase in the sum of weights of the satisfied clauses is flipped. If the increase is negative, the weights of the currently unsatisfied clauses are increased until the increase becomes positive.

We used the following benchmarks to test the algorithms. For more details on the instances we refer to Trick [128]. Here we give only some brief comments on the instances[3].

- The `aim` [4], `dubois` and `pret` instances are constructed 3CNF instances. The latter two turn out to belong to a class of polynomially solvable instances; see Chapter 4. The size of these problems ranges from 20 to 200 variables and from 60 to 1200 clauses.
- The `ii` (*inductive inference*, [83, 84]) and `par*` [32] instances stem from the Boolean function synthesis problem as described in Section 1.3.1. Sizes range up to thousands of variables and clauses.

[2]Note that on specific applications involving non-CNF SAT problems, BDDs are reported to outperform the more classical approaches; see Puri and Gu [107], Uribe and Stickel [130].

[3]`ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/`.

- The `bf` and `ssa` instances are generated from circuit fault analysis (see also Section 1.3.2). The first arise from checking 'bridge-faults', the second from 'single stuck-at' faults. The sizes range to thousands of variables and clauses.

- The `jnh` instances are randomly generated instances, using the *random clause-length* model: as the name suggests, generating instances using this model clauses of different lengths can occur, as opposed to the constant clause-length model. These instances are fairly small, consisting of at most a few hundred variables and up to a thousand clauses.

- The `hole` instances are instances of the notorious *pigeon hole principle*. Small-sized formulas are usually easily solved. Here we refer to moderately sized instances of $h \approx 15$ holes. The number of variables is $\mathcal{O}(h^2)$, the number of clauses is $\mathcal{O}(h^3)$. It is well known that efficient methods for these instances exist, for example by means of *extended resolution* (Cook [27]) or *cutting planes* ((another) Cook et al. [29]) (see also Buss [20]). On the other hand, Haken proved that no polynomial sized resolution proof exists [63]. More details on the pigeon hole principle can be found in Section 5.6.2.

- The `3SAT` instances are hard random 3SAT instances in general, generated according to the constant clause-length model. We refer to moderately sized instances of 300 variables and 1290 clauses and to relatively large sized instances of 600 variables and 2580 clauses.

- The `celar` instances are SAT encodings of large Frequency Assignment Problems; see Section 1.3.3. These instances have up to almost 20000 variables and over 400000 clauses.

In Table 1.2 we distinguish four degrees of effectiveness of the algorithms. A '++' means that the algorithm on average solves the instances easily in very short times; say, at most 5 seconds. A '+' denotes that the instances are solved on average in a matter of minutes to an hour at the most. A '+/–' implies that the algorithm is capable of solving the instances under consideration, but this requires in the order of hours up to a day. Finally, a '—' means that the algorithm does not solve the instances within a day. The results are assumed to be obtained on a representative computer. In two cases (on the large `celar` instances) SATO terminated with an error message; hence the '?' in the table. The '*' is to emphasize the fact that, while many satisfiable 600-variable 3CNF instances were solved by the local search algorithm, we cannot be sure that *all* the satisfiable instances were solved (since we have no means to obtain a proof of unsatisfiability of such formulas, as indicated in the table).

1.4.4 Remarks and research purposes

Before evaluating the results gathered in Table 1.2 let us stress again that it is intended to give a very rough indication of the effectiveness of the types of algorithms used. Considering the table, we first observe that many problems are handled in short time by at least one of the algorithms. Thus it appears that many classes of instances (at least at the sizes considered) do not pose any real challenge for the current state-of-the-art algorithms

		Complete Algorithms			Incomplete
name	SAT/UNSAT	SATO3.0	Böhm	HeerHugo	local search
`aim-50`	SAT	++	+	++	++
	UNSAT	++	+	++	—
`aim-100`	SAT	++	+/–	++	++
	UNSAT	++	+/–	++	—
`aim-200`	SAT	++	+/–	++	++
	UNSAT	++	+/–	++	—
`dubois`	UNSAT	++	+/–	++	—
`pret`	UNSAT	++	+/–	++	—
`ii8`	SAT	++	++	++	++
`ii16`	SAT	+	+/–/—	+/–/—	++
`ii32`	SAT	+	+/–/—	+/–/—	++
`par8`	SAT	++	++	++	++
`par16`	SAT	+	+/–	—	+
`par32`	SAT	—	—	—	—
`bf`	UNSAT	++	+/–	++	—
`ssa`	SAT	++	+/–	++	++
	UNSAT	++	+/–	++	–
`jnh`	SAT	++	+	+	++
	UNSAT	++	+	+	—
`hole`	UNSAT	—	—	—	—
`3SAT-300`	SAT	+	+	—	++
	UNSAT	+	+	—	—
`3SAT-600`	SAT	—	—	—	$+^*$
	UNSAT	—	—	—	—
`celar`	SAT	?/+/–	—	—	+

Table 1.2: Effectiveness of algorithms on selected benchmarks.

and implementations, both due to the quality of the algorithms and implementations and the chilling speed of contemporary computers. In this respect we observe the following.

- It appears that *almost* any satisfiable instance can be rather easily solved using a local search algorithm. This suggests that given an instance of SAT it is always worthwhile to attempt solving it using a local search algorithm first. Obviously, the local search algorithm being an incomplete method solely capable of proving satisfiability, it fails on any unsatisfiable instance.

- The enhanced DPLL algorithm implemented in SATO performs much better than the 'plain' algorithm in Böhm's solver on most instances. It appears that HeerHugo in several cases outperforms Böhm's solver, but SATO3.0 has gained a reasoning strength that is at least comparable to that of HeerHugo.

However, with our increasing ability to solve SAT problems, the size of instances we want to solve grows just as explosively. Indeed, over the last ten years or so the size of non-trivial practical SAT problems that can be solved grew from problems with less than 100 variables to ones involving over 10,000 variables [116]. One could argue that the ever-increasing speed of computers will facilitate solving larger instances. On the other hand, due to the exponentiality of the methods and the fact that even now (relatively small) instances exist that are still (too) hard to solve, research is required as well to improve and to design algorithms and implementations. The focus in this thesis is on algorithms. The research is mainly motivated by the following observations.

- Unsatisfiable random 3SAT formulas on the threshold are (indeed) hard. It seems that for this kind of benchmark intelligent backtracking does hardly improve the performance of the DPLL algorithm. In Chapter 3 we try to improve the performance of the DPLL algorithm by deriving new branching rules.

- The `par32` instances are not solved by any of the algorithms. In fact, solving these instances was posed as a challenge by Selman et al. [116]. In Chapter 4 we develop an extension of the DPLL algorithm which, in combination with a preprocessing method involving linear programming, is capable of solving these instances in a matter of minutes.

- As stated before, the *pigeon hole formulas* are very hard for the algorithms considered. In Chapter 5 we show that a proof of their unsatisfiability can be computed in polynomial time using *semidefinite programming*. Using this approach, no problem-specific additional information is required, whereas the construction of other polynomial size proofs requires explicit use of problem-structure.

- Satisfiable formulas with thousands of variables are often rather easily solved by local search approaches (and in many cases by complete methods as well). Solving *unsatisfiable* instances of the same sizes is in general far beyond the current state-of-the art. Thus it appears that more research is required for proving unsatisfiability efficiently. An attempt in this direction was made by Franco and Swaminathan [48]. Other attempts are made in the Chapters 4 and 5 where linear and semidefinite programming formulations are given that express sufficient conditions for unsatisfiability.

- Finally, the `celar` instances are hard due to their sheer size. In Chapter 6 a heuristic algorithm is developed that exploits a quite concise model. This algorithm provides feasible solutions in very short times.

2

Models And Relaxations

We discuss linear and nonlinear models and approximations of SAT problems. Weighted elliptic approximations are introduced which play a key role in this thesis.

2.1 Introduction and notation

Many algorithms for SAT essentially operate on the symbolic model introduced in Section 1.1, such as resolution and the Davis-Putnam algorithm or its variant known as DPLL. In this thesis we are interested in making use of mathematical models involving binary and/or continuous variables. In Table 2.1 a brief overview of the models considered is given. In the next sections the models are specified.

	Linear models	Nonlinear models
Exact	Binary Linear Programming	Smooth Continuous Functions
Inexact	Linear Relaxations	Elliptic Approximations Semidefinite Relaxations

Table 2.1: Models for SAT problems

We will be mainly interested in making use of *inexact, nonlinear models* to obtain effective algorithms (both complete and incomplete) for various classes of SAT.

Before considering the various models, let us introduce some notation. We associate the value -1 with *false* and the value 1 with *true*. Sometimes we use the value 0 as well to refer to *false*; this will be clear from the context and cause no ambiguities. With each propositional variable p_i, we associate a $\{-1, 1\}$ variable x_i, such that $x_i = 1$ if and only if p_i is *true*. Thus we have a one-to-one correspondence between $\{-1, 1\}$ vectors x and truth value assignments to the propositional variables. In this thesis we will feel free to refer to both as *assignments*. In particular, if x is called a satisfying assignment of a formula Φ, then this means that the corresponding truth value assignment to the propositional variables satisfies Φ. Satisfying assignments are also referred to as *solutions*. *Unsatisfying* assignments are also referred to as *contradictory* assignments.

The models introduced in this chapter rely on a matrix representation of CNF formulas. Let Φ be a CNF formula containing n clauses and m distinct variables. Its associated *clause-variable* matrix A has a row for each clause and a column for each propositional variable, hence $A \in \mathbb{R}^{n\times m}$. The entry of matrix A in row k, column i is denoted by a_{ki}. Recalling the representation of a clause (1.1), a_{ki} is defined as follows:

$$a_{ki} = \begin{cases} 1 & \text{if } i \in I_k, \\ -1 & \text{if } i \in J_k, \\ 0 & \text{if } i \notin I_k \cup J_k. \end{cases} \tag{2.1}$$

Here it is assumed that the propositional variables and the clauses are numbered consecutively from 1 to m, and from 1 to n respectively.

Finally, by $\sigma(l_i)$ we denote the number of occurrences of a literal l_i in a formula Φ, i.e.

$$\sigma(p_i) = |\{k \mid p_i \in \mathbf{C}_k\}| = |\{k \mid i \in I_k\}| = |\{k \mid a_{ki} = 1\}|; \tag{2.2}$$

$$\sigma(\neg p_i) = |\{k \mid \neg p_i \in \mathbf{C}_k\}| = |\{k \mid i \in J_k\}| = |\{k \mid a_{ki} = -1\}|. \tag{2.3}$$

Example 2.1.1 The clause-variable matrix associated with the CNF formula

$$\Phi_1 = (p_1 \vee \neg p_3 \vee \neg p_5) \wedge (p_2 \vee \neg p_3 \vee \neg p_4 \vee p_5) \wedge (\neg p_1 \vee \neg p_4)$$

is given by

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

The number of occurrences of the separate literals can easily be extracted from A, for instance $\sigma(p_5) = \sigma(\neg p_5) = 1$ (since the fifth column of A contains exactly one '1' and exactly one '-1'). □

2.2 Satisfiability and linear programming

2.2.1 An integer linear programming formulation

It is well known that with the SAT problem on a CNF formula Φ a binary feasibility problem can be associated (see e.g. Hooker [72]). A clause $\mathbf{C}_k$ can be represented by a linear inequality in the following way:

$$\sum_{i\in I_k} x_i - \sum_{j\in J_k} x_j \geq 2 - \ell(\mathbf{C}_k).$$

Using the clause-variable matrix A, this can be rewritten as $a_k^T x \geq 2 - \ell(\mathbf{C}_k)$, where a_k^T denotes row k of A. It is easy to verify that for a $\{-1,1\}$ vector x, $a_k^T x \geq 2 - \ell(\mathbf{C}_k)$ if and only if the corresponding truth value assignment to the propositional variables p_i satisfies clause $\mathbf{C}_k$. The minimal value of $a_k^T x$ equals $-\ell(\mathbf{C}_k)$ and it is attained if and only if $x_i = -1$ for all $i \in I_k$ and $x_i = 1$ for all $i \in J_k$; as soon as *at least one* $x_i = 1$ for some $i \in I_k$ or $x_i = -1$ for some $i \in J_k$, $a_k^T x \geq 1 - (|I_k \cup J_k| - 1) = 2 - \ell(\mathbf{C}_k)$.

Hence the integer linear feasibility problem (ILP) of the satisfiability problem can be stated as

$$(\text{IP}_{SAT}) \quad \text{find } x \in \{-1,1\}^m \text{ such that } Ax \geq b,$$

where $b \in \mathbb{R}^n$, with $b_k = 2 - \ell(\mathbf{C}_k)$, $1 \leq k \leq n$.

2.2.2 Solving (IP$_{SAT}$)

The most common approaches to solve integer linear programming problems include *branch and bound* and *branch and cut* algorithms. The integrality constraints are first relaxed to linear constraints (i.e. $x \in \{-1,1\}^m$ is relaxed to $-e \leq x \leq e$; e denotes the all-one vector of length m). The resulting linear program is known as the *LP relaxation.* We refer to the relaxation of (IP$_{SAT}$) as (LP$_{SAT}$). The LP relaxation can be solved using the simplex method or an interior point method (the latter of which runs in polynomial time [87]). If it is infeasible, then the original ILP must also be infeasible, while if the solution to the LP relaxation is a $\{-1,1\}$ vector, it is feasible in the ILP formulation as well; in both cases we are done. Furthermore, in specific cases rounding schemes are available to efficiently round the (fractional) feasible solution to a feasible integer solution; an example is discussed in the next section. Additional classes of SAT that are solvable in polynomial time using LP can be found in the literature; see Conforti and Cornuéjols [25], Chandru and Hooker [23] (see also Schlipf et al. [113]).

Unfortunately, in general the solution of the LP relaxation will be fractional and there does not exist an efficent procedure to obtain a feasible integer solution using the fractional solution (note that the existence of such a procedure would in fact imply that $\mathcal{P} = \mathcal{NP}$). In particular, the LP relaxation may be feasible, while the original ILP is infeasible. If the LP relaxation is feasible, usually one or a combination of the following techniques is applied:

1. *Branch*: a variable is chosen and two new LP relaxations are created: in one the variable is fixed at -1, in the other at $+1$. The (lower dimensional) LP relaxations are solved again.

2. *Cut*: the LP relaxation is tightened by adding *cuts* and it is solved again. A cut is a linear inequality that is redundant in the ILP formulation, but it cuts off a part of the feasible region of the LP relaxation.

These steps are repeated recursively until a feasible solution of the ILP is found, or until it can be decided that no solution exists. In general, such procedures will be effective only when the LP relaxations are sufficiently *tight*. An LP relaxation is called tight if the set of vertices of its feasible region coincides with the set of solutions of (IP$_{SAT}$). Since LP algorithms provide a vertex solution, a tight LP relaxation guarantees that any solution to the LP is a solution to the ILP as well. Unfortunately, the LP relaxation of (IP$_{SAT}$) is weak. Note that $b_k \leq 0$ for any clause $\mathbf{C}_k$ of length two or more, hence the trivial all-zero solution is always feasible when no unit clauses are present. In addition, since we are dealing with a feasibility problem, there is no natural objective function available to add to (LP$_{SAT}$) in order to compute useful bounds. Thus in the absence of unit clauses (all of which can be processed in linear time [41] to reduce problem size) formulations of this type can be of use only to find incumbent solutions.

To better suit the LP approach towards proving unsatisfiability, the LP relaxation can be tightened by adding cuts. Hooker [71, 72] shows that the resolution algorithm can be interpreted as the addition of cutting planes. Thus, as follows from the completeness of resolution [110], cutting planes give a complete method for solving SAT problems. Although the cutting plane approach is theoretically powerful [29] (see also Section 5.6.2), it is not entirely clear in practice how to construct strong cutting planes. Besides, exponentially many cutting planes may be required to obtain a tight LP relaxation. Several studies on the application of LP in a branch and bound or branch and cut environment to obtain effective SAT algorithms are available in the literature (e.g. Blair et al. [11], Hooker [71, 72], Hooker and Fedjki [73], Jeroslow and Wang [77]). The overall conclusion seems to be that although LPs do help in reducing the size of the search trees, they are mainly useful for finding solutions quickly on relatively small problems. However, local search algorithms appear to be more effective in this respect. This is mainly due to the fact that the complexity of solving a single LP is (at least) $\mathcal{O}(3.5^m)$, whereas one iteration in a local search algorithm usually requires linear time at the most. In practice, for determining unsatisfiability in non-trivial cases linear programming does not appear to be useful at all. In fact, in a recent paper by Selman et al. [116] it is posed as a challenge to show that LP approaches can be made practical for SAT solving. A successful application of LP to speedup SAT solving on a particular kind of benchmarks is discussed in Chapter 4 of this thesis. The LP formulation introduced there models a sufficient condition for unsatisfiability.

2.2.3 A class of SAT solvable using linear programming

Let us now consider an alternative LP formulation of SAT problems that yields a sufficient condition for satisfiability. Consider the following primal and dual formulation.

$$(\mathrm{LP}_{aut}) \quad \begin{array}{rl} \max & s \\ \text{s.t.} & Ax \geq se, \\ & -e \leq x \leq e. \end{array} \qquad\qquad (\mathrm{LD}_{aut}) \quad \begin{array}{rl} \min & e^T|z| \\ \text{s.t.} & A^T w = z, \\ & e^T w = 1, w \geq 0. \end{array}$$

Denote the optimal value of (LP_{aut}) (LD_{aut}) by $opt(\mathrm{LP}_{aut})$ $(opt(\mathrm{LD}_{aut}))$. For linear programming problems *strong duality* holds. It is easy to verify that (LP_{aut}) and (LD_{aut}) are both feasible, implying that $opt(\mathrm{LP}_{aut}) = opt(\mathrm{LD}_{aut})$ and that optimal solutions x, s and w, z exist. Observe that a valid lower bound of zero on the optimal value of (LP_{aut}) is obtained by taking $x = 0$, $s = 0$. If this lower bound is not optimal, then the corresponding formula must be satisfiable, as stated in the following theorem.

Theorem 2.2.1 *Let Φ be a propositional formula with associated LP formulation (LP_{aut}). If $opt(LP_{aut}) > 0$ then Φ is satisfiable. A satisfying assignment is obtained by taking $y = sgn(x)$, where x denotes the optimal solution of (LP_{aut}).*

Proof: Let (x, s) be the optimal solution of (LP_{aut}). The fact that $a_k^T x \geq s > 0$ implies that $a_{ki}x_i > 0$ for some $1 \leq i \leq m$. From this we conclude that $a_{ki}\mathrm{sgn}(x_i) = 1$. It follows that $a_k^T y \geq 1 - (\ell(\mathbf{C}_k) - 1) = b_k$, for all $1 \leq k \leq n$. □

Using the dual formulation (LD_{aut}) we have an easy corollary. Note that the optimal value of (LD_{aut}) can be zero only if $A^T w = 0$ for some nonnegative $w \neq 0$.

Corollary 2.2.2 *If there is no $w \geq 0$, $w \neq 0$, such that $A^T w = 0$, then the corresponding formula is satisfiable.*

Hence if there is no (nonnegative and nonempty) combination of clauses that yields the trivial linear inequality $0 \geq 0$, then Φ must be satisfiable. This is (obviously) a severe constraint that cannot be expected to be satisfied often in SAT formulas. We observed empirically that on random 3SAT instances with a small clause/variable ratio (say, $n \leq 1.5m$) satisfying assignments can be generated in this way.

If the optimal solution $x \neq 0$ (even if $s = 0$), it is called a *linear autarky* [90], which is a special case of *autarkness*. This notion was first introduced by Monien and Speckenmeyer [102].

Definition 2.2.3 (Autarkness) *A partial assignment of truth values is called an* autark assignment *of a formula Φ, if under this assignment Φ reduces to a formula Ψ with $\Psi \subset \Phi$.*

By $\Psi \subset \Phi$ we mean that Ψ is a proper subset of Φ in the sense that Ψ contains clauses that are also in Φ only. It holds that Φ and Ψ are satisfiability-equivalent. If Ψ is satisfiable with an assignment x, then Φ is satisfiable by x extended with the autark assignment. On the other hand, if Φ is satisfiable, then the satisfying assignment is obviously also valid for Ψ. Note that autarkness is a direct generalization of *monotone variable fixing*. If a certain variable p occurs only unnegated, then the partial assignment p is an autarky.

Let us now define the linear autarky, which was introduced by Kullmann [90], generalizing on a theorem first proved in [99].

Definition 2.2.4 (Linear autarky) *A vector $x \neq 0$, $x \in \mathbb{R}^m$, with $Ax \geq 0$ is called a* linear autarky.

We verify that x gives rise to the autark assignment $y = \text{sgn}(x)$.

Theorem 2.2.5 *The assignment $y = sgn(x)$ is autark.*

Proof: Consider a single inequality:

$$a_k^T x = \sum_{i \in I_k} x_i - \sum_{i \in J_k} x_i \geq 0.$$

There are two possibilities:

1. $x_i = 0$ for all $i \in I_k \cup J_k$.
2. $x_i \neq 0$ for some $i \in I_k \cup J_k$.

Similar to the proof of Theorem 2.2.1, the second possibility implies that $a_k^T \text{sgn}(x) \geq b_k$. Consequently, all clauses with property 2 are satisfied by y, while the other clauses remain untouched. □

Kullmann [90] shows that using the concept of linear autarkies, 2SAT and HornSAT are solvable in polynomial time by linear programming. In this thesis the concept of linear autarky occurs a number of times.

2.3 Nonlinear models of SAT problems

As suggested in the previous section, it is unlikely (unless $\mathcal{P} = \mathcal{NP}$) that in general one can efficiently construct continuous linear models exactly representing SAT problems. However, using nonlinear formulations, exact models can be constructed. Let us first review a general scheme for obtaining smooth nonlinear models of satisfiability problems as given in [97, 98]. In principle the transformation techniques are not restricted to CNF formulas; general Boolean formulas could be handled. Here we will specifically consider CNF formulas: we discuss some examples and subsequently we explain how nonlinear models are used in this thesis.

A *transform* $\mathcal{T}$ of a propositional formula Φ is defined as a mapping $\mathcal{T} : [-1,1]^m \to [0,1]$ with the property that for all $\{-1,1\}$ vectors x it holds that

$$\begin{aligned} \mathcal{T}(x) \geq th_{true} \quad &\Leftrightarrow \quad x \text{ is a satisfying assignment of } \Phi; \\ \mathcal{T}(x) \leq th_{false} \quad &\Leftrightarrow \quad x \text{ is a contradictory assignment of } \Phi, \end{aligned}$$

where th_{true} and th_{false} are threshold values that are determined by the choice of $\mathcal{T}$. The satisfiability problem could be solved via the following equivalent mathematical programming problem.

$$(\mathrm{P}_{\mathcal{T}}) \quad \begin{array}{ll} \max & \mathcal{T}(x) \\ \text{s.t.} & x \in \{-1,1\}^m. \end{array}$$

If the optimal objective value of $(\mathrm{P}_{\mathcal{T}})$ is below th_{true}, then Φ is a contradiction. The integrality conditions may be relaxed to obtain a *linear* or a *spherical* relaxation. Such relaxations can also be used to compute certificates of unsatisfiability. Unfortunately, in general a formula may be a contradiction while the relaxation of its transform has a maximal value above th_{true}.

Let us specify an approach to constructing an appropriate function $\mathcal{T}(x)$. The idea is to first introduce a mapping $F_k(x) : [-1,1]^m \to [0,1]$ that maps, for any $\{-1,1\}$ vector x, a clause $\mathbf{C}_k$ to 1 if it is satisfied by x, and to 0 if it is not. Subsequently, a mapping $G : [0,1]^n \to [0,1]$ maps the conjunction of clauses to 1 if and only if each clause is *true*, i.e. x is a satisfying assignment. Thus the transformation of a propositional formula Φ is given by

$$\mathcal{T}(x) = G\left(F_1(x)), \ldots, F_n(x)\right).$$

Note that when the F_k are strictly monotone and the F_k and G are either both convex or both concave (this is posed as a condition in [98]), the superposition of these functions is also convex or concave. In order to get a better understanding of these transformation, let us consider some specific examples.

Example 2.3.1 We give three examples.

1. Let the functions $F_\ell : [-\ell,\ell] \to [0,1]$ be defined as

$$F_\ell(x) = 1 + (-1)^{\ell+1}2^{-\ell}\frac{1}{\ell!}\prod_{i=1}^{\ell}(x + \ell - 2i),$$

(hence $F_\ell(x) = 0$ if and only if $x = -\ell$; otherwise $F_\ell(x) = 1$). Transform a clause $\mathbf{C}_k$ of length ℓ using the linear function $a_k^T x$ and substitute this in F_ℓ. Then it holds for $\{-1,1\}$ vectors x that $F(a_k^T x) = 1$ if and only if $a_k^T x \geq 2 - \ell(\mathbf{C}_k)$. For the aggregation, use the function G:

$$G(x_1,\ldots,x_n; w_1,\ldots,w_n) = \sum_{k=1}^{n} w_k x_k,$$

where the w_k's are strictly positive weights. We assume (without loss of generality) that the weights sum to one. Thus $\mathcal{T}(x) = 1$ if and only if x is a satisfying assignment.

2. Another example is the polynomial or *multilinear* representation of SAT problems as used by Gu [61]. A clause $\mathbf{C}_k$ is represented as a polynomial function:

$$P_k(x) = \prod_{i \in I_k} (1 - x_i) \prod_{i \in J_k} (1 + x_i) = \prod_{i=1}^{m} (1 - a_{ki} x_i), \tag{2.4}$$

where it is required that $P_k(x) = 0$ for x to be a satisfying assignment of clause $\mathbf{C}_k$. Use again the function G with associated weights of the previous example to obtain the *weighted polynomial representation* of SAT :

$$\text{(WPR)} \quad \text{find } x \in \{-1,1\}^m \text{ such that } \mathcal{P}(x;w) = \sum_{k=1}^{n} w_k(1 - P_k(x)) = 1.$$

3. Van Maaren [96, 97] uses the following mappings. For $\varepsilon > 0$, the function $\mathcal{A}_\varepsilon : (-\infty, 1] \to [0,1]$ is given by

$$\mathcal{A}_\varepsilon(x) = \frac{x + \sqrt{x^2 + \varepsilon}}{1 + \sqrt{1 + \varepsilon}},$$

and for $r \leq 1$, $r \neq 0$, $\mathcal{A}_r : [0,1]^n \to [0,1]$ is defined by

$$\mathcal{A}_r(x_1, \ldots, x_n) = \left(\frac{1}{n} \sum_{i=1}^{n} x_i^r \right)^{\frac{1}{r}}.$$

Note that $\mathcal{A}_\varepsilon$ is convex and monotone on the given interval, while $\mathcal{A}_r$ is concave. The (concave) transform of a clause $\mathbf{C}_k$ is given by

$$F_k(x) = 1 - \mathcal{A}_\varepsilon(1 - \frac{1}{2}(\ell(\mathbf{C}_k) + a_k^T x).$$

It holds that $F_k(x) = 0$ if and only if x is a contradictory assignment; otherwise $F_k(x) \approx 1$ (assuming ε is small). The transform of a formula Φ is obtained using $\mathcal{T}(x) = \mathcal{A}_r(F_1(x), \ldots, F_n(x))$. The threshold value as a function of ε and r can be obtained by noting that $F_k(x) \geq 1 - \mathcal{A}_\varepsilon(0)$ for any satisfying assignment and substituting this in $\mathcal{A}_r$. □

Although the transformations introduced above provide valid representations of SAT problems, it is not straighforward to exploit them to obtain computationally effective methods for solving SAT in general. Algorithms relying on nonlinear models have been developed by Gu [61] and Shang and Wah [118]; in Chapter 6 we discuss several similar algorithms. The results obtained using these algorithms are often remarkably good. However, due to the fact that the algorithms rely on optimizing *nonconvex* functions, it seems that they are useful to solve satisfiable instances only.

We are interested in making use of nonlinear *convex* structures to facilitate proving both satisfiability and unsatisfiability. Functions of the type $\mathcal{T}(x)$ provide relatively concise (single) expressions that exactly represent the formulas at hand. Thus a sensible approach seems to construct concise smooth *approximations* of $\mathcal{T}(x)$ which are convex and approximate $\mathcal{T}(x)$ with some accuracy. To this end, van Maaren [96, 97, 98] proposes to exploit first and second order Taylor approximations. Obviously, the approximation does not necessarily induce a strict separation of satisfying and contradictory assignments, but hopefully it still captures useful characteristics of the formula. For example, approximating the transform induced by $\mathcal{A}_\varepsilon$ and $\mathcal{A}_r$ in the above example, van Maaren obtains a (convex) ellipsoid, which under certain parameter settings contains all satisfiable assignments and does not contain the average unsatisfiable assignment. In this thesis we make heavy use of a comparable elliptic approximation which was first introduced in [96]. An easy derivation is given in the next section. In Appendix B a derivation via second-order Taylor approximations is given.

2.4 Weighted elliptic approximations

2.4.1 A straightforward derivation

Recall that an assignment $x \in \{-1,1\}^m$ satisfies clause $\mathbf{C}_k$ if and only if $a_k^T x \geq 2 - \ell(\mathbf{C}_k)$. It follows that if x is a satisfying assignment, then

$$1 - \ell(\mathbf{C}_k) \leq a_k^T x - 1 \leq \ell(\mathbf{C}_k) - 1.$$

Squaring this expression, we obtain an elliptic representation of a single clause. An assignment x satisfies clause $\mathbf{C}_k$ if and only if $x \in \mathcal{E}_k$, where $\mathcal{E}_k$ is defined as

$$\mathcal{E}_k = \{x \in \mathbb{R}^m \mid (a_k^T x - 1)^2 \leq (\ell(\mathbf{C}_k) - 1)^2\}. \tag{2.5}$$

Consequently, the satisfiability problem can be expressed as finding a $\{-1,1\}$ vector x lying in the intersection of n ellipsoids, i.e.

$$x \in \bigcap_{k=1}^{n} \mathcal{E}_k \cap \{-1,1\}^m.$$

However, it is hard to characterize the intersection of two or more ellipsoids explicitly. We therefore need to find another way to aggregate the information contained in the n separate ellipsoids. We choose to take the sum over all these ellipsoids, which again yields an ellipsoid. Unfortunately, during summation the discriminative properties of the separate ellipsoids are partly lost. We speak therefore of an *approximation* of a propositional formula. Rather than weighting each clause equally in the summation, let us associate a nonnegative weight w_k with each individual clause and associated ellipsoid. Then for any satisfying assignment $x \in \{-1,1\}^m$ it must hold that

$$\sum_{k=1}^{n} w_k (a_k^T x - 1)^2 \leq \sum_{k=1}^{n} w_k (\ell(\mathbf{C}_k) - 1)^2. \tag{2.6}$$

Let $w \in \mathbb{R}^n$, $w \geq 0$, be a weight vector. Denoting $W = \text{diag}(w)$, we define the quadratic function $\mathcal{Q}(x; w)$ as

$$\mathcal{Q}(x; w) = x^T A^T W A x - 2 w^T A x. \tag{2.7}$$

Furthermore, let the vector $r \in \mathbb{R}^n$ be such that $r_k = \ell(\mathbf{C}_k)(\ell(\mathbf{C}_k) - 2)$, or equivalently $r_k = b_k(b_k - 2)$. We have the following theorem.

Theorem 2.4.1 *Let Φ be a CNF formula with associated clause-variable matrix A and let $w \in \mathbb{R}^n$ be a vector of nonnegative clause weights. The elliptic region*

$$\mathcal{E}(w) = \{x \in \mathbb{R}^m \mid \mathcal{Q}(x; w) \leq r^T w\}$$

contains all satisfying assignments of Φ.

Proof: Rewriting (2.6), we obtain

$$\sum_{k=1}^{n} w_k (a_k^T x)^2 - 2 \sum_{k=1}^{n} w_k a_k^T x \leq \sum_{k=1}^{n} w_k \ell(\mathbf{C}_k)(\ell(\mathbf{C}_k) - 2).$$

This is equivalent to

$$x^T A^T W A x - 2 w^T A x \leq r^T w.$$

The theorem follows immediately. □

The region $\mathcal{E}(w)$ is called an *elliptic approximation.* By construction it is convex. A corollary of Theorem 2.4.1 is the following.

Corollary 2.4.2 *A necessary and sufficient condition for $x \in \{-1, 1\}^m$ to be a satisfying assignment of Φ is that $x \in \mathcal{E}(w)$ for all $w \geq 0$, $w \neq 0$.*

To verify this corollary, note that the necessity of the condition follows from Theorem 2.4.1. Its sufficiency follows by observing that if x is a *contradictory* assignment, then it violates some clause $\mathbf{C}_k$. As a consequence, taking $w_k = 1$ and setting all other weights to zero, $x \notin \mathcal{E}(w)$.

In general, for any given $w \geq 0$, apart from the satisfying assignments some (or even all) contradictory assignments might be contained in the ellipsoid as well. On the other hand, Corollary 2.4.2 indicates that by adjusting the clause-weights, one can attempt to improve the tightness of the elliptic approximation, i.e. to reduce the number of (unwanted) contradictory assignments contained in it. In Section 2.4.3 we illustrate this via an example. Let us first show that for 2CNF formulas the ellipsoid is tight when $w > 0$. The elliptic approximation of a pure ℓCNF formula is denoted as $\mathcal{E}_\ell(w)$.

Lemma 2.4.3 *Let Φ be a (pure) 2SAT formula with elliptic approximation $\mathcal{E}_2(w)$. For any $\{-1, 1\}$ assignment x, it holds that $x \in \mathcal{E}_2(w)$ if and only if x is a satisfying assignment.*

Proof: By Theorem 2.4.1 it is clear that any satisfying assignment $x \in \mathcal{E}_2(w)$. For a 2CNF formula, $r \equiv 0$, implying that the elliptic approximation reduces to $(Ax - 2e)^T W A x \leq 0$. Note that for any satisfying assignment x in fact equality holds, since $a_k^T x \in \{0, 2\}$, $1 \leq k \leq n$. If $x \in \{-1, 1\}^m$ violates a clause $\mathbf{C}_k$, then $(Ax - 2e)^T W A x \geq 8 w_k > 0$ (provided that $w > 0$). □

As stated before, an alternative derivation of weighted elliptic approximations is given in Appendix B.

2.4.2 Entities involved in the elliptic approximation

A CNF formula in conjunction with its vector of clause-weights has a number of problem-specific entities associated with it that are accessible via the elliptic approximation. For easy reference we state these as lemmas. We restrict ourselves to the case where $w = e$, i.e. all clauses are assigned equal weights.

Lemma 2.4.4 *For any formula Φ with associated clause-variable matrix A,*

$$\begin{aligned}(A^TA)_{ii} &= \sigma(p_i) + \sigma(\neg p_i);\\ (A^Te)_i &= \sigma(p_i) - \sigma(\neg p_i).\end{aligned}$$

Proof: Since $a_{ki} = \pm 1$ if and only if $i \in I_k \cup J_k$, it holds that $(A^TA)_{ii} = \sum_{k=1}^{n} a_{ki}^2$. The first equality follows (see also (2.2-2.3)). The second equality is proved similarly. □

In words, diagonal element i is equal to the total number of (weighted) occurrences (both negated an unnegated) of variable p_i. Similarly, element i of the linear term A^Tw is equal to the balance of positive and negative weighted occurrences of variable p_i. Also, the *total* number of variable occurrences can be computed through

$$\sum_{i=1}^{m} (\sigma(p_i) + \sigma(\neg p_i))$$

which is equal to

$$\sum_{k=1}^{n} \ell(\mathbf{C}_k).$$

Combining these with Lemma 2.4.4, we obtain the following equalities.

Lemma 2.4.5 *For any formula Φ and associated clause-variable matrix A,*

$$\sum_{i=1}^{m} (\sigma(p_i) + \sigma(\neg p_i)) = \sum_{i=1}^{m} (A^TA)_{ii} = \sum_{k=1}^{n} \ell(\mathbf{C}_k).$$

2.4.3 An example

To illustrate the notions discussed above and the role of the clause weights, we include a small example of a CNF formula and its elliptic approximation.

Example 2.4.6 The formula Φ consists of the conjunction of 6 clauses.

$$\begin{array}{lclcccccccc}
\mathbf{C}_1 &=& (p_1 & & & & & \vee & \neg p_4) & & \\
\mathbf{C}_2 &=& (\neg p_1 & & & & & \vee & \neg p_4 & \vee & p_5) \\
\mathbf{C}_3 &=& & & & & (p_3 & \vee & p_4) & & \\
\mathbf{C}_4 &=& & & (\neg p_2 & \vee & \neg p_3 & \vee & p_4) & & \\
\mathbf{C}_5 &=& & & (p_2 & \vee & \neg p_3 & & & \vee & \neg p_5) \\
\mathbf{C}_6 &=& & & & & (p_3 & \vee & \neg p_4 & \vee & p_5).
\end{array}$$

Each clause $\mathbf{C}_k$ has a nonnegative weight w_k associated with it. The corresponding elliptic approximation can be constructed using the matrix A^TWA which is given by

$$\begin{bmatrix} w_1+w_2 & 0 & 0 & -w_1+w_2 & -w_2 \\ & w_4+w_5 & w_4-w_5 & -w_4 & -w_5 \\ & & w_3+w_4+w_5+w_6 & w_3-w_4-w_6 & w_5+w_6 \\ & & & w_1+w_2+w_3+w_4+w_6 & -w_2-w_6 \\ & & & & w_2+w_5+w_6 \end{bmatrix},$$

the linear term

$$A^Tw = \begin{bmatrix} w_1-w_2 \\ -w_4+w_5 \\ w_3-w_4-w_5+w_6 \\ -w_1-w_2+w_3+w_4-w_6 \\ w_2-w_5+w_6 \end{bmatrix},$$

and the (also weight-dependent) right hand side:

$$r^Tw = 3(w_2+w_4+w_5+w_6).$$

Let us first consider the unweighted case ($w = e$). In Figure 2.1 the 32 distinct assignments of Φ are evaluated, ordered according to their binary value (i.e. the all *false* assignment has index 1, the assignment $x = (1,-1,-1,-1,-1)$ has index 2 and so on; the all *true* assignment has the highest index $2^5 = 32$). The bars indicate the value of $\mathcal{Q}(x; w \equiv e)$. Dark bars correspond to satisfying assignments (there are five of these) and white bars to contradictory assignments. Note that the approximation is 'tight' in the sense that a satisfying assignment lies on its boundary. Of the 27 *false* assignments, 9 are contained in the elliptic approximation. For instance, the assignment indexed 18 ($x = (1,-1,-1,-1,1)$) has elliptic value $\mathcal{Q}(x;e) = 8 < 12$ but it violates the third clause. As an example of a satisfying assignment, it can be easily verified that the solution with index 5, $x = (-1,-1,1,-1,-1)$ with $\mathcal{Q}(x;e) = 4$, indeed satisfies all clauses.

To illustrate that by adjusting the weights the tightness of the ellipsoid can improve, let us first distinguish between clauses of different lengths only. Consider $w = [2,1,2,1,1,1]$. Now the number of contradictory assignments contained in the ellipsoid reduces to just one which lies on the boundary of the ellipsoid; see Figure 2.2. The corresponding assignment is $x = (-1,1,1,-1,-1)$. The reader may verify that to push this solution out of the ellipsoid as well, a necessary condition is that $-w_2+3w_4-w_5-w_6 > 0$; hence giving all clauses of length 3 equal weight can never work. However, taking $w = [2,1,2,1,1,0]$ (note that we actually neglect the last clause), lo and behold a strict separation of satisfying and contradictory assignments is obtained! This is illustrated in Figure 2.3. Consequently, the last clause is redundant; it is implied by $\mathbf{C}_1 \wedge \mathbf{C}_2$. Let us emphasize that in general it is not necessarily possible to obtain a strict separation of satisfying and contradictory assignments. $\square$

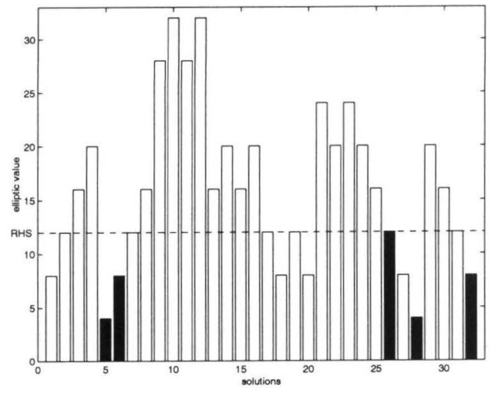


Figure 2.1: Partial separation of assignments via unweighted elliptic approximation.

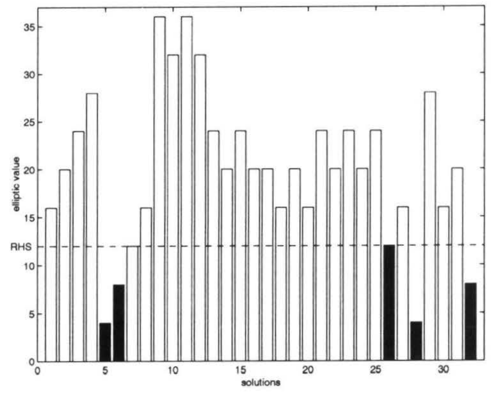


Figure 2.2: Partial separation of assignments via weighted elliptic approximation.

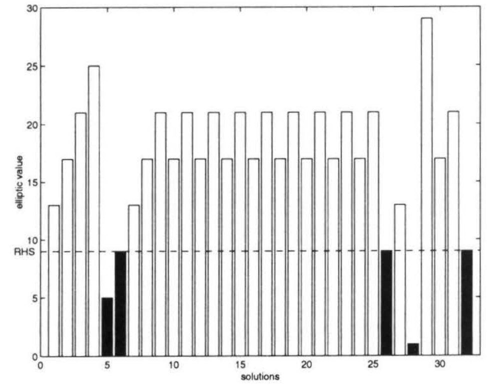


Figure 2.3: Strict separation of assignments via weighted elliptic approximation.

2.4.4 Further adjusting the elliptic approximation

As the example of the previous section highlights, the tightness of the ellipsoid can improve vastly when the weights are adjusted. This suggests that by optimizing over the weights (provided a meaningful optimization problem is defined), interesting and useful characteristics of the formula at hand can be discovered and exploited. Another observation that we can use to further adjust the approximation is the following. For any vector $u \in \mathbb{R}^m$ and any binary vector $x \in \{-1, 1\}^m$, it holds that

$$x^T(A^TWA - \operatorname{diag}(u))x = x^TA^TWAx - e^Tu,$$

since $x_i^2 = 1$ for $x_i \in \{-1, 1\}$. The introduction of the vector u has no effect on the separating quality of the elliptic approximation, but it is of great importance in deriving bounds, models and algorithms for binary quadratic optimization problems in general. Observe that u can be chosen such that the matrix $A^TWA - \operatorname{diag}(u)$ is positive definite, negative definite or indefinite (for the reader who is uncomfortable with these terms some basic notions from linear algebra are reviewed in Section 3.2.2). Accordingly, the quadratic forms associated with it are of a different nature, although they are equivalent when restricted to $\{-1, 1\}$ variables. In this thesis, various choices of u are considered, which yield models exhibiting different characteristics for different purposes.

- In Chapter 3 the model with $u \equiv 0$ is considered. It is used to obtain branching rules. In addition, the problem of obtaining 'good' weights is addressed (in a simplified form).
- Chapter 4 also uses the model with $u \equiv 0$ to characterize and to recognize a specific class of polynomially solvable formulas. By optimizing over the weights, subformulas with the desired structure can be identified.
- By taking both u and w to be variable, *eigenvalue optimization* problems arise. These can be cast as *semidefinite programming problems*, which are related to *semidefinite relaxations* of combinatorial optimization problems. This is the subject of Chapter 5.
- In Chapter 6 a model is derived which is equivalent to the model obtained after substituting $u = \operatorname{diag}(A^TWA)$. The model thus obtained is nicely suited for the design of fast approximation algorithms for a special class of combinatorial optimization problems, which includes the Frequency Assignment Problem (see Section 1.3.3).

3

Enhancing The DPLL Algorithm Using Elliptic Approximations

The most widely used complete search algorithm for SAT solving is the DPLL algorithm. The quality of the branching rule involved is crucial for its performance. Based on elliptic approximations of SAT problems, we devise new branching rules and relate them to the existing ones. To obtain weights for clauses of different lengths, we study the volume of the ellipsoids. Computational evidence to support the claim that the newly devised branching rule outperforms the existing ones in terms of node counts is provided.

3.1 Introduction

One of the best known and most widely used algorithms for solving satisfiability (SAT) problems, is the Davis-Putnam-Logemann-Loveland algorithm [36, 37]. This algorithm implicitly enumerates all solutions to the SAT problem at hand by setting up a binary search tree. At each node of the search tree, a variable must be chosen to branch on (i.e. to fix at $true$ and $false$ in the two subtrees) by some *branching rule*. The actual performance of the DPLL approach (for medium to large sized problems) depends on the effectiveness of the branching rule. For example, Dubois et al. [44] compare the performance of random branching and guided branching. The search trees of the first are larger than that of the latter by orders of magnitude. As a consequence, to accommodate the solution of 'hard' SAT problems (see e.g. [31, 101]), branching rules need to be identified that are computationally cheap and keep the search trees manageable. A number of widely used branching rules can be found in studies by Jeroslow and Wang [77], Hooker and Vinay [74], Freeman [49], Crawford and Auton [31], Dubois et al. [44] and Van Gelder and Tsuji [53].

To provide an intuitive motivation for a branching rule, often a hypothesis is posed that specifies which properties a branching variable must have, such that it is likely to yield a small search tree. Jeroslow and Wang [77] introduce the *satisfaction* hypothesis, which asserts that a branching variable must be chosen such that the resulting subproblem is likely to be satisfiable. Later, Hooker and Vinay [74] argue that the *simplification* hypothesis is more appropriate: rather than aiming for satisfiable subproblems, one should aim for relatively easy subproblems (either satisfiable or unsatisfiable). Essentially, we also take the simplification hypothesis as motivation. However, the way in which the hypothesis is interpreted to obtain branching rules differs completely from Hooker and Vinay's approach. Instead of (directly) counting variable occurrences, we use an elliptic approximation to capture the important characteristics of the CNF formula under consideration in a single expression (from which variable counts can be extracted as well; see Section 2.4.2). The ellipsoid involved is a smooth approximation of the SAT problem under consideration. Using the particular structure of the ellipsoid branching variables with 'a great impact' on the formula can be identified. Below we sketch how we aim to measure this impact via the ellipsoid. If a variable has great impact on the formula, the resulting subformulas seem likely to be relatively easy. This is the intuition behind the branching rules we devise. The design of a branching rule involves two major issues:

- Assigning weights to clauses;
- Specifying the actual branching rule.

Let us discuss these issues in some detail. The problem of determining the relative importance of individual clauses can play a key-role for the actual performance of a branching rule. Many algorithms might benefit greatly from identifying 'important' clauses at an early stage during execution. Usually, the relative importance of clauses is measured solely by their lengths, and different weights are assigned to clauses of different lengths, where short clauses are weighted higher than long ones. The natural question is then how to choose these weights. As far as we know, mostly the weights are determined based on some rather intuitive probabilistic argument first stated by Jeroslow and Wang [77], or in an *ad hoc* experimental way (Dubois et al. [44], Crawford [30]). We propose to find weights, such that the elliptic approximation is in some sense 'tight': as a measure of tightness, we use the *volume* of the ellipsoid. We experimentally determine weights which yield ellipsoids with small volumes. The weights deduced using this approach are confirmed to yield better results in the DPLL branching algorithm than the weights that are usually used in the literature, especially when a branching strategy is used that is based on the elliptic approximations as well.

The other step is to devise the actual branching rule. Our approach is to find a branching variable such that after fixing this variable the elliptic approximation in lower dimension can be expected to be (relatively) smaller, and therefore tighter. Interestingly, using this approach, we obtain some new branching rules, but also rediscover well known branching rules such as proposed in [44, 53, 74]. These are now obtained with a geometrical motivation rather than a counting argument. In fact, these rules can be considered as approximations of the branching rules making full use of the elliptic structure. The latter outperform the old ones in terms of node counts (on hard random 3SAT instances).

This chapter is organized as follows. In the following section we discuss the preliminaries and review some basic notions from linear algebra. In Section 3.3 the DPLL algorithm is reviewed and we give a general framework to classify branching rules. Also some enhancements of the DPLL algorithm are mentioned. Section 3.4 is concerned with the derivation of various branching rules using the elliptic approximation. In Section 3.5 the problem of finding adequate weights is addressed. We report on computational results on hard random 3SAT problems in Section 3.6, and conclude with some remarks.

3.2 Preliminaries and notation

In this section we briefly review the satisfiability problem and its elliptic approximation, and some notions from linear algebra which we need for deriving the new branching rules.

3.2.1 SAT and elliptic approximations

We use the notation introduced in the Chapters 1 and 2. We review the notation most relevant for this chapter. A CNF formula Φ is the conjunction of n clauses $\mathbf{C}_k$, where each clause is a disjunction of a number of literals l_i. A literal is a proposition (or variable) p_i or its negation $\neg p_i$. By $\ell(\mathbf{C}_k)$ the length of clause $\mathbf{C}_k$ is denoted. Let m be the number of distinct variables occurring in Φ. By $A \in \mathbb{R}^{n\times m}$ we denote its associated *clause-variable* matrix (see (2.1)). The associated right hand vector is denoted by $b \in \mathbb{R}^n$, where $b_k = 2-\ell(\mathbf{C}_k)$. To each clause a nonnegative weight $w_k \geq 0$ is assigned. Let $w = [w_1, \ldots, w_n]$ and $W = \mathrm{diag}(w)$. A *(weighted) elliptic approximation* of Φ (see Theorem 2.4.1) is given by

$$\mathcal{E}(w) = \{x \in \mathbb{R}^m \mid x^TA^TWAx - 2w^TAx \leq b^TW(b-2e)\}.$$

This ellipsoid contains all satisfying assignments of Φ. In this chapter clauses with the same length are assigned equal weights. Considering the pure ℓCNF problem, the expression for the ellipsoid reduces to (noting that $b = (2-\ell)e$)

$$\mathcal{E}_\ell = \{x \in \mathbb{R}^m \mid x^TA^TAx - 2e^TAx \leq \ell(\ell-2)n\}.$$

By $\sigma(l_i)$ we denote the number of occurrences of literal l_i in a formula Φ. It holds that (see Lemma 2.4.4)

$$(A^TA)_{ii} = \sigma(p_i) + \sigma(\neg p_i); \tag{3.1}$$

$$(A^Te)_i = \sigma(p_i) - \sigma(\neg p_i). \tag{3.2}$$

3.2.2 Some basic linear algebra

For easy reference we review some basic linear algebra. For a rigorous overview on the subject, see e.g. Strang [123]. Consider the quadratic function

$$\mathcal{Q}(x) = x^TQx - 2q^Tx.$$

Assume that $Q \in \mathbb{R}^{m\times m}$ is a symmetric, positive definite matrix, i.e. $x^TQx > 0$ for any vector $x \in \mathbb{R}^m$. Then $\mathcal{Q}(x)$ is a convex function, which attains its unique minimum (with value equal to $-q^TQ^{-1}q \leq 0$) in $\tilde{x} = Q^{-1}q$. The matrix Q has a *spectral decomposition* $Q = S\Lambda S^T$. Here the matrix Λ is a diagonal matrix with on its diagonal the m strictly positive *eigenvalues* of Q, and the matrix S contains a set of orthonormal *eigenvectors* corresponding to the eigenvalues; it holds that $Qs_i = \lambda_i s_i$, $1 \leq i \leq m$. Now consider the region

$$\mathcal{E}(r) = \{x \in \mathbb{R}^m \mid \mathcal{Q}(x) \leq r^2\}.$$

This region is a bounded nonempty elliptic region, provided that $r^2 \geq -q^TQ^{-1}q$. This can be verified as follows. Letting $y = x - \tilde{x}$, we obtain

$$\mathcal{E}(\tilde{r}) = \{y \in \mathbb{R}^m \mid y^TQy \leq \tilde{r}^2\}.$$

It may be verified that $\tilde{r}^2 = r^2 + q^T Q^{-1} q$. We call r the *radius* of the ellipsoid. Using the change of variables $z = S^T y$ we find that

$$\mathcal{E}(\tilde{r}) = \{z \in \mathbb{R}^m \mid z^T \Lambda z = \sum_{i=1}^{m} \lambda_i z_i^2 \leq \tilde{r}^2\}.$$

Since $\lambda_i > 0$, $1 \leq i \leq m$, it is easy to see that this is a bounded elliptic region. In z-space, the ellipsoid is centered at the origin and the axes point along the unit vectors. The lengths of the axes are equal to $\tilde{r}/\sqrt{\lambda_i}$. It follows that in the original x-space the ellipsoid is centered at $\tilde{x}$, while its axes point along the eigenvectors. Note that the longest axis of the ellipsoid is the one corresponding to the eigenvector corresponding to the smallest eigenvalue. If the matrix Q has an eigenvalue zero (then it is only positive *semi*definite) the region is not bounded.

3.3 The DPLL algorithm and enhancements

We first describe the basic ingredients of the generic branching algorithm that is most widely used to solve instances of satisfiability: the DPLL algorithm. Subsequently, we give a framework in which most known branching rules can be systematically classified. This section ends with the description of some enhancements of the DPLL procedure.

3.3.1 The original DPLL algorithm

We describe the variant of the Davis-Putnam algorithm [37] introduced by Davis, Logemann and Loveland [36] which is known as the DPLL-algorithm. The DPLL-algorithm is an implicit enumeration algorithm, that enumerates solutions by setting up a binary search tree. In figure 3.1 the DPLL-algorithm is summarized.

```
procedure DPLL ($\Phi$, depth);
  $\Phi$:=unit_resolution($\Phi$);
  $\Phi$:=mon_var_fix($\Phi$);
  if $\Phi = \emptyset$ then
    $\Phi$ is satisfiable: return(satisfiable);
  if $\mathbf{C}_k = \emptyset$ for some $\mathbf{C}_k \in \Phi$ then
    $\Phi$ is contradictory: backtrack;
  $l$:=branch_rule($\Phi$);
  DPLL($\Phi \cup \{l\}$, depth+1);
  DPLL($\Phi \cup \{\neg l\}$, depth+1);
return(unsatisfiable);
```

Figure 3.1: The DPLL algorithm.

We discuss the DPLL algorithm in some more detail.

- The procedure `unit_resolution` finds all unit clauses (i.e. clauses consisting of a single literal) and sets the corresponding literal to *true*. All clauses in which this literal appears are then satisfied, while all occurrences of the negation of this literal are eliminated. Note that performing unit resolution, new unit clauses can emerge; these are then added to the list of unit clauses and processed in turn. The procedure terminates when no unit clauses remain, or when the empty clause is derived. It can be implemented to run in linear time (Gallier and Dowling [41]).

- The procedure `mon_var_fix` performs *monotone variable fixing*. If some variable occurs negated only (or unnegated only), it can be set to $false$ ($true$) such that all clauses it appears in are satisfied. The remaining formula is obviously satisfiability-equivalent to the original one. In fact, a monotone variable gives rise to an autark assignment (see Definition 2.2.3). Note that this procedure is often not included in current state-of-the-art implementations.

- When the problem is not solved after exhaustively applying the previous steps, a literal is chosen to branch on. It is added as a unit clause to the formula in the left branch of the tree and subsequently the DPLL procedure is recursively called. If no solutions are found in the left subtree, on returning the negation of the literal is added as a unit clause to the formula in the right branch of the tree, and again the DPLL procedure is recursively called. Obviously, in any solution the literal is either *true* or $false$; consequently the DPLL method is complete.

As stated before, a very important step of the DPLL algorithm is the choice of the branching rule. If good branching variables are chosen, the search tree is kept relatively small. Several studies are concerned with (among other things) finding good branching rules, e.g. [44, 53, 74, 77]. In the next section we will discuss some well known branching rules, and in the sections thereafter a number of new branching rules, based on the elliptic approximation, are derived.

3.3.2 Uniform structure of branching rules

In this section we devise a general framework to systematically classify the branching rules that are discussed in the literature [31, 44, 49, 53, 74]. Most branching rules can be interpreted to consist of the following steps.

1. **Restrict**. Determine a set $\mathcal{P}^* \subseteq \{p_1, \dots, p_m\}$ of candidate branching variables.

2. **Compute**. For each variable $p_i \in \mathcal{P}^*$ compute $f(p_i)$ and $f(\neg p_i)$, where the function f measures the 'quality of branching on $(\neg)p_i$'.

3. **Balance**. For each variable $p_i \in \mathcal{P}^*$ balance the values obtained in the previous step using some balancing function g, i.e. $bal_i = g(f(p_i), f(\neg p_i))$.

4. **Choose**. Take the index i that maximizes bal_i. If there is a tie, some rule needs to be specified to break it. In the left subtree, assume that p_i is *true* if $f(p_i) \geq f(\neg p_i)$, otherwise assume $\neg p_i$ (i.e. add the unit clause p_i or $\neg p_i$ respectively).

Let us consider these four steps in more detail. Branching rules can be motivated by the *simplification hypothesis* (Hooker and Vinay [74] argue that this is more appropriate than Jeroslow and Wang's *satisfaction hypothesis* [77]; in Section 3.4 we show that branching rules can also be motivated by geometrical reasoning). This means that the branching rules are aimed towards simplifying the subproblems occurring in the subtrees as much as possible.

As a first step, it may be worthwhile to restrict the set of candidate branching variables in order to try and aim for deriving a trivially satisfiable formula. For example, if in some formula each clause contains at least one unnegated variable it is trivially satisfiable (by setting all variables to $false$). To aim for such a formula, only the variables that occur in all-positive clauses could be considered as candidates.

The simplification of a formula is often measured by a counting argument. Basically, the idea is that if a variable occurs often, the subproblems obtained after branching on that variable should be relatively small. The usual choice for f is

$$f(l_i) = \sum_{l_i \in \mathbf{C}_k} w(\mathbf{C}_k),$$

for some weight vector w. Usually $w(\mathbf{C}_k) = w(\ell(\mathbf{C}_k))$; i.e. the weights depend on lengths of clauses only. Then f reduces to (denoting the number of occurrences of literal l_i in clauses of length ℓ by $\sigma_\ell(l_i)$)

$$f_{ws}(l_i) = \sum_{\ell=2}^{\ell_{\max}} w_\ell \sigma_\ell(l_i),$$

a *weighted sum* of occurrences. By $\ell_{\max}$ the maximum clause length is denoted. Note that the actual value of w_ℓ is not important; rather one should consider the ratio $w_\ell / w_{\ell+1}$.

The third step is made to try to 'balance' the search tree. If $f(p_i)$ is large branching to p_i in the left subtree is a good choice, according to the measure implied by f. However, if during backtracking the current node is visited again, this implies that in the right subtree one branches to $\neg p_i$, which might a bad choice. Thus one likes to ensure that according to the measure used both branches are reasonable. This indicates that balancing is especially important when solving unsatisfiable formulas (since then each interior node of the search tree is visited twice).

For the balancing function g the following choices occur most frequently in the literature.

1. $g_+(f(p), f(\neg p)) = f(p) + f(\neg p)$.
2. $g_*(f(p), f(\neg p)) = f(p) \cdot f(\neg p)$.
3. $g_\alpha(f(p), f(\neg p)) = \max\{f(p), f(\neg p)\} + \alpha \min\{f(p), f(\neg p)\}$.

Observe that it is reasonable to require for the balancing function that $g(f(p_i), f(\neg p_i)) \geq g(f(p_j), f(\neg p_j))$ if $f(p_i) \geq f(p_j)$ and $f(\neg p_i) \geq f(\neg p_j)$. All functions above satisfy this condition (provided $\alpha \geq 0$). Note that g_+ corresponds to taking the *arithmetical* mean of two numbers, while g_* computes the *geometrical* mean.

Example 3.3.1 We consider branching rules from the literature and put them in the above framework. In all cases f_{ws} is used in step 2; the weights vary and are specified. If not explicitly stated otherwise $\mathcal{P}^* = \{p_1, \ldots, p_m\}$.

- JEROSLOW-WANG [77]. Take $w_\ell = 2^{-\ell}$ and use balancing function g_α, $\alpha = 0$.
- FIRST ORDER PROBABILITY RULE [74]. Take $w_\ell = 2^{-\ell}$ and use balancing function g_α with $\alpha = -1$.
- 2-SIDED JEROSLOW-WANG [74]. Take again $w_\ell = 2^{-\ell}$, but now use balancing function g_+. Variations of this rule are known as max-occurrence, or maxscore [53] (then $w_\ell = 1$ for all ℓ).
- POSITIVE 2-SIDED JEROSLOW-WANG [74]. Same as the previous, but now $\mathcal{P}^*$ contains variables occurring in all-positive clauses only.
- MINLEN [53]. Take $w_2 = 1$, $w_\ell = 0$ for $\ell \geq 3$, and use balancing function g_*.
- SATO [141]. Similar to MINLEN, except $\mathcal{P}^*$ is restricted to a subset of variables occurring in the shortest non-Horn (i.e. clauses with more than one positive literal) clauses and $f(p) := f(p) + 1$, $f(\neg p) := f(\neg p) + 1$.
- POSIT [49]. Similar to MINLEN, except as balancing function $N \cdot g_* + g_+$ is used, N large (Freeman takes $N = 1024$).
- DSJ [53]. Take $w_\ell = 2^{-\ell}$ for $\ell = 2, 3, 4$, $w_\ell = w_{\ell-1}$ for $\ell \geq 5$, and use balancing function g_*.
- B_2 [44]. Take $w_\ell = -\log(1 - 1/(2^\ell - 1)^2)$, and use balancing function g_α with $\alpha = 2.5$. It may be noted that the rule B_{C-SAT} that is actually used by Dubois et al. uses a more sophisticated function f in step 2, namely

$$f(l_i) = f_{ws}(l_i) + \sum_{l_j \in l_i \vee l_j} f_{ws}(\neg l_j).$$

- BÖHM [12]. Take $w_2 = 1$, $w_\ell = 0$ for $\ell \geq 3$. Use balancing function g_α with $\alpha = 2$. In case of a tie, recompute f for the tied variables but now with $w_3 = 1$, $w_\ell = 0$ for $\ell \neq 3$. □

Later on we devise branching rules and provide geometrical intuitions based on the elliptic approximation. First we mention two enhancements of the DPLL algorithm that have been shown to be quite effective on many classes of problems.

3.3.3 Enhancements of the DPLL algorithm

Dubois et al. [44] introduce an enhancement of the DPLL-algorithm based on *(single) lookahead unit resolution.* Instead of directly choosing a branching variable, first some *probing* is done for additional unit resolutions or pruning of the search tree. A subset of the variables is considered (chosen according to some heuristic scheme). Each of these variables is set to *true* and *false* in turn, and subsequently unit resolution is performed. Obviously, if for some variable p the empty clause is derived when setting it to *true* (*false*) then this variable can be set to *false* (*true*) without losing any solutions. Moreover, if for some variable p the empty clause is derived for both truth values, the current branch can be closed and the algorithm backtracks. Note that for contradictory formulas, single lookahead unit resolution reduces the size of the search trees by at least a factor of two. Schlipf et al. [113] prove that a simple version of single lookahead unit resolution is complete for extended Horn formulas. The algorithm is also used (in conjunction with additional enhancements) to obtain the $\mathcal{O}(1.497^m)$ bound on solving 3SAT formulas [112].

A further enhancement of the DPLL algorithm is *intelligent backtracking* and the addition of valid new clauses as implemented by Marques Silva and Sakallah [120] (see also Zhang [141]). This is based on the observation that by carefully analyzing where the assumptions (i.e. branchings) leading to the current contradiction are made, sometimes large parts of the search tree can be pruned by 'back-jumping' over several levels (or 'depths') rather than backtracking one level. At the same time, new clauses are derived which may be added to the CNF formula.

Suppose that p is the current branching variable and that after branching to p and applying unit resolution the empty clause is derived. Then a new clause can be derived using a directed acyclic graph, which is called the *implication graph.* It is constructed as follows.

1. Each source node is labelled with a branching variable.

2. Each internal node is labelled with a unit literal that is derived in a unit resolution phase.

3. The predecessors of a node p_1 are labelled with the unit literals implying p_1. For example, if a clause $p_1 \vee p_2 \vee \neg p_3$ is present and $\neg p_2$ and p_3 have been derived or assumed, then the predecessors of node p_1 are labelled $\neg p_2$ and p_3.

4. The sink node is labelled with $false$.

The negation of the conjunction of the literals labelling the source nodes (one of which is labelled with p) is concluded to be a valid clause and can be added to the formula. Note that, given the current partial assignment (except p), applying unit resolution to the newly derived clause $\neg p$ is derived. Suppose that this branch again leads directly to a contradiction via unit resolution. Again the corresponding implication graph is constructed. As in the previous case, the unit literals labelling the source nodes are the source of the present contradiction (observe that now $\neg p$ labels an *internal* node). Considering the depths at which these are set, it is easy to see that we can 'jump' back to the depth at which the last one of these was assumed. In many cases (for an indication see Table 1.2, section 1.4.2) this can lead to substantial reductions of search times.

We do not incorporate these techniques in our implementation since they obscure the effect of the various branching strategies. (Note that on random 3SAT instances the improvement in search times seems to be negligible.)

3.4 Branching rules using elliptic approximations

In this section branching rules are derived using the elliptic approximation. First we consider the elliptic approximation and show that we may assume that it constitutes a bounded ellipsoid. Then branching rules making full use of the elliptic structure are derived and subsequently these are approximated to accommodate their incorporation in a potentially efficient DPLL implementation.

3.4.1 The elliptic approximation

We use the following notation.

$$\Phi = \bigcup_{\ell=2}^{\ell_{\max}} \Phi_\ell,$$

where each subformula Φ_ℓ is in CNF, and contains clauses of length exactly ℓ only. It is assumed that no unit clauses are present. Each of the subformulas Φ_ℓ has an integer linear programming formulation (see also Section 2.2)

$$(\text{IP}_{\ell SAT}) \quad \text{find } x \in \{-1,1\}^m \text{ such that } A_\ell x \geq (2-\ell)e,$$

where A_ℓ is the clause-variable matrix associated with subformula Φ_ℓ. Now let us denote the elliptic approximation by

$$\mathcal{E}(w) = \{x \in \mathbb{R}^m | \mathcal{Q}(x;w) \leq r^2\}, \tag{3.3}$$

where

$$\mathcal{Q}(x;w) = x^T Q x - 2q^T x,$$

and

$$Q = \sum_{\ell=2}^{\ell_{\max}} w_\ell A_\ell^T A_\ell,\ q = \sum_{\ell=2}^{\ell_{\max}} w_\ell A_\ell^T e,\ r^2 = \sum_{\ell=3}^{\ell_{\max}} w_\ell \ell(\ell-2)|\Phi_\ell|.$$

Here it is assumed that in some way nonnegative weights w_ℓ are specified for clauses with length ℓ. An approach to determine such weights using the elliptic approximations is discussed in Section 3.5; for the time being we leave them unspecified. Note that Q, q and r^2 are dependent on the weights, but this dependence is not explicit in the notation.

From subsection 3.2.2 it is clear that $\mathcal{E}(w)$ is a bounded elliptic region if Q is positive definite. It is immediately clear that Q is positive *semi*definite (since $x^T A_\ell^T A_\ell x = \|A_\ell x\|^2 \geq 0$, for all x). We show that we may assume Q to be positive definite.

Theorem 3.4.1 *Let Φ be a CNF formula with associated elliptic approximation (3.3). Suppose the matrix Q has an eigenvalue zero, with eigenvector s. Then s is a linear autarky.*

Proof: Each of the matrices $A_\ell^T A_\ell$ is positive semidefinite, hence $Qs = 0$ implies $A_\ell^T A_\ell s = 0$ for all $\ell = 2, \ldots, \ell_{\max}$. This implies that $s^T A_\ell^T A_\ell s = \|A_\ell s\|^2 = 0$, for all ℓ, which in turn implies that s is a linear autarky (Definition 2.2.4; since s is an eigenvector, $s \neq 0$). □

Recalling Theorem 2.2.5, we conclude that if the matrix Q is not positive definite, by computing and rounding the eigenvector(s) corresponding to the zero eigenvalue(s) a satisfiable subformula is detected. This can be removed to obtain a smaller satisfiability-equivalent formula, whose associated matrix Q is positive definite. Thus we have established that any formula can be reduced to an equivalent formula with bounded elliptic approximation.

As stated before, the elliptic approximation does not strictly separate the satisfying and unsatisfying assignments. In particular, if Φ is a contradictory formula, the ellipsoid will in general not be empty. Still, we can use the ellipsoid irrespective of the constant term (right hand side) involved. Intuitively, the geometry of the ellipsoid contains useful information on the formula at hand. For example, in [97] a branching rule is proposed that uses the eigenvalues and -vectors of the matrix Q, and it is pointed out that the center of the ellipsoid (which obviously is also independent of the right hand side), provides a good heuristic for finding a satisfying assignment. In the next subsection we derive new branching rules using the elliptic approximation.

3.4.2 Derivation of branching rules

Let us now address the issue of how to exploit (3.3) to obtain good branching rules. Mainly, we are interested in finding effective functions f to evaluate step 2 of the branching rule. No restrictions on the set of branching variables are made, while due to the symmetry of ellipsoids it is usually not necessary to do any further balancing. This implies that $f(p_i) = f(\neg p_i)$; therefore, to decide which branch to explore first, some additional rule has to specified. This is addressed later on.

First note that (using (3.1-3.2))

$$\begin{aligned} Q_{ii} &= \sum_{\ell=2}^{\ell_{\max}} w_\ell (A_\ell^T A_\ell)_{ii} = \sum_{\ell=2}^{\ell_{\max}} w_\ell(\sigma_\ell(p_i) + \sigma_\ell(\neg p_i)), & (3.4) \\ q_i &= \sum_{\ell=2}^{\ell_{\max}} w_\ell (A_\ell^T e)_i = \sum_{\ell=2}^{\ell_{\max}} w_\ell(\sigma_\ell(p_i) - \sigma_\ell(\neg p_i)). & (3.5) \end{aligned}$$

Thus we have that

$$\begin{aligned} \tfrac{1}{2}(Q_{ii} + q_i) &= \sum_{\ell=2}^{\ell_{\max}} w_\ell \sigma_\ell(p_i) = f_{ws}(p_i), \\ \tfrac{1}{2}(Q_{ii} - q_i) &= \sum_{\ell=2}^{\ell_{\max}} w_\ell \sigma_\ell(\neg p_i) = f_{ws}(\neg p_i). \end{aligned}$$

Balancing these values using g_+ and g_* respectively, we conclude that

- the MAXIMAL WEIGHTED OCCURRENCE RULE [44, 53, 74] can be expressed as finding the maximal diagonal element of Q. For a specific choice of weights this is the 2-SIDED JEROSLOW-WANG RULE [74]. We refer to this rule as MWO henceforth.
- the PRODUCT OF WEIGHTED OCCURRENCES RULE amounts to maximizing the difference $(Q_{ii})^2 - q_i^2$. For specific choices of weights this is the MINLEN or DSJ rule [53] or the rule used in [31, 49, 141]. We refer to it as PWO.

First, we investigate some simple properties of the ellipsoid.

- Consider the function $\mathcal{Q}(x; w)$. Suppose that we want to find a good descent direction from the center of the $\{-1, 1\}$ unit hyper cube (which is the origin), in the direction of a satisfying solution. This amounts to maximizing the absolute gradient in the center. Thus we obtain

$$f(p_i) = f(\neg p_i) = |q_i| = \left| \sum_{\ell=2}^{\ell_{\max}} w_\ell(\sigma_\ell(p_i) - \sigma_\ell(\neg p_i)) \right|.$$

For a specific choice of weights this is the *first order probability rule* [74].

- Assume that from the center of the unit hyper cube $x \equiv 0$, we want to make a step Δx along one of the axes, such that $\mathcal{Q}(x + \Delta x; w)$ is maximized. Denoting the ith standard unit vector by u_i, we obtain $(x + \Delta x)^T Q(x + \Delta x) - 2q^T(x + \Delta x) = u_i^T Q u_i - 2q^T u_i$. Letting

$$f(p_i) = Q_{ii} - 2q_i, \quad f(\neg p_i) = Q_{ii} + 2q_i,$$

and using balancing function g_+, we obtain the MWO rule. Using g_* a rule similar to (but not quite the same as) the PWO rule is obtained.

Thus, using simple properties of the ellipsoid well known branching rules are rediscovered. Now let us use some more sophisticated properties to obtain more powerful branching rules. In [97] a branching rule based on an elliptic approximation is investigated, the *maximal eigenvalue rule*, with encouraging results. The idea is that moving along an eigenvector corresponding to the maximal eigenvalue, the value of $\mathcal{Q}(x; w)$ increases the fastest, i.e. the ascent is the steepest. Thus 'branching along this eigenvector' is likely to have great impact on the formula. Obviously, to obtain a branching variable, a single coordinate needs to be selected. A sensible choice is to use the in absolute value largest entry.

Branching Rule 3.1 Maximal Eigenvalue Rule (MEV). *Let $s^{\max}$ be an eigenvector corresponding to the largest eigenvalue of Q. Use $f_{mev}(p_i) = f_{mev}(\neg p_i) = |s_i^{\max}|$.*

For symmetry reasons, no further balancing is performed in this rule. The MEV rule is illustrated in Figure 3.2.

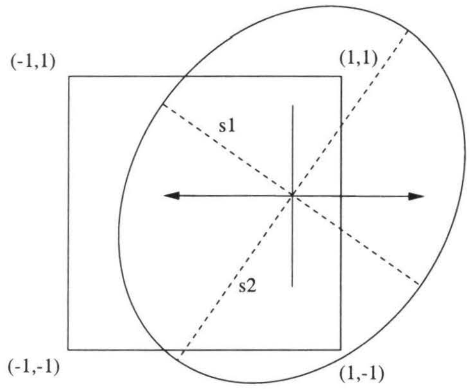


Figure 3.2: The axes of the ellipsoid correspond to the eigenvectors of the matrix Q involved. The branching variable is the one corresponding to the largest element of the shortest axis. It is indicated by the arrows.

Observe that we have not taken the center of the ellipsoid into account thus far. It appears there is no obvious way to do this in the context of the MEV rule.

We now argue that the reasoning behind the MEV rule can be made more precise since it can be applied more specifically to $\{-1, 1\}$ variables. Instead of considering the eigenvectors of Q (which are the axes of the ellipsoid), we can also determine the *axis of the unit hypercube* parallel to which the ascent of the value of $\mathcal{Q}(x; w)$ is the steepest. Let us make this more explicit. By $c_{\mathcal{E}}$ the center of the ellipsoid is denoted, hence $Qc_{\mathcal{E}} = q$. The following theorem provides us with the tools to formalize this idea.

Theorem 3.4.2 *The range $[x_i^{\min}, x_i^{\max}]$ of a variable x_i over the ellipsoid with radius $r^2 \geq -q^TQ^{-1}q$ is given by*

$$x_i^{\min} = (c_{\mathcal{E}})_i - \tilde{r}\sqrt{(Q^{-1})_{ii}} \leq x_i \leq (c_{\mathcal{E}})_i + \tilde{r}\sqrt{(Q^{-1})_{ii}} = x_i^{\max}.$$

Here $\tilde{r}$ is such that $\tilde{r}^2 = r^2 + q^TQ^{-1}q \geq 0$.

Proof: Consider the following optimization problem.

$$(\mathrm{P}_{\mathcal{E}}) \quad \begin{array}{ll} \min & u^Tx \\ \text{s.t.} & x^TQx - 2q^Tx \leq r^2. \end{array}$$

By taking u equal to plus or minus a unit vector this is equivalent to minimizing or maximizing the value of one coordinate over the ellipsoid. A solution to $(\mathrm{P}_{\mathcal{E}})$ is optimal if there exists $\mu \geq 0$ such that

$$\begin{aligned} \mu(Qx - q) + u &= 0 \\ \mu(x^TQx - 2q^Tx - r^2) &= 0. \end{aligned}$$

From the first we obtain $x = Q^{-1}q - (1/\mu)Q^{-1}u$, and substituting this in the second expression we obtain a value for μ, which we can use to eliminate it from the expression for x. We find

$$x = Q^{-1}q - \sqrt{\frac{q^TQ^{-1}q + r^2}{u^TQ^{-1}u}}Q^{-1}u.$$

Noting that $c_{\mathcal{E}} = Q^{-1}q$, and using that u is a positive or negative unit vector, we find that the optimal value to $(\mathrm{P}_{\mathcal{E}})$ is equal to

$$\pm(c_{\mathcal{E}})_i - \tilde{r}\sqrt{(Q^{-1})_{ii}}.$$

In other words, to obtain the minimal value of x_i over the ellipsoid, we need to compute the optimal value of $(\mathrm{P}_{\mathcal{E}})$ with u equal to the ith positive unit vector. Hence we find $x_i^{\min} = (c_{\mathcal{E}})_i - \tilde{r}\sqrt{(Q^{-1})_{ii}}$. On the other hand, taking u equal to minus the ith unit vector, the optimal value of $(\mathrm{P}_{\mathcal{E}})$ is equal to $-x_i^{\max}$. This concludes the proof. □

Note that if $x_i^{\min} > -1$ or $x_i^{\max} < 1$ variable fixings are possible, and if for some i both hold the formula under consideration is unsatisfiable (see also [96]). This rarely occurs; only specific fixings recognizable via single lookahead unit resolution can be discovered.

Using Theorem 3.4.2, we conclude that the steepness of ascent along axis i is proportional to $1/\sqrt{(Q^{-1})_{ii}}$. Thus we arrive at the following rule.

Branching Rule 3.2 Maximal Radius Rule (MR). *Let*

$$f_{mr}(p_i) = f_{mr}(\neg p_i) = \frac{1}{(Q^{-1})_{ii}}.$$

Furthermore, we can take into account the center of the ellipsoid $c_{\mathcal{E}}$. The minimal radius $\tilde{r}$ for which the intersection of the ellipsoid $\mathcal{E}(w)$ and the hyperplanes $x_i = \pm 1$ is nonempty can be computed. To this end, set $x_i^{\max} = 1$ or $x_i^{\min} = -1$ respectively, and use Theorem (3.4.2) to find $\tilde{r}$. The variable for which $\tilde{r}$ must be increased the most appears to be a good candidate. Thus we arrive at the most sophisticated rule to be derived from the elliptic approximation.

Branching Rule 3.3 BALANCED MAXIMAL RADIUS RULE (BMR). *Let*

$$f_{bmr}(p_i) = \frac{1 - (c_{\mathcal{E}})_i}{\sqrt{(Q^{-1})_{ii}}}, \quad f_{bmr}(\neg p_i) = \frac{1 + (c_{\mathcal{E}})_i}{\sqrt{(Q^{-1})_{ii}}},$$

and use balancing function g_*.

Figure 3.3 illustrates the BMR rule. When instead of balancing function g_*, g_+ is used, the BMR rule reduces to the MR rule.

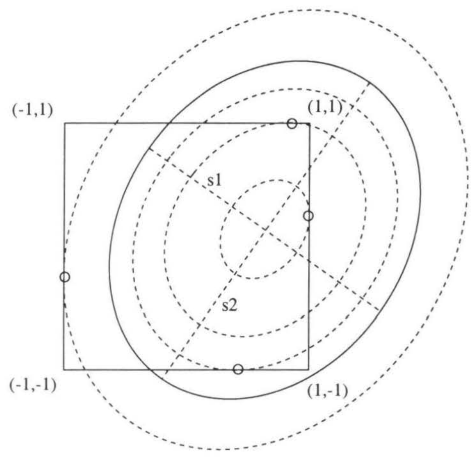


Figure 3.3: The radius $\tilde{r}$ is increased, starting from zero. The smallest ellipsoids having a nonempty intersection with the horizontal (vertical) lines (hyperplanes) are drawn. The respective intersections are highlighted with an 'o'.

Note that one has to be careful if $|(c_{\mathcal{E}})_i| > 1$; in the expression stating the BMR rule it is assumed that $-e \leq c_{\mathcal{E}} \leq e$. If this is not the case the rule should be modified in the obvious way.

To get a first impression of the relative efficiency of the branching rules described above in terms of numbers of nodes of the search tree, we used them to solve a number of randomly generated 3SAT instances of various sizes. For the weights we used $w_2 = 6w_3$; this is similar to the weights used by Dubois et al. [44]. A quick glance at Table 3.1 demonstrates that the rules MR and BMR stand out from the others and it appears that the MR rule performs somewhat better than the BMR rule. Even though these rules seem to be good in practice in the sense that the search trees are kept small, they are

	$m = 50$		$m = 75$		$m = 100$		$m = 125$		$m = 150$	
Rule	mean	std	mean	std	mean	std	mean	std	mean	std
MWO	29.6	8.4	89.1	24.5	246	80	737	302	1805	716
PWO	28.4	7.7	82.9	24.0	233	78	659	242	1591	729
MEV	30.0	8.5	87.6	24.8	245	80	732	253	1704	743
MR	26.5	8.4	78.1	24.0	211	68	617	198	1410	583
BMR	26.2	7.7	77.2	21.4	213	69	640	235	1528	616

Table 3.1: Average number of nodes and standard deviations taken over 100 unsatisfiable 3SAT instances with $n = 4.3m$.

(too) expensive to compute in the context of large scale satisfiability solving. Inverting an $(m \times m)$ matrix requires $\mathcal{O}(m^3)$ operations while for the construction of the matrix Q each of the at most $\ell_{\max} \cdot n$ literal occurrences has to be enumerated. For most other branching rules the latter only is sufficient, indicating that we cannot hope to find a branching rule based on properties of Q that yields a branching variable in better time. Note that in practice the complexity of branching rules can be reduced even further, since $f_{ws}(l)$ can be dynamically updated in each node; analogously Q can be dynamically updated.

3.4.3 Approximation of the MR rule

Let us now try to approximate (B)MR such that computing the branching variable requires $\mathcal{O}(n)$ time as well, while the approximated rule preserves the heuristic quality of the original rule. In the following we restrict ourselves to approximating the MR rule. We assume that $\ell_{max} = \mathcal{O}(1)$ and $n = \mathcal{O}(m)$. Then Q can be regarded as a *diagonally dominant* matrix, by which we mean that

$$\mathcal{O}(\sum_{j \neq i} |Q_{ij}|) = \mathcal{O}(Q_{ii}), \ 1 \leq i \leq m.$$

We are interested in finding the index of the smallest diagonal entry of the inverse matrix of Q. It holds that (see e.g. [123])

$$(Q^{-1})_{ii} = \frac{1}{\det Q} \left(\det Q^{ii} \right), \tag{3.6}$$

where Q^{ii} denotes the matrix Q with row and column i removed. By definition,

$$\det Q = Q_{ii} \det Q^{ii} + \sum_{j \neq i} (-1)^{i+j} Q_{ij} \det Q^{ij}. \tag{3.7}$$

Thus the 'first-order' approximation would be to simply neglect the (second) sum, and to approximate $\det Q$ by the product of the diagonal elements; this corresponds to approximating Q by its diagonal, i.e. $Q \sim \text{diag}(Q)$. It is straightforward to verify that this approximation reduces the balancing function of the BMR rule to the ratio of g_* and g_+, and the MR rule to the MWO rule. Note that if we assume that $Q \sim \text{diag}(Q)$ and $c_{\mathcal{E}} \equiv 0$, the rules MEV, MR, BMR, PWO and MWO in fact all coincide! In any case, the center does not play a part in the rules MEV, MR and MWO (since the linear term q does not occur in these rules); this suggests that MWO is in fact an approximation of MEV, which in turn approximates MR. We conclude that by using a very crude approximation of Q our newly derived rules reduce to the most widely used branching rules.

Now let us approximate (3.7) a bit more precise; instead of completely neglecting the second sum, we use a 'first-order' approximation. For each j, $\det Q^{ij}$ is approximated by its dominant term, which is the term involving as many diagonal elements of the original matrix Q as possible. By construction, *two* original diagonal terms are removed from Q^{ij} (namely Q_{ii} and Q_{jj}) which implies that the term involving $m-2$ original diagonal terms is in this sense optimal. Thus we use as approximation

$$\det Q^{ij} \approx Q_{ij} \det Q^{iijj} \approx Q_{ij} \prod_{k \neq i,j} Q_{kk},$$

yielding

$$\det Q \approx Q_{ii} \det Q^{ii} - \sum_{j \neq i} (Q_{ij})^2 \prod_{k \neq i,j} Q_{kk}$$

from which we obtain that

$$\frac{\det Q}{\det Q^{ii}} \approx Q_{ii} - \sum_{j \neq i} \frac{(Q_{ij})^2}{Q_{jj}}.$$

Note that here we approximated $\det Q^{ii}$ by the product of its diagonal entries to obtain a nice expression again. Now using (3.6) and considering the MR rule, we specify its approximation as follows.

Branching Rule 3.4 MAXIMAL APPROXIMATED RADIUS RULE (MAR). *Let*

$$f_{mar}(p_i) = f_{mar}(\neg p_i) = Q_{ii} - \sum_{j \neq i} \frac{(Q_{ij})^2}{Q_{jj}}.$$

Recalling the assumption that $n = \mathcal{O}(m)$ and $\ell_{\max} = \mathcal{O}(1)$, the maximal number of nonzeros in Q can be computed to be $m + \frac{1}{2}\ell_{\max}(\ell_{\max} - 1)n$. Consequently, the MAR rule can be implemented to require $\mathcal{O}(n)$ operations. Now that we have specified a branching rule, let us address the issue of weighting clauses of different lengths.

3.5 Weighting clauses of different lengths

In the literature the weighting problem occurs often (albeit restricted to clauses of different lengths), but usually not much attention is devoted to it. Jeroslow and Wang [77] argue that the ratio $w_\ell / w_{\ell+1}$ should be taken equal to 2, since clauses of length ℓ rule out twice as many assignments as clauses of length $\ell + 1$. This has been accepted more or less as the default in many later studies, see the rules mentioned in Section 3.3.2. Dubois et al. [44] use different weights $w_\ell = -\log(1 - 1/(2^\ell - 1)^2)$, and show empirically that these have a better performance than the previous in a DPLL algorithm. In other studies the weighting problem is circumvented by considering clauses of different lengths separately (see the examples in Section 3.3.2). The only attempts that we are aware of to weight clauses individually are made by Crawford [30] and Mazure et al. [100]: weights are computed by first running a local search algorithm several times to determine the 'hard-to-satisfy' clauses. Also, in the Chapters 4 and 5 we define optimization problem with clause-weights as variables.

In most of the references cited above, very limited attention is paid to the reasoning behind the choice of the weights; mostly the weights are chosen by experimentation (as far as we can tell). It is our aim to derive weights with (at least some) geometrical motivation: we argue that the volume of the elliptic approximations is an appropriate measure for computing relative weights of clauses.

3.5.1 Using the volume as a measure

The volume of the ellipsoid (3.3) can be computed by the following formula.

$$vol(\mathcal{E}(w)) = \gamma_m \frac{(r^2 + q^T Q^{-1} q)^{\frac{m}{2}}}{\sqrt{\det(Q)}},$$

where $\gamma_m = (2\pi^{\frac{m}{2}})/m\Gamma(\frac{m}{2})$ is a 'constant', depending on the dimension m. As motivation for using the volume as a measure for choosing weights, consider Figure 3.4. There a graph is depicted of the average (normalized) volume as a function of the number of clauses, for a set of 100-variable random 3SAT formulas on and around the threshold [31, 101]. In these experiments all clauses were given equal weight. Note the clear correlation between the volume and (un)satisfiability of the formula. Formulas with associated ellipsoids that have 'small' volume tend to be unsatisfiable, while those with 'large' volume tend to be satisfiable. Moreover, the volume of the elliptic approximation of an unsatisfiable formula is rarely larger than the average volume of the elliptic approximations of the satisfiable ones, and vice versa.

Given a formula Φ, let us consider the volume of its associated ellipsoid as a function of the weights w_k. Intuitively, choosing the weights such that the volume is minimized appears to be a sensible choice. Since for any choice of strictly positive weights all satisfying solutions lie in the interior or on the boundary of the ellipsoid, it appears reasonable to assume that one approximation is tighter than another if the volume of the first is smaller. Thus we are faced with the problem of computing the minimum of the function $vol(\mathcal{E}(w))$ as a function of n variables. The solution to this problem can be approximated in polynomial time via semidefinite programming [134]. However, presently we restrict ourselves to distinguishing between clauses of different lengths only, and to simplify matters even further we consider the 2,3SAT-case only. This allows us to fix the weight for 3-clauses to one, and thus $vol(\mathcal{E}(w))$ reduces to a function v in a single dimension w. Therefore we can compute an approximate stationary point by simple evaluating $v(w)$ for various values of w. Now let us examine the graph of $v(w)$ for a specific 2,3SAT instance, obtained after fixing one variable in a random 3SAT formula. A typical example of such a graph is given in Figure 3.5. The 'optimal' weight is located near $w = 11$. Note that over the interval shown, $v(w)$ is convex; a near-optimal weight can be determined by applying a binary search procedure on the interval $[1, w_{\max}]$, for some sufficiently large $w_{\max}$.

In the top picture of Figure 3.6 the approximately optimal weight as a function of the ratio of 2-clauses and total number of clauses for a large set of randomly generated test problems is shown (note that we restricted the weights to integral values). The problems

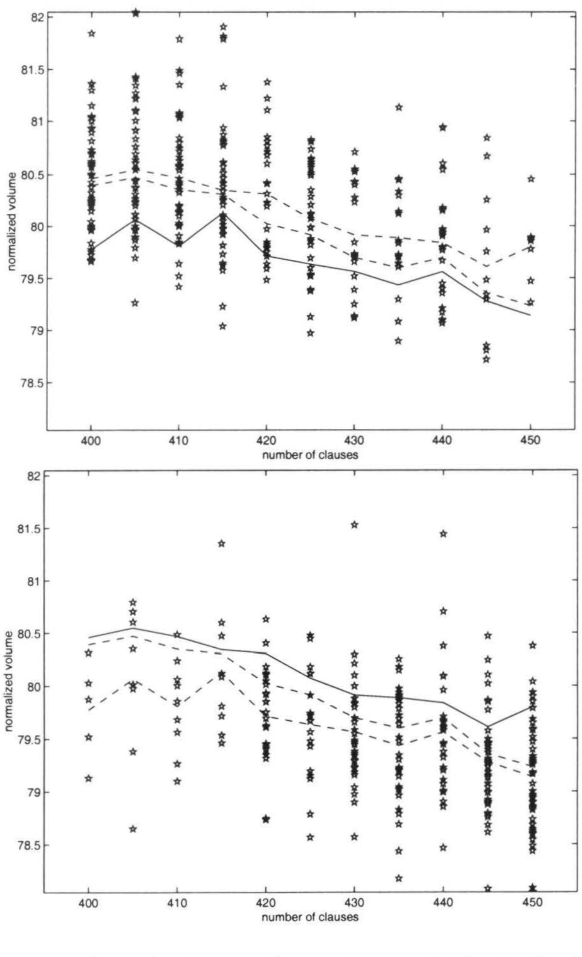


Figure 3.4: In the upper (lower) picture, the pentagons indicate the individual volumes of the (un)satisfiable formulas and their average (upper (lower) dashed line). Also, in the lower (upper) picture the average volume of the (un)satisfiable formulas is drawn (the solid line). In addition, the central dashed lines indicate the overall average volume.

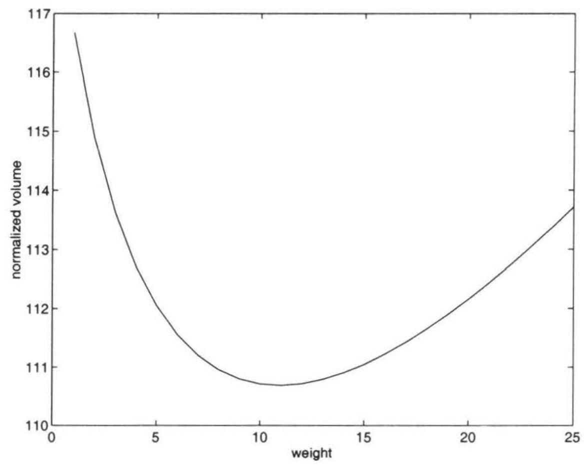


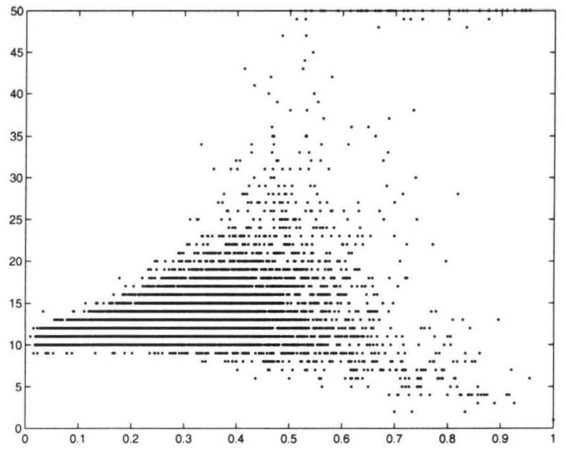
Figure 3.5: A typical example of the volume function $v(w)$.

consisted of $m = 80, 100, 120$ variables and the number of clauses n was chosen to be equal to $4.1, 4.2 \ldots 4.5$ times m (i.e. on and around the threshold). Of each problem size 10 instances were generated. The optimal weights were computed in each node of the DPLL search tree; in these experiments random branching was applied.

On inspection of the graph, it is clear that the optimal weight is 'rooted' at approximately 10-11. For increasing ratios, the behaviour of the optimal weight appears rather wild. However, two clear tendencies can be distinguished. There is an upward tendency, which corresponds to the cases in which the 2SAT subformula 'tends to unsatisfiability' (i.e. its associated ellipsoid has a small volume), and a downward one corresponding to the cases in which the 2SAT subformula 'tends to satisfiability'. Since it does not seem practical to capture both tendencies in one weight function or to compute an approximately optimal weight at each node of the search tree, it appears to be reasonable to fix the weight w at eleven.

Although the behaviour of the optimal weight is rather wild, the behaviour of the optimal volume as a function of the ratio 2-clauses to total number of clauses is quite stable. This is demonstrated in the lower picture of Figure 3.6, which shows the optimal volumes in the nodes of the search trees for the same set of problems as before. Note that to enable comparison of volumes in different dimensions the volumes are normalized, using the volume of the unit hyper cube of appropriate dimension as reference. An obvious trend in this graph is the decrease of volume when the ratio increases. This is partly explained by the fact that the elliptic approximation for the 2SAT subformula is in fact an elliptic representation (see Lemma 2.4.3), and thus tighter than the elliptic approximation of the 3SAT subformula. When the size of the 2SAT subformulas increases, the overall volume decreases.

We conclude this section by mentioning that using different branching strategies similar pictures emerge.



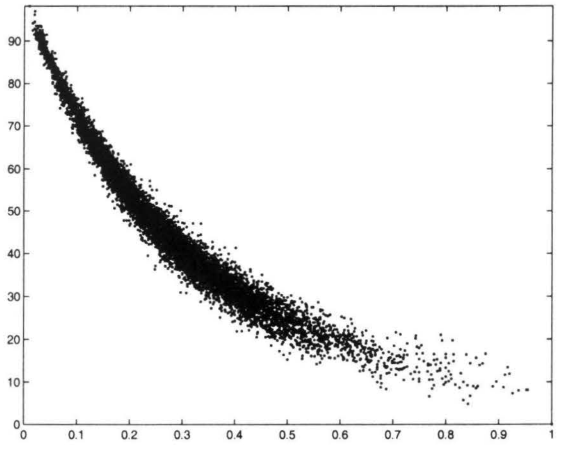


Figure 3.6: The approximately optimal weight (top) and the approximately optimal volume (bottom) as a function of the of ratio 2-clauses and total number of clauses.

3.5.2 Some computational experience

Let us now check whether choosing $w = 11$ really performs better in a DPLL algorithm than choosing for example $w = 2$ (like Jeroslow-Wang [77]) or $w = 6$ (which is comparable to Dubois et al. [44]). In Figure 3.7 the average node counts as a function of w are given, for 50 random 3SAT problems with 200 variables and 860 clauses. The branching rule used is the MAR rule. Since this rule very much relies on the elliptic structure, it is of importance that the ellipsoid approximates the formula (or rather its associated integral polytope) as good as possible. Examining the pictures it is clear that the best node counts indeed are situated on or near $w = 11$. This supports our opinion that the volume is a good measure for computing weights. This is also the weight that we use in the computations in the next section.

Remark: We carried out a similar analysis for the mixed 3,4SAT case. This yielded $w_3 : w_4$ as approximately $100 : 1$, indicating that according to this measure, the importance of 4-clauses relative to 3-clauses is negligible.

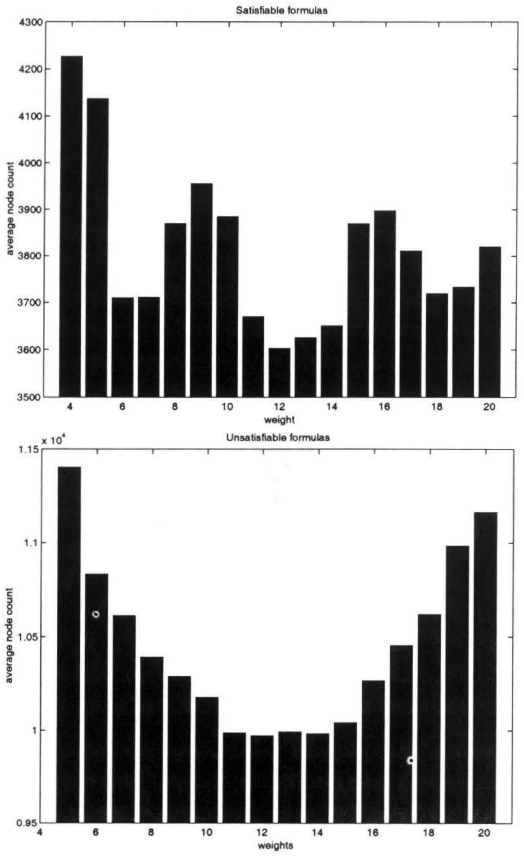


Figure 3.7: Average node counts for increasing weights on satisfiable (upper graph) and unsatisfiable formulas (lower graph).

3.6 Computational results

Let us now compare the performances of a number of branching rules. We compare the MAR, PWO, MWO and Böhm's rule; the latter three being typical examples of branching rules that have been used in practice with good results. In order to do so we have decided to restrict ourselves to random 3SAT formulas on the threshold, i.e. formulas with constant clause length 3 and $n \approx 4.3m$. Recalling Table 1.2 in Section 1.4, it appears this is a class of problems where other enhancements of the DPLL algorithm are not successful. We generated threshold problems ranging in size from 100 to 300 variables, and solved each of these with the four branching strategies mentioned.

First, we have to decide which branch to explore first after a branching variable has been obtained. Obviously, this is of significance for satisfiable formulas only. The following reasoning turned out to give, on average, the best results. Since it is our aim to branch in the direction that is most likely to 'go towards a satisfiable assignment', a natural choice is to exploit the negative gradient of the function $\mathcal{Q}(x; w)$. Considering this in the center of the unit hyper cube suggests that we should first branch to $x_i = \text{sgn}(q_i)$. Recalling equation (3.5), this is equivalent to branching to p_i if $f_{ws}(p_i) \geq f_{ws}(\neg p_i)$, and branching to $\neg p_i$ otherwise.

Furthermore, we have to choose the weights w_ℓ. As we are considering random 3SAT formulas, we can normalize the weight for clauses of length 3 to $w_3 = 1$, and need only specify w_2. For the MWO and PWO rules, following Dubois et al. [44] we set $w_2 = 6$, while for the MAR rule we take $w_2 = 11$ as derived in the previous section. Note that for Böhm's rule no weight needs to be set [12].

We generated 50 instances of each problem size. The results on unsatisfiable and satisfiable instances are given separately in Tables 3.2 and 3.3. We report on average node counts and their standard deviation. The node count refers to the number of times the routine for computing the branching variables was called. We do not report on computation times here. For MWO, PWO and Böhm's rule we used, and where necessary modified, Böhm's implementation. The MAR rule was implemented utilizing less sophisticated data structures. To give an indication of the relative speed of these implementations; using the same branching rule, our implementation is about four times slower than Böhm's on 200-variable instances.

Examining Table 3.2 it is clear that on unsatisfiable instances the MAR rule consistently outperforms the other rules as far as node counts are concerned. However, the computation times for this rule are still larger than for the other rules, at least for the size of instances that we have considered thus far. Concerning satisfiable instances, considering Table 3.3 it is hard to draw any firm conclusions. Overall MAR performs clearly better than the others; the standard deviations however are very large, indicating that neither of the rules perform in a very robust manner.

	MAR		PWO		MWO		Böhm	
m	mean	std	mean	std	mean	std	mean	std
100	247	93	252	93	273	108	263	73
110	317	125	324	127	360	139	351	130
120	447	145	487	145	544	159	547	170
130	723	213	801	245	836	262	863	264
140	1218	394	1285	460	1473	445	1400	443
150	1658	687	1842	713	2056	743	1992	796
160	2557	905	2830	769	3114	1004	3128	936
170	3245	1258	3714	1319	3930	1445	4098	1330
180	5456	1623	6199	1993	6762	2129	6557	1723
190	7398	2867	8220	2738	9234	3571	9388	3152
200	9987	4201	11951	5445	13461	5940	13229	5870
210	16555	7543	19673	9098	22053	9735	22047	8948
220	26366	8905	30186	8325	34170	10102	32787	8583
230	36902	14724	44099	18873	48682	20062	49909	17951
240	49725	22936	58337	24737	67608	30153	67240	28108
250	82142	30042	97534	35480	112256	42696	110928	35825
260	103882	36926	124156	40306	142880	54992	150452	52694
270	150362	69275	184813	80584	208577	96182	221192	99251
280	254648	106168	311969	141098	360071	168637	377618	159390
290	415330	160495	514485	203959	606760	228171	575694	207819
300	506006	221126	629219	266301	726155	298946	753994	299569

Table 3.2: Average node counts and standard deviations for unsatisfiable random 3SAT formulas with $n = 4.3m$.

	MAR		PWO		MWO		Böhm	
m	mean	std	mean	std	mean	std	mean	std
100	102	76	111	82	105	90	105	81
110	204	134	213	140	222	137	217	151
120	202	158	251	196	235	191	234	164
130	282	240	338	367	334	307	425	423
140	579	627	532	502	729	860	630	665
150	396	544	441	371	384	439	491	461
160	859	905	1057	1131	1123	1140	997	1294
170	884	669	1263	1033	974	966	893	892
180	2067	1543	1603	1735	2137	1975	1596	1806
190	3312	2780	4282	3676	4336	4037	3321	3758
200	3702	4116	3671	3110	3609	3674	3569	3403
210	5282	6635	7424	7151	6999	7029	6556	6439
220	7732	10899	7233	11597	8543	11976	8235	13148
230	9620	9502	10510	12744	10042	11132	13540	13120
240	25328	28050	26793	23017	37066	29523	34065	34301
250	32219	33959	52885	46681	52085	37553	50754	56503
260	44854	39082	54564	46668	59662	48905	52914	60002
270	57696	59826	84391	87036	92362	104310	95508	103325
280	78452	53778	85669	61354	102182	77483	88607	71524
290	87573	86079	129594	114544	111281	105078	140353	190855
300	156377	157384	198463	183506	220293	237197	362540	485458

Table 3.3: Average node counts and standard deviations for satisfiable random 3SAT formulas with $n = 4.3m$.

Rule	β
MAR	.0543
PWO	.0552
MWO	.0554
Böhm	.0557

Table 3.4: Results of fitting the node counts to $\alpha 2^{\beta \cdot m}$

In order to get a clearer impression of the growth rate of the number of nodes as a function of the number of variables for unsatisfiable formulas, we fit our results to the following function:

$$\#\text{ nodes} = \alpha 2^{\beta \cdot m}.$$

In Table 3.4 the estimates of β for the various branching rules are given. The values of β confirm that the growth rate for the MAR rule is the lowest.

As indicated before (Table 1.2), *local search* algorithms turn out to be extremely effective on hard random satisfiable instances. To quantify this claim we include Figure 3.8 which depicts a plot of the logarithm of the solution times using the DPLL algorithm with the PWO rule (the upper solid line) and a local search algorithm (the lower solid line; the local search algorithm is briefly explained in Section 1.4.3), on all satisfiable instances listed in Table 3.3. Also drawn are the times plus standard deviation (dashed). It appears that indeed the local search algorithm is several orders of magnitude faster than the DPLL algorithm. For increasing problem sizes, the difference becomes even more substantial.

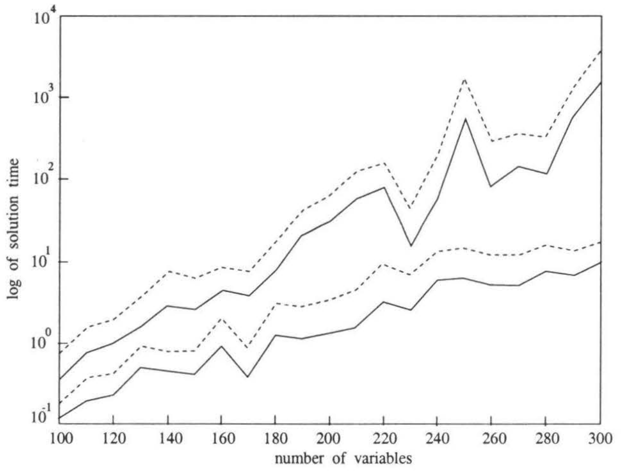


Figure 3.8: Comparing solution times local search and DPLL

3.7 Concluding remarks

In this study we have shown that branching rules can also be motivated from a geometrical viewpoint instead of by a satisfaction [77] or simplification [74] hypothesis. We have specified branching rules based on the properties of an elliptic approximation of SAT formulas. Taking a first order (i.e. linear) approximation of the ellipsoid, well known branching rules that earlier were motivated by the simplification and satisfaction hypotheses, are rediscovered. The most effective branching rule (in terms of size of the search tree) we have found is a *second-order* branching rule since it considers joint occurrences of variables rather than single occurrences. In this respect it has similarities with branching rules proposed by Hooker and Vinay [74] and Dubois et al. [44]. However, in these branching rules a combination of the clause- *and* variable- structure of a formula is considered, whereas in the elliptic approximation we (can) only consider the variable structure. Our conclusion is that the use of 'higher-order' information does help solving satisfiability problems more efficiently. The effect becomes more apparent when the problem size increases. For the problem sizes that we have considered, in terms of computation times the first-order rules are still better; theoretically though, for very large-scale SAT problems we may expect the (second order) MAR rule to become faster, since its computational complexity is comparable to that of the first order rules. On the other hand, even though the size of the search trees can be reduced quite significantly using these sophisticated branching rules, for large hard random 3SAT problems the number of nodes in the tree is substantial. This seems to justify the conclusion that the reduction in size is not sufficient to allow large-scale 3SAT solving as yet. Thus it appears that branching rules should exploit structure in formulas even more. Recently, implementations have become available that involve branching rules making explicit use of *both* the variable- and clause-structure of a formula by employing lookahead unit resolution. Li [92] reports that hard 3CNF problems up to a size of 500 variables [92] can be adequately handled.

As a byproduct, it has become clear that the 'default' choice of 2 for the ratio $w_\ell/w_{\ell+1}$ performs rather poorly, especially on hard unsatisfiable instances of SAT. This observation holds irrespective of the branching rule used. Our experiments suggest that this ratio should be chosen significantly higher. For the MAR rule, which attempts to make full use of the elliptic structure, the computation of weights via volumes (yielding $w_2/w_3 \approx 11$) results in a considerable reduction in search tree size. The rules PWO and MWO appear less sensitive to the choice of weights, although w_2/w_3 should be chosen to be at least 5. This suggests that the weighting problem is dependent on the branching rule used, rather than a separate problem.

4

A Two-Phase Algorithm For A Class Of Hard Satisfiability Problems

A specific class of polynomially solvable SAT problems is considered: the conjunctions of (nested) equivalencies (CoE). CNF translations of such formulas are extremely hard for branching and resolution algorithms. We derive a characterization of CNF translations of CoE formulas. Via this characterization, using linear programming certain CNF (sub)formulas that are equivalent to CoE formulas can be recognized in polynomial time. Thus several notoriously difficult benchmarks are found to be polynomially solvable. In addition, a set of previously unsolved benchmarks is solved in a matter of minutes, using the LP approach in conjunction with an extended version of the DPLL algorithm.

4.1 Introduction

Since the satisfiability problem is NP-complete, it is in general hard to solve. However, in practice SAT problems can often be easily solved when the proper algorithm is applied, even if its worst-case running time is exponential. Furthermore, there are various specific classes of SAT formulas for which polynomial time algorithms exist. For example, 2SAT formulas (i.e. formulas in which each clause contains at most two literals), and Horn formulas (in which each clause contains at most one positive literal) are solvable in linear time [6, 41], while various generalizations of these classes are polynomially solvable as well [15, 22, 23, 50, 113].

Given a CNF formula Φ it might be worthwhile to know whether Φ belongs to one of the polynomially solvable classes, or whether it is equivalent to a polynomially solvable formula. While establishing the first is mostly straightforward (although there are cases in which it is not clear how to establish membership of a certain class; for example *extended Horn* [113]) checking the second is difficult. Creignou proved that, given a CNF formula Φ, the problem of the existence of an 'easy' equivalent formula is coNP-complete [33].

In this chapter we consider another class of polynomially solvable SAT problems, the *conjunctions of (nested) equivalencies* (CoE). In the literature, CoE formulas are also known as XOR ('exclusive or') SAT formulas, which were shown to be polynomially solvable by Schaefer [111]. We address the problem of recognizing CoE formulas in CNF formulas. A special case arises from so-called *balanced (CNF) formulas* [43]. Balanced formulas are balanced with respect to the number of variable occurrences, and positive and negative occurrences of individual variables. The notion of *doubly balancedness* [96] is related to joint occurrences of variables. Doubly balanced formulas can be concluded to be equivalent to CoE formulas by means of their *elliptic approximation* such as introduced in Section 2. By making use of polynomial representations of satisfiability problems such as used by Gu [61], a more general characterization of CNF translations of CoE formulas is obtained. This involves *balanced polynomial representations* (BPR), a concept that we define later on. We will show that any CNF formula with BPR is equivalent to a CoE formula and as such can be solved efficiently. Moreover, using linear programming, a CNF formula can be checked to have BPR, while using the same formulation subformulas with BPR can be identified as well, both in polynomial time [87]. The solution of such a subformula can speed up the solution of the full formula substantially. To this end, the DPLL-algorithm is extended to solve conjunctions of CNF and CoE formulas (rather than plain CNF formulas).

Even though CoE formulas can be solved in polynomial time by a special purpose algorithm, their CNF translations are hard to solve for most algorithms. Indeed, Tseitin [129] introduced a specific kind of CoE formulas (in which each variable occurs exactly twice) to prove an exponential lower bound on the running time of regular resolution. In general, balanced formulas are hard for branching and other resolution-like algorithms [43]. It turns out that several of the well known DIMACS benchmarks are in fact equivalent to CoE formulas and thus can be solved efficiently. Furthermore, several benchmarks have large subformulas with BPR. Some of these benchmarks were as yet unsolved; in fact, solving them was posed as a challenge by Selman et al. [116]. Using the solution of the CoE subformula as a first step, and subsequently applying the extended DPLL-algorithm, these benchmarks are solved within five minutes.

It may be noted that specific CoE formulas, namely those arising from a particular kind of doubly balanced formulas, can also be solved efficiently by making use of the notion of *symmetry* as introduced by Benhamou and Saïs [8, 44]. A CNF formula is said to contain symmetries if it remains invariant under a permutation of variable names. However, in general the CoE formulas need not be symmetric in this sense.

This chapter is organized as follows. In the next section we introduce some notation and discuss the preliminaries. Subsequently we review an algorithm for solving CoE formulas. In Section 4.4 we address the problem of recognizing CoE formulas in CNF formulas. As a special case we consider doubly balanced formulas and show that by their elliptic approximation these can be seen to be equivalent to CoE formulas. Then the notion of doubly balancedness is generalized by making use of polynomial representations of satisfiability problems. In the subsequent sections we give a linear programming formulation to identify CoE (or unsatisfiable) (sub)formulas and describe the extended version of the DPLL to solve conjunctions of CNF and CoE formulas. The last sections of this chapter contain computational results on several DIMACS benchmarks and concluding remarks.

4.2 Preliminaries and notation

We use the notion introduced in Chapter 1 (Section 1.1) and Chapter 2. Briefly, a CNF formula is a conjunction of n clauses $\mathbf{C}_k$, each of the form

$$\bigvee_{i \in I_k} p_i \vee \bigvee_{i \in J_k} \neg p_i,$$

and $A \in \mathbb{R}^{n \times m}$ is the associated clause-variable matrix (see eq. (2.1)). In addition, apart from CNF formulas, we also use *CoE formulas*, which are also known as XOR SAT formulas. Such formulas are solvable in polynomial time (Schaefer [111]) as opposed to CNF formulas which are in general NP-complete. In the next section a polynomial-time algorithm for CoE formulas is reviewed. A CoE formula Ψ is the conjunction of t *equivalency-clauses*, where each equivalency-clause $\mathbf{Q}_k$ is a (nested) equivalency of literals or its negation. In the first case we refer to $\mathbf{Q}_k$ as a *positive equivalency-clause*, otherwise it is called a *negative equivalency-clause*. In general, an equivalency-clause is denoted as

$$\mathbf{Q}_k = [\neg] \underset{i \in S_k}{\overset{\leftrightarrow}{\bigcup}} p_i, \tag{4.1}$$

where the square brackets indicate the optionality of the negation. Note that to satisfy a positive equivalency-clause, an *even* number of literals must be $false$, while for a negative one an *odd* number of literals must be $false$. In the following we associate an indicator δ_k with an equivalency-clause $\mathbf{Q}_k$, and let $\delta_k = 1(-1)$ if $\mathbf{Q}_k$ is a positive (negative) equivalency-clause.

Example 4.2.1 An example of a (negative) equivalency-clause is

$$\neg(p_2 \leftrightarrow p_4 \leftrightarrow p_7),$$

which is $true$ if either one or all three of its literals are $false$. For the reader who is unfamiliar with nested equivalencies/bi-implications, let us verify this. Note that $p_i \leftrightarrow p_j$ is $true$ if and only if p_i and p_j have the same truth value (obviously $\neg p_i \leftrightarrow \neg p_j$ has the same set of solutions and thus any equivalency-clause can be reduced to the standard form (4.1)). Assuming that p_2 and p_4 have the same value, the equivalency-clause reduces to $\neg(true \leftrightarrow p_7)$ implying that p_7 must be $false$, while if p_2 and p_4 have opposite values we find $\neg(false \leftrightarrow p_7)$, so p_7 must be $true$. In the first case either one (i.e. only p_7) or three literals are $false$, while in the second case either p_2 or p_4 is $false$. □

4.3 Solving conjunctions of equivalencies

Let us now review an algorithm for solving CoE formulas, which is in fact Gaussian elimination in $\mathbb{Z}_2$ [111]. Suppose we are given a CoE formula Ψ. Obviously, for any satisfying assignment it holds that

$$p_j \leftrightarrow \left(\underset{i \in S_k \setminus \{j\}}{\overset{\leftrightarrow}{\bigcup}} p_i \right), \tag{4.2}$$

for a positive equivalency-clause $\mathbf{Q}_k$, $1 \leq k \leq t$, and any $j \in S_k$. If $\mathbf{Q}_k$ is a negative equivalency-clause, the right hand side of (4.2) must be negated. This observation can be exploited to obtain a polynomial-time algorithm. All but one occurrences of p_j can be eliminated by making use of equivalency (4.2), to obtain a formula Ψ' that is equivalent to formula Ψ. Note that the value of p_j is uniquely determined by the values of the other variables in $\mathbf{Q}_k$. We call such a variable p_j a *dependent* variable. Initially, all the variables are said to be independent and contained in the set of independent variables $\mathcal{I}$. Let us use the notation

$$S_k \oplus S_l = (S_k \cup S_l) \backslash (S_k \cap S_l).$$

This is convenient, since when we use expression (4.2) to eliminate p_j from equivalency-clause $\mathbf{Q}_l$ we obtain

$$p_j \leftrightarrow \left(\mathop{\leftrightarrow}\limits_{i \in S_l \backslash \{j\}} p_i \right) \equiv \left(\mathop{\leftrightarrow}\limits_{i \in S_k \backslash \{j\}} p_i \right) \leftrightarrow \left(\mathop{\leftrightarrow}\limits_{i \in S_l \backslash \{j\}} p_i \right) \equiv \mathop{\leftrightarrow}\limits_{i \in S_k \oplus S_l} p_i.$$

Note that if $S_k = \{j\}$, this substitution performs unit resolution; the length of equivalency-clause $\mathbf{Q}_l$ then reduces by one. If $S_k \oplus S_l = \emptyset$, while $\delta_k = -\delta_l$ an inconsistency is detected, implying that the formula under consideration is unsatisfiable. This occurs if and only if $\mathbf{Q}_k$ is the negation of $\mathbf{Q}_l$. If no inconsistency is detected, the algorithm terminates when the CoE formula is rewritten to a form where each equivalency-clause contains exactly one dependent variable. After termination of the algorithm, *all* satisfying solutions can be constructed by assigning all possible combinations of truth values to the independent variables. Algorithm SOLVE_CoE is summarized in Figure 4.1. In the outer loop each equivalency-clause is considered at most once; after it is considered it is labelled. Note that at the end of each outer loop, unlabelled clauses consist of independent variables only, while any labelled clause either contains a dependent variable or is empty. The algorithm returns that Ψ is a contradiction, or the rewritten formula Ψ and the set of independent variables $\mathcal{I}$ using which all satisfying assignments can be constructed.

Example 4.3.1 Let the formula Ψ be given by

$$\Psi = (p_1 \leftrightarrow p_2 \leftrightarrow p_3) \wedge (p_2 \leftrightarrow p_3 \leftrightarrow p_4) \wedge \neg(p_1 \leftrightarrow p_3 \leftrightarrow p_4).$$

Initializing, we have $\mathcal{I} = \{p_1, p_2, p_3, p_4\}$ and all clauses are unlabelled.

Iteration 1 We choose $l = 1$, $j = 1$. Note that $j \notin S_2$, but $j \in S_3$, hence we carry out the substitution. We have that $S_1 \oplus S_3 = \{2, 4\}$ and $\delta_1 \delta_3 = -1$, thus

$$\Psi := (p_1 \leftrightarrow p_2 \leftrightarrow p_3) \wedge (p_2 \leftrightarrow p_3 \leftrightarrow p_4) \wedge \neg(p_2 \leftrightarrow p_4),$$

and $\mathcal{I} = \{p_2, p_3, p_4\}$.

Iteration 2 Now let us choose $l = 2$, $j = 2$ thus $\mathcal{I} := \{p_3, p_4\}$. Performing the substitution for $k = 1$ and $k = 3$ we find

$$\Psi := (p_1 \leftrightarrow p_4) \wedge (p_2 \leftrightarrow p_3 \leftrightarrow p_4) \wedge \neg p_3.$$

Iteration 3 Finally $l = 3$, $j = 3$, hence $\mathcal{I} := \{p_4\}$. We find

$$\Psi := (p_1 \leftrightarrow p_4) \wedge \neg(p_2 \leftrightarrow p_4) \wedge \neg p_3.$$

Since $|S_3| = 1$, the last action is equivalent to applying unit resolution.

```
Initialize the set $\mathcal{I} = \{p_1, \ldots, p_m\}$.
while not all clauses are labelled do
   choose an unlabelled clause $\mathbf{Q}_l$;
   choose a variable $p_j \in S_l$;
   $\mathcal{I} := \mathcal{I} \backslash \{p_j\}$;
   for $k = 1$ to $t$ do
      if $j \in S_k$ and $k \neq l$ then
         $S_k := S_k \oplus S_l$; $\delta_k := \delta_k \delta_l$;
         if $S_k = \emptyset$ then
            if $\delta_k = -1$ return(contradiction);
            else label clause $\mathbf{Q}_k$;
         endif
      endif
   endfor
   label clause $\mathbf{Q}_l$;
endwhile
return(satisfiable, $\mathcal{I}, \Psi$);
```

Figure 4.1: Algorithm SOLVE_CoE

All clauses are labelled now, hence the algorithm terminates. The single independent variable p_4 can be arbitrarily set, and substituting it in Ψ we can construct two satisfying assignments, corresponding to p_4 and $\neg p_4$ respectively; $(p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge p_4)$ and $(\neg p_1 \wedge p_2 \wedge \neg p_3 \wedge \neg p_4)$. □

Let us consider the complexity of the algorithm. It requires $\min\{m, t\}$ iterations, since each time the outer loop is executed the number of independent variables and the number of unlabelled clauses decrease. In each iteration all equivalency-clause are considered once, and the length of these clauses is bounded by m. Thus we have the following complexity bound.

Lemma 4.3.2 *The algorithm runs in $\mathcal{O}(mt \cdot \min\{m, t\})$ time.*

We conclude that this algorithm solves CoE formulas in polynomial time. Let us now turn to the issue of recognizing CoE formulas by their CNF translation.

4.4 Recognition of CoE formulas in CNF formulas

First, we show that members of a specific class of CNF formulas can be concluded to be equivalent to CoE formulas. Subsequently we consider more general CNFs.

4.4.1 A special case: doubly balanced formulas

In this section we restrict ourselves to 3SAT formulas. We consider a specific class of 3CNF formulas and show that these are equivalent to CoE formulas. Let us consider the

CNF equivalent of an equivalency-clause of length 3, say $p \leftrightarrow q \leftrightarrow r$, using De Morgan's laws. Then

$$\Phi = (p \vee \neg q \vee \neg r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee r) \wedge (p \vee q \vee r). \tag{4.3}$$

Note the particular structure of this CNF formula. A formula with this structure is called *doubly balanced* [43, 96]. Doubly balanced formulas are defined as follows.

Definition 4.4.1 *A 3SAT formula Φ is called* doubly balanced *if*

1. *Of each proposition p_i, the number of unnegated and negated occurrences are equal; equivalently $\sigma(p_i) = \sigma(\neg p_i)$ or*

$$\sum_{k=1}^{n} a_{ki} = 0, \; i = 1, \dots, m.$$

2. *For any two propositions p_i and p_j, the number of clauses in which both appear simultaneously with the same sign (i.e. both are negated or both are unnegated) is equal to the number of clauses in which both appear simultaneously with opposite signs (i.e. one appears negated and the other unnegated), or equivalently*

$$\sum_{k=1}^{n} a_{ki}a_{kj} = 0, \; i, j = 1, \dots, m, i \neq j.$$

Examining (4.3) suggests that doubly balanced formulas are equivalent to CoE formulas. That this is indeed the case is the main result of the present section. Let us state it as a theorem.

Theorem 4.4.2 *A doubly balanced formula is equivalent to a CoE formula and as such can be solved in polynomial time.*

The remainder of this section is concerned with proving this theorem. In the proof we make use of elliptic approximations of satisfiability problems; see Section 2.

Lemma 4.4.3 *Let Φ be a 3CNF-formula with associated clause-variable matrix A. The ellipsoid*

$$\mathcal{E}_3 = \{x \in \mathbb{R}^m | x^T A^T A x - 2e^T A x \leq 3n\}$$

contains all satisfying assignments of Φ.

Proof: See Theorem 2.4.1. Let $w = e$ and note that $r_k = 3$. □

Considering the ellipsoid, it is clear that certain contradictory assignments $x \in \{-1, 1\}^m$ may also be contained in it; therefore we speak of an *approximation.* In general most contradictory assignments are not contained in the ellipsoid, and thus using specific properties of the ellipsoid such as its eigenvalue structure and its center, effective branching rules and satisfiability tests are obtained; see the previous chapter and [97].

If the ellipsoid is *normalized and centered,* i.e. it is centered at the origin and its axes are parallel to the unit vectors, these heuristics do not distinguish between the variables and thus are of no use. However, the structure of such ellipsoids can be exploited in a different way. Note that for these ellipsoids $A^T A$ is diagonal, and $A^T e \equiv 0$. Let us first establish the following easy lemma.

Lemma 4.4.4 *Φ is doubly balanced if and only if A^TA is a diagonal matrix and $A^Te \equiv 0$.*

Proof: Follows straightforwardly from Definition 4.4.1. □

We conclude that the elliptic approximation of a doubly balanced formula is normalized and centered. Normalized, centered ellipsoids have the following property.

Lemma 4.4.5 *Assume that the ellipsoid $\mathcal{E}_3$ is normalized and centered. Then each vector $x \in \{-1, 1\}^m$ lies on its boundary.*

Proof: Using that A^TA is diagonal, $A^Te \equiv 0$ and $x_i^2 = 1$ for $x_i \in \{-1, 1\}$, we find that

$$x^TA^TAx - 2e^TAx = \sum_{i=1}^{m}(A^TA)_{ii}x_i^2 = \sum_{k=1}^{n}\ell(\mathbf{C}_k) = 3n.$$

Here we also used Lemma 2.4.5. □

Corollary 4.4.6 *If all assignments lie on the boundary of $\mathcal{E}_3$, then for any satisfying assignment $x \in \{-1, 1\}^m$, a_k^Tx equals either -1 or 3 for each $k = 1, \dots, n$.*

Proof: First note that for any satisfying assignment $a_k^Tx \in \{-1, 1, 3\}$. Next, assume that for a satisfying assignment $a_k^Tx = 1$, for some k. Then $(Ax - e)^T(Ax - e) \leq 4(n - 1)$, implying that $x^TA^TAx - 2e^TAx < 3n$, thus arriving at a contradiction. □

Hence, for each clause a satisfying assignment either satisfies *exactly one* of its literals or *all three*. This implies that each clause k can be regarded as a nested equivalence of its three literals. Let us state this in a lemma.

Lemma 4.4.7 *The requirement that of a clause of length three $\mathbf{C}_k$ either one or all three literals must be satisfied is expressed by the equivalency-clause*

$$\mathbf{Q}_k = [\neg]\mathop{\overleftrightarrow{\bigcup}}_{i \in S_k} p_i,$$

where $S_k = I_k \cup J_k$ and the negation operator is present (and $\delta_k = -1$) if and only if $|J_k|$ is odd.

Proof: The correctness of the lemma is verified by the truth table in Table 4.1. The first three columns show all possible truth valuations of the three variables involved (denoted by p, q, r). The subsequent two columns indicate whether the associated equivalency-clauses under the given assignments evaluate to *true* (t) or *false* (f). The last four columns indicate whether the associated clauses with $|J_k| = 0, 1, 2, 3$ negative literals and the additional requirement that exactly one or all three literals are *true*, evaluate to *true* (t) or *false* (f). □

We conclude that a CoE formula can be constructed that is fully equivalent to the original doubly balanced CNF formula, thus completing the proof of Theorem 4.4.2. Let us return for a moment to the example formula (4.3). The CNF formula Φ is doubly balanced and consequently (using Lemma 4.4.7) it is equivalent to

$$\Psi = (p \leftrightarrow q \leftrightarrow r) \wedge (p \leftrightarrow q \leftrightarrow r) \wedge (p \leftrightarrow q \leftrightarrow r) \wedge (p \leftrightarrow q \leftrightarrow r),$$

thus, indeed, $\Psi = p \leftrightarrow q \leftrightarrow r$.

p	q	r	$p \leftrightarrow q \leftrightarrow r$	$\neg(p \leftrightarrow q \leftrightarrow r)$	$\|J_k\| = 0$	$\|J_k\| = 1$	$\|J_k\| = 2$	$\|J_k\| = 3$
t	t	t	t	f	t	f	t	f
t	t	f	f	t	f	t	f	t
t	f	t	f	t	f	t	f	t
t	f	f	t	f	t	f	t	f
f	t	t	f	t	f	t	f	t
f	t	f	t	f	t	f	t	f
f	f	t	t	f	t	f	t	f
f	f	f	f	t	f	t	f	t

Table 4.1: Proof of Lemma 4.4.7.

Note that the complexity of constructing the elliptic approximation is linear in the number of clauses, and thus checking whether a formula is doubly balanced can be done efficiently. In the next section we address the issue of recognizing CoE formulas in CNF formulas in general (i.e. not restricted to 3SAT and doubly balancedness).

4.4.2 A general characterization of CoE formulas

The ellipsoid considered in the previous section is in fact the second order Taylor truncation of a full representation of satisfiability problems (see Appendix B). We now investigate such full polynomial representations of general CNF formulas which enable us to give a *general characterization* of CNF translations of CoE formulas.

Let us first consider a polynomial representation of a CoE formula. Recall that with each proposition letter p_i a $\{-1,1\}$-variable x_i is associated. It holds that the equivalency-clause $\mathbf{Q}_k$ is satisfied if and only if (see (4.1))

$$Q_k(x) = \delta_k \prod_{i \in S_k} x_i = 1,$$

since this equation is satisfied if and only if $\delta_k = 1$ (-1) and an even (odd) number of x_i, $i \in S_k$, variables equal -1. Note that for any assignment $x \in \{-1,1\}^m$, $Q_k(x) = \pm 1$. Thus we have a concise alternative formulation of Ψ.

$$\text{(CoE)} \quad \text{find } x \in \{-1,1\}^m \text{ such that } \sum_{k=1}^{t} Q_k(x) = \sum_{k=1}^{t} \delta_k \prod_{i \in S_k} x_i = t.$$

Conversely, it is easy to see that any problem of the form (CoE) can be directly translated to a CoE formula. Let us now derive a condition under which a CNF formula Φ can be reduced to the form (CoE) (and thus is polynomially solvable).

Consider a clause $\mathbf{C}_k$ and its associated linear inequality $a_k^T x \geq 2 - \ell(\mathbf{C}_k)$. A $\{-1,1\}$-vector x satisfies $a_k^T x \geq 2 - \ell(\mathbf{C}_k)$ if and only if

$$P_k(x) = \prod_{i \in I_k} (1 - x_i) \prod_{j \in J_k} (1 + x_j) = \prod_{i=1}^{m} (1 - a_{ki} x_i) = 0. \tag{4.4}$$

Indeed this equation is satisfied if and only if at least one $x_i = 1$, $i \in I_k$ or $x_i = -1$, $i \in J_k$. If x is not a satisfying assignment it holds that $P_k(x) = 2^{|I_k \cup J_k|} > 0$. Denote $\mathcal{M} = \{1, \dots, m\}$. In general, $x \in \{-1,1\}^m$ is a satisfying assignment of a formula Φ, if and only if

$$\mathcal{P}(x) = \sum_{k=1}^{n} P_k(x) = n + \sum_{I \subseteq \mathcal{M}} (-1)^{|I|} \sum_{k=1}^{n} \prod_{i \in I} a_{ki} x_i = 0,$$

where in principal I runs through all subsets of $\mathcal{M}$ ($I \neq \emptyset$). Note that the number of subsets that has to be taken into account can be restricted substantially, since in fact only subsets $I \subseteq \mathcal{M}$ for which $I \subseteq I_k \cup J_k$ for some $k = 1, \dots, n$ need to be considered. In general, for a clause with length $\ell(\mathbf{C}_k)$, $2^{\ell(\mathbf{C}_k)} - 1$ coefficients need to be computed.

Obviously, the multiplicative representation (4.4) remains valid when multiplied with a nonzero weight w_k. Let us associate a strictly positive real weight w_k with each clause. We use the notation

$$c_I(w) = (-1)^{|I|} \sum_{k=1}^{n} w_k \prod_{i \in I} a_{ki}, \tag{4.5}$$

where $I \subseteq \mathcal{M}$. Then the satisfiability problem has the following *weighted polynomial representation* (note that this formulation slightly differs from the one introduced in Section 2.3):

$$\text{(WPR)} \quad \text{find } x \in \{-1,1\}^m \text{ such that } \mathcal{P}(x;w) = \sum_{k=1}^{n} w_k + \sum_{I \subseteq \mathcal{M}} c_I(w) \prod_{i \in I} x_i = 0.$$

Observe that by construction $\mathcal{P}(x;w) \geq 0$ for any $x \in \{-1,1\}^m$. Strict inequality implies that the corresponding CNF formula is unsatisfiable.

Now we can generalize the notion of doubly balancedness (that is restricted to 3SAT) to a notion of balancedness for general SAT formulas. Let us give a definition.

Definition 4.4.8 *Consider the weighted polynomial representation (WPR). We call the polynomial function* $\mathcal{P}(x;w)$ balanced *if*

$$\sum_{I \subseteq \mathcal{M}} |c_I(w)| = \sum_{k=1}^{n} w_k.$$

Furthermore, $\mathcal{P}(x;w)$ *is called* (strictly) positive *if*

$$\sum_{I \subseteq \mathcal{M}} |c_I(w)| < \sum_{k=1}^{n} w_k.$$

Assume we are given a SAT formula Φ and its weighted polynomial representation (WPR).

- If $\mathcal{P}(x;w)$ is balanced, we say that Φ has *balanced polynomial representation* (BPR).
- Similarly, if $\mathcal{P}(x;w)$ is positive, we say that Φ has *positive polynomial representation* (PPR).

Example 4.4.9 Consider the formula $\Phi = (p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (\neg p \vee \neg q)$. Φ has BPR; assigning the weights $\frac{1}{2}$, $\frac{1}{2}$ and 1 to the clauses, it is easily verified that $\mathcal{P}(x;w) = 2 + 2x_p x_q$. An example of a formula with PPR is $\Phi \wedge (p \vee \neg q) \wedge (\neg p \vee q)$. Using the same weights as before and giving the additional clauses a weight of 1, we find that $\mathcal{P}(x;w) = 4$. □

Lemma 4.4.10 *If Φ has balanced polynomial representation, then it is equivalent to a conjunction of equivalencies.*

Proof: We need to show that for balanced weighted polynomials $\mathcal{P}(x; w)$, (WPR) (and hence Φ) can be reduced to the form (CoE). Note that if $\mathcal{P}(x; w)$ is balanced, then for any feasible vector $x \in \{-1, 1\}^m$ it must hold that

$$c_I(w) \prod_{i \in I} x_i = -|c_I(w)|,$$

for all $I \subseteq \mathcal{M}$. This implies that for all subsets $I \subseteq \mathcal{M}$ for which $c_I(w) \neq 0$,

$$\prod_{i \in I} x_i = \begin{cases} 1 & \text{if } c_I(w) < 0, \\ -1 & \text{if } c_I(w) > 0. \end{cases}$$

Enumerating the $k = 1, \ldots, t \leq n$ sets $I \subseteq \mathcal{M}$ for which $c_I(w) \neq 0$, we take $\delta_k = -\text{sgn}(c_I(w))$ and $S_k = I$ to prove the lemma. □

Example 4.4.11 Returning to the example for a moment, by Lemma 4.4.10 the formula Φ is concluded to be equivalent to the formula $\neg(p \leftrightarrow q)$. □

Let us now state a theorem.

Theorem 4.4.12 *Given are a propositional formula Φ and its weighted polynomial representation (WPR).*

- *If Φ has a positive polynomial representation, it is unsatisfiable.*
- *If Φ has a balanced polynomial representation, it is equivalent to a CoE formula and can be solved in in $\mathcal{O}(mn \cdot \min\{m, n\})$ time, using algorithm* SOLVE_CoE.
- *More general, if*

$$\sum_{I \subseteq \mathcal{M}} |c_I(w)| = n + 2z,$$

a satisfying solution, or proof that no solution exists, can be found using a modified version of algorithm SOLVE_CoE *with complexity*

$$\mathcal{O}\left(\binom{n + 2z}{z} mn \cdot \min\{m, n\} \right).$$

Proof: The first statement follows from the fact that $\mathcal{P}(x; w) > 0$ for any $x \in \{-1, 1\}^m$; the second statement is clear from Lemma 4.4.10. Let us now consider the third statement. Assume that all nonzero coefficients $c_I(w)$ equal ± 1, and consider (WPR). For any satisfying assignment exactly z of the terms $c_I(w) \prod_{i \in I} x_i$ must equal '1' instead of '-1' to imply that $\mathcal{P}(x; w) = 0$. Thus z out of $n + 2z$ sets $I \subseteq \mathcal{M}$ (with $c_I(w) \neq 0$) need to be chosen to contribute '+1' to $\mathcal{P}(x; w)$, and subsequently algorithm SOLVE_CoE can be applied. This concludes the proof. □

Obviously, Φ having a positive polynomial representation is merely a *sufficient* condition for its unsatisfiability. Similarly, Φ having a balanced polynomial representation is a *sufficient* condition only for it to be equivalent to a CoE formula. Furthermore, concerning statement 3 of this Theorem, it may be interesting to mention that in specific cases, by making use of the particular numerical values of the coefficients $c_I(w)$, the complexity bound given might turn out to be somewhat pessimistic. We do not pursue this here.

To avoid ambiguities in the terminology, let us stress the difference between BPR and CoE subformulas. If a CNF formula has BPR or a BPR subformula, this implies that it is equivalent to a CoE formula, or that it has a subformula that is equivalent to a CoE subformula. On the other hand, if a CNF (sub)formula is equivalent to a CoE formula, the CNF formula does not necessarily have BPR.

To relate the result of the previous section with that of the present section, let us briefly consider 3SAT formulas, assuming that all weights are set to one. Then

$$\mathcal{P}(x;w) = n - \sum_{k=1}^{n}\left[\sum_{i=1}^{m} a_{ki}x_i - \sum_{i=1}^{m}\sum_{j\neq i} a_{ki}a_{kj}x_ix_j + \sum_{i=1}^{m}\sum_{j\neq i}\sum_{l\neq i,j} a_{ki}a_{kj}a_{kl}x_ix_jx_l\right].$$

From this it is clear that doubly balanced formulas either have a balanced or positive polynomial representation; the linear and bilinear terms vanish, while at most n (trilinear) terms remain (see Definition 4.4.1). However, 3SAT formulas need not be doubly balanced to have such a representation. Consider the following example.

Example 4.4.13 Given is the formula Φ.

$$\Phi = (p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee \neg s) \wedge (\neg p \vee \neg q \vee s).$$

Obviously, Φ is not doubly balanced in the sense of Definition 4.4.1, but (assuming that all weights are equal to one)

$$\mathcal{P}_\Phi(x) = 4 + 4x_px_q.$$

Thus, by Definition 4.4.8, $\mathcal{P}_\Phi(x)$ is balanced. □

Hence the notion of balanced polynomial representations is stronger than doubly balancedness. Obviously, checking the first is, in general, also computationally more involved. If the maximum clause length is bounded by ℓ it requires computing (at most) $(2^\ell - 1)n$ coefficients; doubly balancedness can be checked by computing $4n$ coefficients.

4.5 A two-phase algorithm

To speed up solving satisfiability problems, it may be helpful to detect CoE subformulas, or even unsatisfiable subformulas. If an unsatisfiable subformula is isolated, obviously the full formula is also unsatisfiable. It may be noted that in random formulas the occurrence of BPR or PPR subformulas is unlikely; in SAT translations of practical problems on the other hand, often structure is present that might be detected in this way. In the next sections we discuss two of the main ingredients of our two-phase algorithm in detail and subsequently we outline the full algorithm.

4.5.1 Finding subformulas with BPR

We address the problem of finding a CNF (sub)formula that is equivalent to a CoE formula. Following the previous section, we search for a subformula with BPR. We can make use of a linear programming (LP) formulation to find a BPR subformula of maximal weight. Since the construction of the LP can be done in polynomial time (assuming that the maximal clause length is bounded and fixed), and LP problems are polynomially solvable [87], the PPR/BPR recognition problem can be solved in polynomial time.

In the formulation the weights w_k occurring in the weighted polynomial representation (WPR) are the main decision variables. Essentially, we want to find a set of non-negative weights w_k and a *slack* $s \geq 0$ such that (see Definition 4.4.8 and equation (4.5)),

$$\sum_{I \subseteq \mathcal{M}} \left| \sum_{k=1}^{n} \left(\prod_{i \in I} a_{ki} \right) w_k \right| + s = \sum_{k=1}^{n} w_k. \tag{4.6}$$

We allow the weights to be equal to zero; if $w_k = 0$ for some k, this implies that clause k is not in the subformula, while if $w_k > 0$ clause k is in the subformula. Our first goal should be to find a solution with s strictly positive (since then the associated subformula has PPR and is unsatisfiable); if no such solution exists, the goal is to identify a subformula of maximal weight with BPR. To check whether solutions with the desired properties exist, we first solve an LP with the objective of maximizing s, and if the optimal value of this LP is equal to zero, a second LP must be solved with the objective to maximize the sum of the weights. Consider the following LP.

$$(\mathrm{LP}_{WPR}) \quad \begin{array}{lll} \max & \alpha s + \beta \sum_{k=1}^{n} w_k & \\ \text{s.t.} & \sum_{I \subseteq \mathcal{M}} (z_I^+ + z_I^-) - \sum_{k=1}^{n} w_k + s = 0, & \\ & \sum_{k=1}^{n} \left(\prod_{i \in I} a_{ki} \right) w_k - z_I^+ + z_I^- = 0, & I \subseteq \mathcal{M}, \\ & 0 \leq w_k \leq 1, & 1 \leq k \leq n, \\ & z_I^+,\ z_I^- \geq 0, & I \subseteq \mathcal{M}, \\ & s \geq 0. & \end{array}$$

The two separate LPs are obtained by setting $\beta = 0$ and $\alpha = 0$, $s = 0$ respectively. The first constraint evaluates expression (4.6) and in the subsequent set of constraints the $c_I(w)$ are computed (see (4.5)). The auxiliary variables z_I^+ and z_I^- associated with the (nonempty) set I are used to eliminate the absolute values in (4.6) in the usual way, i.e. $|c_I(w)| = z_I^+ + z_I^-$, $c_I(w) = z_I^+ - z_I^-$. For a formula in which the clauses have a maximum length ℓ, the numbers of variables and constraints are bounded by $(2^{\ell+1} - 1)n + 1$ and $(2^\ell - 1)n + 1$ respectively. Let us state the properties of (LP_{WPR}) in a lemma.

Lemma 4.5.1 *Assume we are given a formula Φ and its associated linear programming problem (LP_{WPR}). Denote by (s^α, w^α) the optimal solution of (LP_{WPR}) with $\beta = 0$. Similarly, w^β denotes the optimal solution of (LP_{WPR}) with $\alpha = 0, s = 0$.*

- *If $s^\alpha > 0$ then Φ is unsatisfiable. The subformula obtained by extracting the clauses $\mathbf{C}_k$ with positive weight w_k^α has PPR and is therefore unsatisfiable.*

- *If $w^\beta > 0$ then Φ is equivalent to a CoE formula.*

- *If $w^\beta \not> 0$, but $w_k^\beta > 0$ for some clauses $\mathbf{C}_k$, then the subformula consisting of all clauses with positive weight is equivalent to a CoE formula.*

Proof: All claims can be verified by carefully considering (LP_{WPR}). □

If the second LP has a positive optimal value, the CoE subformula can be constructed by taking the conjunction of the equivalency-clauses associated with the sets I for which $|c_I(w)| = z_I^+ + z_I^- \neq 0$, with $\delta_I = \text{sgn}(-c_I(w))$. If the optimal value of the second LP equals zero, no BPR (sub)formula exists. As stated before, this does not necessarily imply that no part of the formula is equivalent to a CoE formula.

Note that a BPR subformula of maximal weight is not guaranteed to be a subformula of maximal *size*. In particular, if a CNF formula contains only clause-disjoint BPR subformulas, the LP approach will identify the maximal size BPR subformula (i.e. the union of the clause-disjoint BPR subformulas). If however some of the subformulas are not clause-disjoint, then the maximal weight BPR subformula does not necessarily coincide with the maximal size subformula. In this respect using an interior point method for solving (LP_{WPR}) might be better than the simplex method, since an IPM yields an optimal solution with a maximal number of nonzero variables.

In practice, heuristics that look for particular structures may often succeed in identifying BPR subformulas. Indeed, for the parity formulas solved in the last section of this chapter the following heuristic suffices. Considering the particular structure of CNF translations of equivalency-clauses (see (4.3)), the strategy to look for 'blocks of clauses' with such a structure gives a large CoE subformula. However, if a subformula is 'well hidden', or does not conform this standard structure, using the LP approach described above will succeed in identifying it, whereas the heuristic methods are likely to fail.

4.5.2 A DPLL algorithm for solving mixed CNF/CoE formulas

One of the best known exact algorithms for solving CNF formulas is the variant of the Davis-Putnam algorithm introduced by Davis, Logemann and Loveland, which is known as the DPLL algorithm; see also Section 3.3. We can easily extend this algorithm to solve conjunctions of CNF and CoE formulas. In figure 4.2 the extension of the algorithm is summarized.

Let us look a bit more closely at the algorithm. First we consider the unit resolution phase. When a unit literal is propagated through the formula, some clauses become *true*, while others reduce in length by one. For equivalency-clauses it holds that each in which the current unit literal occurs simply reduces in length by one. As usual, unit resolution is applied until no unit clauses remain, where it is noted that an equivalency-clause of length one can be regarded as a unit clause in the usual sense. After the unit resolution phase it is checked whether the current formula can be declared satisfiable or contradictory. If not, a *branching* or *splitting* variable l is chosen in some pre-specified way and the DPLL procedure is recursively called with this variable set to *true* and *false* respectively. Note that if a set $\mathcal{I}$ of independent variables is specified, it appears to be sensible to restrict the set of candidate branching variables to $\mathcal{I}$; then the dependent variables are considered in the unit resolution phase only.

```
procedure DPLL ($\Phi = \Phi_{CNF} \cup \Phi_{CoE}$, depth);
    $\Phi$:=unit_resolution($\Phi$);
    if $\Phi = \emptyset$ then
        $\Phi$ is satisfiable: return(satisfiable);
    if $\mathbf{C}_k = \emptyset$ for a $\mathbf{C}_k \in \Phi_{CNF}$ then
        $\Phi$ is contradictory: backtrack;
    if $\mathbf{Q}_k \equiv false$ for a $\mathbf{Q}_k \in \Phi_{CoE}$ then
        $\Phi$ is contradictory: backtrack;
    $l$:=branch_rule($\Phi$);
    DPLL($\Phi \cup \{l\}$, depth+1);
    DPLL($\Phi \cup \{\neg l\}$, depth+1);
return(contradiction);
```

Figure 4.2: The DPLL algorithm extended for CNF/CoE formulas.

Obviously, if in the extended algorithm $\Phi_{CoE} = \emptyset$ (then no equivalency-clauses are present), the algorithm reduces to the usual DPLL algorithm. On the other hand, if $\Phi_{CNF} = \emptyset$, the formula under consideration is a pure CoE. As the DPLL algorithm has exponential complexity, the formula should then rather be solved using the polynomial time special purpose algorithm SOLVE_CoE.

4.5.3 Outline of the two-phase algorithm

We now have all the ingredients to specify the two-phase algorithm. Let Φ be a CNF formula. The two-phase algorithm runs as follows.

1. The LP for detecting PPR or BPR subformulas is constructed and solved. Using the optimal solution, one of the following actions is taken.
    - If a PPR (sub)formula is found, Φ is concluded to be a contradiction.
    - If Φ is fully equivalent to a CoE formula, the appropriate CoE formula is solved using algorithm SOLVE_CoE.
    - If Φ has a (nonempty) BPR subformula Ψ, Ψ is solved/rewritten to the CoE formula Φ_{CoE} using algorithm SOLVE_CoE and $\Phi_{CNF} := \Phi \backslash \Psi$. The algorithm proceeds with phase two.
    - If no BPR structure is detected, the algorithm proceeds with phase two (which in this case reduces to the usual DPLL procedure).
2. The extended DPLL algorithm is applied to decide about satisfiability of the formula $\Phi = \Phi_{CNF} \cup \Phi_{CoE}$.

Note that the first phase of the algorithm runs in polynomial time, while the second phase has an exponential worst case complexity. The first phase may be considered as a *preprocessing method*. Furthermore, note that the CoE formula does not need to be solved separately for the modified DPLL algorithm to be valid. However, if it is solved, and subsequently it turns out that some dependent variable p_j does not occur in the CNF part of the formula, this variable and the equivalency-clause it occurs in need not be considered in the DPLL search procedure, since the value of p_j is uniquely determined by this equivalency-clause. Therefore, during the execution of algorithm SOLVE_COE, if we have the choice to transfer one of two independent variables p_i and p_j from the set of independent variables to the set of dependent variables, it is worthwhile to check whether both appear in the CNF subformula as well. For instance, if only p_i occurs in the CNF subformula, we choose to remove p_j from the set of independent variables. This allows us to reduce the problem size for phase two considerably. This is illustrated in Figure 4.3: the rectangle depicts a CNF formula as a list of clauses. Left, it is discovered that a substantial part of the CNF formula is equivalent to a CoE formula. After first solving this formula, in the second phase only the highlighted part of the formula, depicted in the right hand figure, needs to be considered; the rest of the formula can be uniquely satisfied by extending a satisfying assignment of the highlighted subformula.

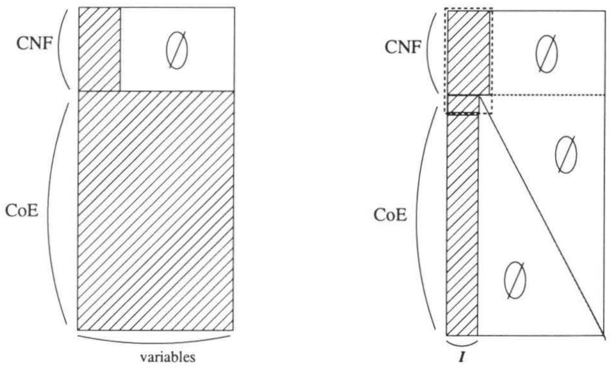


Figure 4.3: Reduction of the search space.

4.6 Application to DIMACS benchmarks

Let us now apply the two-phase algorithm to a number of benchmarks. In the DIMACS suite there are several formulas that are doubly balanced (and thus have a balanced polynomial representation as well) and thus are solved in the first phase of the algorithm, or have BPR subformulas. We start with considering some instances that can be solved in polynomial time by translating them to an equivalent CoE formula first. In the section thereafter some instances with large BPR subformulas are solved in two phases.

4.6.1 Formulas solved in phase one

It turns out that the `dubois*.cnf` and the `pret*_*.cnf` (which are all 3SAT formulas) can be proved a contradiction by Lemma 4.6.1. Also if this lemma is not used, these instances are solved within fractions of seconds by algorithm SOLVE_COE.

Lemma 4.6.1 *Let Ψ be a conjunction of equivalencies. If*

- *each variable occurs an* even *number of times in Ψ, and*
- *the number of negative equivalency-clauses is* odd,

then Ψ is unsatisfiable.

Proof: Consider problem (CoE). A relaxation of this is, to find $x \in \{-1,1\}^m$ that satisfies

$$\prod_{k=1}^{t} \delta_k \prod_{i \in S_k} x_i = 1.$$

If the number of occurrences of each variable is even, this reduces to the product over all δ_k's. If the number of negative equivalency-clauses is odd, a contradiction follows. Consequently, the formula under consideration is not satisfiable. □

In fact these formulas can be considered as special cases of the propositional formulas associated with *Tseitin graphs* [129], which Tseitin used to prove an exponential lower bound on the running time of regular resolution. For branching and other resolution-like algorithms these formulas are very hard, and the number of nodes required in the tree grows exponentially with the size of the formula.

Yet another interpretation of these formulas is, that they are 3SAT translations of formulas of the form

$$([\neg]p_l \leftrightarrow (\ldots([\neg]p_2 \leftrightarrow ([\neg]p_1 \leftrightarrow ([\neg]p_l \leftrightarrow (\ldots([\neg]p_2 \leftrightarrow [\neg]p_1))\ldots).$$

Here the '$[\neg]$' denotes that the negation operator is optional. Such formulas without any negations are known as Urquhart formulas [131]. To obtain `dubois*.cnf`- and `pret*.cnf`-like formulas, an *odd* number of proposition letters must be negated, and subsequently the formula must be translated to 3SAT using auxiliary variables. When the number of negations is odd, the resulting formula is clearly unsatisfiable.

4.6.2 Formulas solved in phase two

In a recent paper by Selman et al. [116] ten challenges in propositional reasoning are formulated. Challenge 2 is stated as follows:

Challenge 2: *(2-5 years) Develop an algorithm that finds a model for the DIMACS 32-bit parity problem.*

These instances arise from the parity learning problem on 32 bits and were generated by Crawford [32]; see also Section 1.3.1. In [116] it is stated that the instances "appear to be too large for current systematic algorithms, while they also defeat the hill-climbing techniques used by current local search algorithms. Given the amount of effort that has been spent on these instances, any algorithm solving them will have to do something significantly different from current methods."

We apply the techniques discussed previously to solve the DIMACS `par*-*-c.cnf` instances. These instances all contain a subformula with balanced polynomial representation. This subformula is a CNF translation of a CoE formula in which all equivalency-clauses have length three. It is not strictly necessary to apply the LP approach to identify this formula, since it can be easily found by inspection. For completeness we list the required time for constructing and solving the LPs in Table 4.2. These tests were run on a HP9000/C200 workstation, 200 MHz. CPLEX was used to solve the LPs, using the barrier algorithm. The BPR subformulas are clause-disjoint, therefore the maximal size BPR subformula is identified by the LP approach. In Table 4.2 are listed, for each instance, the number of clauses n, the number of rows row and columns *col* in the corresponding LP, the time for constructing and solving the LP, and the value of the optimal solution (*opt*). By construction it holds that $n + 2 * row = col$; furthermore, due to the particular structure of the instances (cf. (4.3)), the number of equivalency-clauses t induced by the optimal solution is equal to *opt*/4.

instance	n	row	*col*	time	*opt*
par8-1-c.cnf	254	282	818	.16	224
par8-2-c.cnf	270	301	872	.18	240
par8-3-c.cnf	298	338	974	.19	268
par8-4-c.cnf	266	297	860	.17	236
par8-5-c.cnf	298	335	968	.19	268
par16-1-c.cnf	1264	1537	4338	1.37	1080
par16-2-c.cnf	1392	1692	4776	1.60	1208
par16-3-c.cnf	1332	1619	4570	1.75	1148
par16-4-c.cnf	1292	1567	4426	1.51	1108
par16-5-c.cnf	1360	1653	4666	1.81	1176
par32-1-c.cnf	5254	6524	18302	16.84	4632
par32-2-c.cnf	5206	6466	18138	16.08	4584
par32-3-c.cnf	5294	6574	18442	17.20	4672
par32-4-c.cnf	5326	6618	18562	15.12	4704
par32-5-c.cnf	5350	6648	18646	16.18	4728

Table 4.2: Results of using the LP approach for identifying CoE subformulas

The first and second phase of the algorithm were implemented in C and compiled using `gcc` with the flag `-O2` set. The results reported in Tables 4.3 and 4.4 were obtained running the code on a SGI POWER CHALLENGE with a 200 Mhz R10k processor. All times reported are in seconds. In Table 4.3 we report on the results of the first phase of the algorithm which consists of isolating (by inspection; this requires less than .01 second) and solving the CoE subformulas. The initial numbers of variables and clauses are given by m and n. The number of equivalency-clauses in the CoE subformula is denoted by t; note that indeed $t = opt/4$, while the size of the remaining CNF is $n - opt$ clauses. In the table we also indicate the number of independent variables determining the solutions of the CoE formula. The number of satisfying solutions for the CoE subformula equals $2^{|\mathcal{I}|}$.

instance	m	n	t	time	$\lvert\mathcal{I}\rvert$
par8-1-c.cnf	64	254	56	.01	8
par8-2-c.cnf	68	270	60	.01	8
par8-3-c.cnf	75	298	67	.01	8
par8-4-c.cnf	67	266	59	.01	8
par8-5-c.cnf	75	298	67	.01	8
par16-1-c.cnf	317	1264	270	.04	47
par16-2-c.cnf	349	1392	302	.06	47
par16-3-c.cnf	334	1332	287	.05	47
par16-4-c.cnf	324	1292	277	.06	47
par16-5-c.cnf	341	1360	294	.06	47
par32-1-c.cnf	1315	5254	1158	4.49	157
par32-2-c.cnf	1303	5206	1146	3.80	157
par32-3-c.cnf	1325	5294	1168	4.50	157
par32-4-c.cnf	1333	5326	1176	4.39	157
par32-5-c.cnf	1339	5350	1182	4.62	157

Table 4.3: Results of the first phase of the algorithm

Before starting the second phase of the algorithm, as explained in Section 4.5.3 (with reference to Figure 4.3), first as many dependent variables and corresponding equivalency-clauses as possible are removed. On branching strategies considering only the CNF subformula this has no effect as far as the node count is concerned; computation times however will reduce. The remaining numbers of variables, clauses and equivalency-clauses are given by m, n and t. Note that $m = t + |\mathcal{I}|$; each dependent variable occurs in exactly one equivalency-clause. We tested several branching strategies on the `par16*` instances, and used the one that appeared to be the most effective to solve the larger instances. In Table 4.4 we report on the results. The branching strategy we arrived at is simply the *maximal occurrence in shortest clause* rule, with a lexicographic tie-break, where the candidate branching variables are restricted to the set of independent variables. Note that for determining a branching variable the equivalency-clauses are not considered. We report on the node counts obtained by first branching to l and $\neg l$ respectively. The node count gives the number of times that a branching variable was chosen. A typical phenomenon of DPLL algorithms that we also encountered here is that using different branching strategies the computation times and node counts may vary heavily.

Examining the tables we conclude that the smaller instances are solved in fractions of seconds, while the largest take at most about four minutes. To the best of our knowledge, none of the current state-of-the-art implementations of the DPLL procedure are capable of solving the `par32*` instances in less than 24 hours, and often they require several days of computation time. Recently, it came to our attention that the instances were solved by an unspecified algorithm ('GT6') in two to four hours (Greentech Computing Ltd. [93]).

instance	m	n	t	nodes	time	nodes	time
`par8-1-c.cnf`	31	30	23	3	.00	1	.00
`par8-2-c.cnf`	31	30	23	3	.00	1	.00
`par8-3-c.cnf`	31	30	23	2	.00	1	.00
`par8-4-c.cnf`	31	30	23	3	.00	3	.00
`par8-5-c.cnf`	31	30	23	4	.00	4	.00
`par16-1-c.cnf`	124	184	77	82	.02	67	.02
`par16-2-c.cnf`	124	184	77	58	.01	144	.03
`par16-3-c.cnf`	124	184	77	55	.01	137	.03
`par16-4-c.cnf`	124	184	77	51	.01	131	.03
`par16-5-c.cnf`	124	184	77	49	.01	85	.02
`par32-1-c.cnf`	375	622	218	410634	193	130258	62
`par32-2-c.cnf`	375	622	218	201699	90	335988	160
`par32-3-c.cnf`	375	622	218	502747	248	6712	3
`par32-4-c.cnf`	375	622	218	218021	101	267032	135
`par32-5-c.cnf`	375	622	218	179325	84	328253	164

Table 4.4: Results of the second phase of the algorithm

4.7 Concluding remarks

The two-phase algorithm developed in this chapter provides (at least partial) answers to two of the challenges posed by Selman et al. [116]. Most notably, it efficiently solves the `par32-c-*.cnf` instances. Another challenge posed by Selman et al. is to make integer linear programming practical for satisfiability solving. The first phase of our algorithm partly relies on a linear programming formulation to detect formulas with BPR. With the help of this LP formulation, it is shown to run in polynomial time. Since the first phase is crucial for solving the parity instances, it appears that this is the first truly successful application of linear programming in SAT solving.

As explained before, a similar LP formulation can be used to identify certain contradictions, namely those that have PPR. Unfortunately, it appears that such contradictions do not occur at all in the standard sets of benchmarks. Thus, although pathological instances with PPR can be constructed that are very hard for the usual algorithms, it seems that the notion of PPR is mainly of theoretical interest. When considering random 3CNF formulas for example, the clause-variable ratio required to obtain formulas with PPR is large and increases with the number of variables. Still, the LP formulation does provide a polynomial time algorithm for such unsatisfiable random 3CNF formulas. However, the progress presented by Franco and Swaminathan [48] is likely to yield stronger results. They show that, under the assumption that a polynomial time approximation algorithm for the Hitting Set problem with sufficient guarantee is available (and they argue that the existence of such an algorithm is likely), random 3CNF formulas with a large, but *constant* clause-variable ratio can be verified to be contradictory in polynomial time.

5

Semidefinite Relaxations Of Satisfiability Problems

Motivated by recent theoretical results on approximating combinatorial optimization problems, we examine the use of semidefinite programming in SAT solving. Using the well-known Goemans-Williamson algorithm, and exploiting problem sparsity, a competitive algorithm for (small-sized) MAX2SAT problems is obtained. Subsequently, we derive a semidefinite relaxation of general satisfiability problems, and discuss its strength to decide unsatisfiability. Using this relaxation, a certificate of unsatisfiability of the notorious pigeon hole and mutilated chess board problems can be obtained in polynomial time. In addition, the relaxation yields a polynomial time algorithm for 2SAT and by slightly enhancing it polynomially solvable 3SAT problems can be identified. In general, the relaxation can be used in a branching algorithm to reduce the size of the search tree.

5.1 Introduction

In the early nineties, Goemans and Williamson introduced a new approximation algorithm for MAXCUT and MAX2SAT using semidefinite programming (SDP) [56]. Since then much attention has been devoted to this field. Most of the research thus far focused on developing polynomial time algorithms for SDP problems, and on developing approximation algorithms for various combinatorial optimization problems. For example, semidefinite programming (SDP) relaxations – in conjunction with randomized rounding schemes – yield 7/8 and 0.931 approximation algorithms for MAX3SAT and MAX2SAT respectively [86, 46]. In spite of these powerful theoretical results, it is not clear whether SDP can be used as a practical tool for solving MAX-SAT problems to optimality, or for solving general SAT problems. In this regard the usefulness of the SDP approach depends on two important issues:

1. The tightness of the SDP relaxation.
2. The efficiency with which SDP relaxations can be solved.

In this chapter we address both of these issues. Due to the relatively short history of the area it is early to make definitive statements, hence some of the observations made are of a preliminary nature; yet, they show promise for the future.

We first consider the Goemans-Williamson relaxation of MAX2SAT problems [56]. For any instance of SAT a MAX2SAT formula can be constructed whose optimal solution gives a certificate of (un)satisfiability of the original SAT instance. Based on the elliptic approximation introduced in Chapter 2, a quadratic model for MAX2SAT is derived, whose 'standard' semidefinite relaxation is identical to the Goemans-Williamson SDP relaxation. Using this relaxation, a 0.879 approximation algorithm can be obtained. This bound was improved to 0.931 by Feige and Goemans [46]. We are interested in applying the SDP approach to obtain exact MAX2SAT solutions. Recent numerical studies indicate that it is very hard to prove optimality for MAX2SAT by only tightening the SDP relaxation [81]. Therefore it seems necessary to incorporate the relaxation in some branch and cut framework. Until recently, the bottleneck for such an approach has been the computational cost of solving the SDP relaxations: it was unclear how to exploit sparsity in the SDP relaxations. In a recent breakthrough, Benson, Ye and Zhang [9] proposed to solve the *dual* of the SDP relaxation in order to exploit sparsity. They applied their method to obtain approximately optimal solutions for MAXCUT and other graph partition problems and reported a promising computational efficiency. We test a branch and cut procedure to solve MAX2SAT to optimality, in each node solving the dual of the SDP relaxation using Benson's implementation. The results of this approach are compared to the results of two other complete algorithms for MAX2SAT, namely an extended version of the DPLL algorithm (see also Section 3.3) and the LP based general purpose branch and cut solver MINTO [14, 103]. We show that using the SDP approach in a branch and cut scheme, instances of MAX2SAT with 50 variables and up to thousands of clauses are solved to optimality in a few minutes on a workstation, whereas the other methods tend to fail as problem sizes increase. Unfortunately, using our current implementation the size of problems that can be solved to optimality is not sufficient to attempt solving general SAT instances as MAX2SAT formulas, due to the increase in problem size when reducing SAT to MAX2SAT.

Instead, we develop an SDP relaxation for general SAT problems whose size is more acceptable. Similar to the MAX2SAT relaxation, this relaxation is inspired by elliptic approximations of propositional formulas as well. Here they are used to characterize a sufficient condition of the unsatisfiability of a formula. This condition can be expressed in terms of an *eigenvalue optimization* problem, which in turn can be cast as a semidefinite program (see e.g. [38, 134]). (The MAX2SAT SDP relaxation also can be interpreted as an eigenvalue optimization problem.) Using duality theory, we show that the dual of our formulation is a semidefinite feasibility problem, which is closely related to the formulation developed by Karloff and Zwick for MAX3SAT formulas [86]. However, rather than for finding approximate MAX-SAT solutions, our principal aim is to use it for proving unsatisfiability. We show that the relaxation provides a polynomial-time certificate of unsatisfiability of the notorious pigeon hole problems in a truly automated way; i.e. without additional problem-specific tricks. As a byproduct, we obtain a new 'sandwich theorem' that is similar to Lovász' famous ϑ-function [94]. Furthermore, we show that the relaxation is exact for 2SAT formulas and we indicate how it can be used to help solving 3SAT problems. In particular, a certain class of polynomially solvable 3SAT formulas can be recognized by adding an objective function to the dual formulation (namely doubly balanced 3SAT formulas; see Chapter 4).

This chapter is organized as follows. In the next section we discuss the preliminaries and notation, and give a brief introduction to semidefinite programming. In Section 5.3 the Goemans-Williamson relaxation is derived, and a technique to tighten it is discussed. Subsequently, Section 5.4 is concerned with obtaining exact solutions for MAX2SAT instances. In Section 5.5, we derive a semidefinite relaxation of the satisfiability problem, give its dual formulation and mention a number of properties. The strength of the relaxation is investigated in Section 5.6, by considering several subclasses of satisfiability problems, namely 2SAT problems, a class of covering problems (to which the pigeon hole and mutilated chess board problem belong) and 3SAT problems. We conclude with some empirical observations and further remarks.

5.2 Preliminaries and notation

5.2.1 SAT and MAX2SAT

As usual, we consider the satisfiability problem in conjunctive normal form (CNF); see Chapter 1. Associating a $\{-1,1\}$-variable x_i with each proposition letter p_i, clauses can be written as a linear inequalities. Using the clause-variable matrix A (see (2.1)), the integer linear programming formulation of the satisfiability problem can be stated as

$$(\text{IP}_{SAT}) \quad \text{find } x \in \{-1,1\}^m \text{ such that } Ax \geq b.$$

In an instance of MAX2SAT, all clauses have length less than or equal to two. Checking whether an assignment exists that satisfies all clauses can be done in linear time [6]; in general however, the MAX2SAT problem is NP-complete [52]. By complexity theory, this implies that any SAT problem can be solved as a MAX2SAT problem. For example, the most concise MAX2SAT representation of a 3-clause $p \vee q \vee r$ is given by the following set of (weighted) 2-clauses (Trevisan et al. [127]):

$$p \vee s, \; p \vee s, \; \neg q \vee s, \; q \vee \neg s, \; \neg r \vee s, \; r \vee \neg s, \; q \vee r, \; \neg q \vee \neg r,$$

where s is an auxiliary variable. Note that if the original clause is not satisfied (i.e p, q and r are all $false$), then five out of eight clauses are satisfied; otherwise seven clauses can be satisfied (by giving s its appropriate truth value). Therefore, if the 3SAT instance has n clauses and m variables, the associated MAX2SAT instance has $7n$ weighted clauses and $m + n$ variables. The original instance is satisfiable if and only if the optimal value of the weighted MAX2SAT instance equals $7n$.

5.2.2 An introduction to semidefinite programming

Recently much attention has been devoted to the field of semidefinite programming. It was shown that efficient approximation algorithms for hard combinatorial optimization problems can be obtained using semidefinite relaxations (Goemans-Williamson [56], Alizadeh [1]), while there are also applications in control theory (Vandenberghe and Boyd [134]). Using interior point methods, semidefinite programs can be solved (to a given accuracy) in polynomial time. For the reader that is unfamiliar with semidefinite programming, we review some of the basic concepts.

The standard primal (SP) and dual (SD) semidefinite programming formulations can be denoted as (see e.g. de Klerk [38])

$$
\text{(SP)}\quad \begin{array}{ll} \inf & \mathbf{Tr}\, CX \\ \text{s.t.} & \mathbf{Tr}\, A_i X = b_i, \quad 1 \le i \le n, \\ & X \succeq 0. \end{array}
\qquad
\text{(SD)}\quad \begin{array}{ll} \sup & b^T y \\ \text{s.t.} & \sum_{i=1}^{n} y_i A_i + Z = C, \\ & Z \succeq 0. \end{array}
$$

In the above programs, the A_i, C, X and Z are symmetric real $(m \times m)$-matrices and b and y are n-vectors. The matrix X denotes the primal decision variables, while (Z, y) are the dual decision variables; the constraint $X \succeq 0$ (resp. $Z \succeq 0$) indicates that X (resp. Z) must be positive semidefinite. Positive semidefiniteness of a matrix can be characterized in several ways (see e.g. Strang [123]). A symmetric real matrix $A \in \mathbb{R}^{m\times m}$ is said to be positive semidefinite if *(i)* $x^T A x \ge 0$ for all $x \in \mathbb{R}^m$, *(ii)* all the eigenvalues of A are nonnegative, *(iii)* there exists a matrix R such that $A = R^T R$. Furthermore, **Tr** denotes the *trace*-operator. The trace of a matrix A is equal to the sum of its diagonal elements. A useful easy-to-check property of the trace operator is $\mathbf{Tr}\ AB = \mathbf{Tr}\ BA$, where A and B are matrices of appropriate sizes. Also, the trace of a matrix is equal to the sum of its eigenvalues.

When all the data matrices involved in the pair (SP, SD) are diagonal matrices, the semidefinite programming problem reduces to a linear programming problem. Note that the (nonlinear) constraints $X \succeq 0$, $Z \succeq 0$ then reduce to nonnegativity constraints.

The duality theory for semidefinite programming is similar to – but slightly weaker than – the duality theory of linear programming. For the pair (SP, SD) weak duality holds, i.e. $b^T y \le \mathbf{Tr}\ CX$ if X is feasible for (SP) and y is feasible for (SD). As in linear programming, we call the nonnegative quantity $\mathbf{Tr}\ CX - b^T y$ the *duality gap*. It is easy to show that the duality gap equals $\mathbf{Tr}\ XZ$ for feasible X, Z.

We say that (SP, SD) are in perfect duality if their optimal values coincide, where we adopt the convention that $\mathbf{Tr}\ CX = \infty$ if (SP) is infeasible, $b^T y = -\infty$ if (SP) is unbounded, etc. Perfect duality is guaranteed if one of (SP) and (SD) is *strictly feasible*; (SP) (resp. (SD)) is strictly feasible if a strictly interior solution $X \succ 0$ (resp. $Z \succ 0$) exists. Infeasibility of one then implies unboundedness of the other. Note that perfect duality always holds in linear programming; in the semidefinite programming case pathological duality effects can occur when, for example, (SP) is infeasible but (SD) has a finite optimal value.

If *both* (SP) and (SD) are strictly feasible, then optimal solutions (X, Z) exist for (SP, SD) with duality gap zero (i.e. $\mathbf{Tr}\ XZ = 0$). Such solutions are called *complementary*, since $\mathbf{Tr}\ XZ = 0$ is equivalent to $XZ = 0$ for positive semidefinite matrices. In linear programming, all solutions to the primal and dual problems are complementary, but for semidefinite programming (SP) and (SD) may have optimal solutions with positive duality gap, if strict feasibility does not hold. We speak of *strict infeasibility* of (SP) if there exists an *improving direction* for (SD), and vice versa. If an improving direction of (SD) exists, then its objective function can be increased indefinitely, which means that (SD) must be unbounded if it is feasible. Improving directions for (SP) are defined similarly. Once again, strict infeasibility is the only kind of infeasibility which can occur in linear programming, but in semidefinite programming *weak infeasibility* is also possible. The semidefinite programs we consider in this chapter cannot give rise to these pathological duality effects, and the duality relations used here will be no more complicated than in the linear programming case. For more details on semidefinite programming duality issues we refer to [38].

5.3 Goemans & Williamson's MAX2SAT algorithm

In this section a quadratic model of the MAX2SAT problem is derived and relaxed to a semidefinite program. The quality of the relaxation and of the solutions obtained using the Goemans-Williamson algorithm is considered. Finally, we discuss techniques to tighten the relaxation.

5.3.1 A quadratic model of the MAX2SAT problem

We derive a quadratic model for the MAX2SAT problem. This model gives rise to the same semidefinite relaxation as the quadratic models given by Goemans and Williamson [56] and Delorme and Poljak [39]. However, our model follows very naturally from the linear model for 2SAT and allows recognition of a linear autarky (see Section 2.2) via the eigenvalues of the matrix involved. This may yield a reduction of problem size.

Specifically applying the linear inequalities associated with clauses to clauses of length two, it is clear that a 2-clause $(\neg)p_i \vee (\neg)p_j$ is satisfied if and only if it holds that $(-)x_i + (-)x_j \geq 0$. Similarly, a one-literal clause $(\neg)p_i$ is satisfied if and only if $(-)x_i \geq 0$, or equivalently $(-)x_i = 1$. Such a clause is modelled as $(-)2x_i \geq 0$, which obviously is equivalent. Let A be the clause-variable matrix associated with a mixed 1,2CNF formula. Making use of the elliptic representation of 2SAT formulas as introduced in Chapter 2, we obtain the following result. This lemma is closely related to Lemma 2.4.3.

Lemma 5.3.1 *Let Φ be a 2CNF formula with associated clause-variable matrix A. Let $x \in \{-1,1\}^m$ be an assignment. The number of clauses that is* not *satisfied by x is equal to $\frac{1}{8}(x^T A^T A x - 2e^T A x)$.*

Proof: Consider a single element of the vector Ax. By construction and the fact that $x \in \{-1,1\}^m$ this element is equal to -2, 0 or 2. In the first case the corresponding clause is not satisfied, in the other two cases it is satisfied. Note that the value '0' cannot occur in the case of one-literal clauses. Now suppose that a vector x is such that it does not satisfy k clauses. It is easily verified that then the following holds:

$$(Ax - e)^T(Ax - e) = (n - k) + 9k.$$

Expanding the product (noting that $e^T e = n$) and rearranging the terms gives the desired result. □

Thus we arrive at the following quadratic formulation.

$$\text{(M2S)} \quad \begin{array}{ll} \min & \frac{1}{8}(x^T A^T A x - 2e^T A x) \\ \text{s.t.} & x \in \{-1,1\}^m. \end{array}$$

Denoting the optimal value to (M2S) by *opt*(M2S), from Lemma 5.3.1 we conclude that the maximum number of satisfied clauses is given by $n -$ *opt*(M2S). Note that (M2S) also models *weighted* MAX2SAT instances, since it is obviously also allows multiple copies of a clause in the matrix A.

We have the following decomposition theorem.

Theorem 5.3.2 *If the matrix A^TA has an eigenvalue zero, then the corresponding eigenvector s is a linear autarky.*

Proof: This is a special case of Theorem 3.4.1. □

We have a straightforward corollary from this theorem.

Corollary 5.3.3 *If a 2SAT formula has more variables than clauses (i.e. $m > n$), it is satisfiable or decomposable.*

This implies that any 2SAT formula with $m > n$ that is not directly seen to be satisfiable, can be solved by considering an appropriate subproblem, in accordance with Theorem 5.3.2.

Note that $(A^TA)_{ij}$ can be nonzero only if p_i and p_j appear together in some clause. Thus the fraction of nonzeros of A^TA can never exceed the ratio $(2n + m) : m^2$. For example, for a MAX2SAT instance with $m = 100$ variables and $n = 400$ clauses the upper bound on the density of A^TA is 9%. We will show in later sections why this ratio is an important consideration when choosing an algorithm for solving the SDP relaxation.

We conclude this section by mentioning a property of *pure* MAX2SAT problems (i.e. only clauses with length two are present) that can be easily derived from the quadratic model. If the matrix A^TA is diagonal and the linear term $A^Te \equiv 0$ (this implies that the objective function contains purely quadratic terms only), the maximal number of clauses that can be satisfied is equal to

$$n - \frac{1}{8}(x^TA^TAx - 2e^TAx) = n - \frac{1}{8}\sum_{i=1}^{m}(A^TA)_{ii} = n - \frac{1}{8}\sum_{k=1}^{n}\ell(\mathbf{C}_k) = n - \frac{1}{4}n = \frac{3}{4}n.$$

(here we used Lemma 2.4.5); in this case *any* assignment is optimal, which follows directly from the quadratic model.

5.3.2 The semidefinite relaxation

Let us now derive a semidefinite relaxation of (M2S). To this end, we first consider the following general homogeneous quadratic optimization problem over the unit hyper cube.

$$\text{(QP)} \quad \begin{array}{ll} \min & x^TQx \\ \text{s.t.} & x \in \{-1,1\}^m. \end{array}$$

A lower bound for (QP) can be obtained by considering the spectral decomposition of Q, i.e. $Q = S\Lambda S^T$. Here Λ is the diagonal matrix containing the eigenvalues of Q (see also Section 3.2.2). Thus,

$$x^TQx = x^TS\Lambda S^Tx = y^T\Lambda y = \sum_{i=1}^{m}\lambda_i y_i^2,$$

where $y = S^T x$. Note that $y^T y = x^T x = m$. We assume that $\lambda_{\min} \equiv \lambda_1 \leq \ldots \leq \lambda_m \equiv \lambda_{\max}$. It is easy to see that $x^T Q x$ attains its minimum over the sphere $x^T x = m$ in $x = \sqrt{m} s_{\min}$ (where $s_{\min}$ denotes an eigenvector associated with $\lambda_{\min}$), and the corresponding minimal value is $\lambda_{\min} m$. Since $x^T x = m$ for any $x \in \{-1, 1\}^m$ this gives a lower bound of (QP).

Considering (M2S), note that if the linear term $A^T e \not\equiv 0$ then (M2S) is not homogeneous quadratic and thus the eigenvalue bound is not immediately available. By introducing one auxiliary $\{-1, 1\}$ variable, a problem with $A^T e \neq 0$ can be made homogeneous quadratic as follows.

$$\begin{bmatrix} x \\ x_{m+1} \end{bmatrix}^T \begin{bmatrix} A^T A & -A^T e \\ -e^T A & 0 \end{bmatrix} \begin{bmatrix} x \\ x_{m+1} \end{bmatrix} = x^T A^T A x - 2 x_{m+1} e^T A x.$$

If $(x; x_{m+1})$ minimizes this quadratic form, then $x_{m+1} x$ minimizes its non-homogenized form.

Thus (M2S) can be cast as a homogeneous quadratic optimization problem such as (QP), where $Q \in \mathbb{R}^{(m+1)\times(m+1)}$ is given by

$$Q = \frac{1}{8} \begin{bmatrix} A^T A & -A^T e \\ -e^T A & 0 \end{bmatrix}.$$

The optimal MAX2SAT solution can still be computed via $n - \mathit{opt}(\text{M2S})$.

Let us now recall an observation earlier made in Section 2.4.4. Using the property that $x_i^2 = 1$ other equivalent formulations can be obtained: adding some vector u to the diagonal and subsequently updating the objective value by subtracting $e^T u$ leaves the problem essentially unchanged, since

$$x^T (Q + \operatorname{diag}(u)) x = x^T Q x + e^T u$$

for any $\{-1, 1\}$ vector x. The vector u is called a *correcting vector* [39]. To obtain a tight lower bound, the minimal eigenvalue of $Q + \operatorname{diag}(u)$ can be maximized over all correcting vectors u. Thus we obtain an *eigenvalue optimization* problem, which can be expressed as a semidefinite program as follows.

$$(\text{SD}_{M2S}) \quad \begin{array}{rl} \sup & (m+1)\lambda \\ \text{s.t.} & (Q + \operatorname{diag}(u)) \succeq \lambda I, \\ & e^T u = 0. \end{array}$$

To verify the validity of this formulation, note that if μ is an eigenvalue of Q, then $\mu - \lambda$ is an eigenvalue of the matrix $Q - \lambda I$. The associated dual formulation can be simplified to (using the standard primal-dual pair (SP, SD))

$$(\text{SP}_{M2S}) \quad \begin{array}{rl} \inf & \mathbf{Tr}\, QY \\ \text{s.t.} & \operatorname{diag}(Y) = e, \\ & Y \succeq 0. \end{array}$$

It is easy to see that both (SD_{M2S}) and (SP_{M2S}) are strictly feasible; by duality theory it follows that their optimal values exist and are equal (see Section 5.2.2). As a consequence, the 'sup and 'inf' can be replaced by 'max' and 'min'. The optimal solutions can be computed (to a given accuracy) in polynomial time.

An alternative derivation of the SDP relaxation of (M2S) is the following. Rewrite

$$x^TQx = \mathbf{Tr}\ x^TQx = \mathbf{Tr}\ Qxx^T = \mathbf{Tr}\ QY,$$

where $Y = xx^T$ is a rank one matrix with ones on its diagonal (since $x \in \{-1,1\}^m$). Subsequently the rank one condition is relaxed to the condition that $Y \succeq 0$, i.e. Y must be positive semidefinite. The latter is obviously a necessary condition since $a^T(xx^T)a = (a^Tx)^2 \geq 0$ for any vector a.

Note that in the relaxation all products x_ix_j are replaced by matrix entries Y_{ij}. Given a Choleski decomposition of Y, say $Y = V^TV$, one can write $Y_{ij} = (v_i)^Tv_j$ where the v_i's are the columns of V. This means that the product x_ix_j is in fact relaxed to a inner product $(v_i)^Tv_j$. This type of relaxation was originally suggested by Lovász and Schrijver [95].

5.3.3 The quality of the SDP relaxation

Goemans and Williamson [56] proved that

$$\frac{n - opt(\mathrm{M2S})}{n - opt(\mathrm{SP}_{M2S})} \geq 0.87856. \tag{5.1}$$

As stated before, if all distinct clauses on m variables are included, then exactly $\frac{3}{4}$ of the clauses are satisfiable, i.e. $opt(\mathrm{M2S}) = \frac{n}{4}$. The SDP relaxation is then exact, since

$$\mathbf{Tr}\ QY = \sum_{i=1}^{m} Q_{ii}Y_{ii} = \frac{1}{8}\sum_{i=1}^{m}(A^TA)_{ii} = \frac{n}{4}.$$

It is therefore reasonable to expect that the SDP relaxation will become even tighter than guaranteed by (5.1) as the ratio n/m grows. This can be observed from numerical experiments shown in Table 5.1, where the average ratio for the left hand side of expression (5.1) is given as a function of the number of clauses (for random MAX2SAT instances with 50 variables).

Goemans and Williamson proposed the following randomized heuristic for use in conjunction with the SDP relaxation:

- Solve the SDP relaxation (SP_{M2S}) to obtain an ε-optimal $Y = V^TV$.
- Choose a random vector $s \in \mathbb{R}^{m+1}$ and normalize s.
- Set $x_i = 1$ if $s^Tv_i \geq 0$ or set $x_i = -1$ otherwise.

This randomized algorithm yields an approximately optimal solution to MAX2SAT with expected objective value at least 0.87856 times the optimal value.

# clauses	$\frac{n\text{-}opt(\mathrm{M2S})}{n\text{-}opt(\mathrm{SP}_{M2S})}$
50	0.95658
100	0.97502
200	0.98538
500	0.98911
1000	0.99128
2000	0.99258
3000	0.99400
4000	0.99503
5000	0.99592

Table 5.1: The average quality of the SDP relaxation improves as the number of clauses grows (50 variables).

5.3.4 Tightening the relaxation

Feige and Goemans [46] have shown that there exist instances of MAX2SAT where the ratio (5.1) is no better than 0.88889. They propose to strengthen the SDP relaxation (SP_{M2S}) by adding a number of valid inequalities (cuts). The proposed cuts (called *triangle inequalities*) are based on the following observation. Let e_{ijk} be a vector whose entries are zero, except for the entries i, j and k, each of which is equal to either $+1$ or -1. Then it is easy to see that for any $\{-1,1\}$ vector x, $e_{ijk}^T x$ is odd. In particular, this implies that $(e_{ijk}^T x)^2 \geq 1$, from which it follows that

$$1 \leq (e_{ijk}^T x)^2 = (x^T e_{ijk})(e_{ijk}^T x) = \mathbf{Tr}\ (e_{ijk} e_{ijk}^T) x x^T.$$

Feige and Goemans have shown that adding a subset of such inequalities in conjunction with a modified rounding scheme, improves the quality guarantee of the SDP relaxation from 0.87856 to 0.93109 (cf. (5.1)). A bound on the worst-case approximation is 0.94513, i.e. there exist problems where the ratio (5.1) is no larger than 0.94513. To obtain this approximation result the following inequalities are added. For all (i,j),

$$\begin{aligned} x_{m+1}x_i + x_{m+1}x_j + x_i x_j &\geq -1 \\ -x_{m+1}x_i - x_{m+1}x_j + x_i x_j &\geq -1 \\ -x_{m+1}x_i + x_{m+1}x_j - x_i x_j &\geq -1. \end{aligned}$$

In the SDP relaxation these inequalities correspond to $\frac{3}{2}m(m-1)$ additional linear constraints of the form

$$\mathbf{Tr}\ (A_i Y) \geq 1,$$

where the A_i are rank one matrices. If all distinct $\frac{2}{3}m(m^2-1)$ triangle inequalities are added, the quality guarantee remains 0.93109, but the worst-known behaviour now becomes 0.98462. In practice all these inequalities cannot be added beforehand because of the increase in problem size; it is more feasible to re-solve the SDP relaxation after having added (some of) the violated inequalities. Recently, Halperin and Zwick [64] reported that using *outward rotations*, an algorithm with the same performance guarantee can be obtained. Unfortunately, no details are provided.

5.4 Computing exact MAX2SAT solutions via SDP

So far we have derived a semidefinite relaxation for MAX2SAT problems and considered its theoretical performance. Let us now try to use the relaxation to obtain exact (i.e. proven to be optimal) solutions.

5.4.1 Solving the SDP relaxation of MAX2SAT

The SDP relaxations mentioned so far can be cast in the generic form (SP, SD), where the A_i's include the rank one matrices corresponding to the valid inequalities of Section 5.3.4. Let us consider the dual problem (SD_{M2S}) in some detail, with cuts included. It can be rewritten as follows (here $z = \lambda e + u$ and the equality constraint $e^T u = 0$ is eliminated).

$$(\mathrm{SD}_{M2S}) \qquad \begin{array}{ll} \max & e^T(y+z) \\ \text{s.t.} & \operatorname{diag}(z) + \sum_{i=1}^{t} y_i A_i + Z = Q, \\ & y \geq 0, Z \succeq 0. \end{array}$$

Note that the dual matrix Z will have more or less the same sparsity structure as Q, if the number of cuts t is small. Recall further that Q will be sparse in general, as discussed in Section 5.3.1, while the primal decision variable matrix Y will be dense in general. (Note that due to the homogenization, the upper bound on the density of Q is slightly higher than that of $A^T A$, namely $(2n + 3m) : (m + 1)^2$. For $n = 400$, $m = 100$ this is less than 10.8%.) This suggests to solve the dual problem instead of the primal in order to exploit this sparsity structure.

Dual interior point methods are based on the dual logarithmic barrier function

$$f_d(Z, y) = \log\det(Z) + \sum_{i=1}^{t} \log(y_i),$$

which can be added the dual objective function in order to replace the (matrix) inequality constraints $Z \succeq 0$ and $y \geq 0$. Thus one can solve a sequence of problems of the form

$$(\mathrm{SD}^*_{M2S}) \qquad \begin{array}{ll} \max & e^T(y+z) + \mu f_d(Z, y) \\ \text{s.t.} & \operatorname{diag}(z) + \sum_{i=1}^{t} y_i A_i + Z = Q, \end{array}$$

for decreasing values of $\mu > 0$. The projected Newton direction for this problem can be calculated from a positive definite linear system with coefficient matrix consisting of four blocks (see the collected works [3, 9, 38, 45, 69]):

$$L := \begin{bmatrix} Z^{-1} \circ Z^{-1} & B \\ B^T & C \end{bmatrix}$$

where '$\circ$' indicates the Hadamard (component-wise) product, and the entries (i, j) of the blocks B and C are respectively of the form

$$b_{ij} = \mathbf{Tr}\left(A_i Z^{-1} e_j e_j^T Z^{-1}\right) = e_j^T Z^{-1} A_i Z^{-1} e_j$$

where e_j is the jth standard unit vector, and

$$c_{ij} = \mathbf{Tr}\ \left(A_i Z^{-1} A_j Z^{-1}\right).$$

Once Z^{-1} is known, the computation of $[Z^{-1} \circ Z^{-1}]_{ij}$, b_{ij} and c_{ij} all require only one multiplication and some additions.

Benson, Ye and Zhang propose a method to quickly assemble the matrix L when all constraint matrices A_i have rank one. This method does not require explicitly computing and storing Z^{-1}. The complexity of each iteration is shown to be $\mathcal{O}(m^3 + n^2m + nm^2 + n^3)$, where n denotes the number of constraints (which is here equal to $m+t$). The computation per iteration is dominated by the solution of the linear system with coefficient matrix L. For comparison, Benson et al. note that the most efficiently computable primal-dual search direction (which is due to Nesterov and Todd [105]) requires $\mathcal{O}(n^3m+n^2m^2+\max\{n,m\}^3)$ operations. Benson implemented a dual scaling method (using the search direction described above) which requires $O(\sqrt{n+m})$ iterations for convergence; for details the reader is referred to [9]. This implementation is used in the numerical experiments below. The primal variable Y can be computed as a byproduct as necessary. It may be noted that currently only the constraints on the diagonal entries of Y are included in the implementation; additional rank-one constraints are not yet supported.

To give an impression of the speed with which the relaxed problem can be solved using this approach, the average CPU-times (in seconds) for MAX2SAT relaxations of the (randomly generated) benchmark problems from Joy et al. [80] are given in Table 5.2. The computation was done on a HP 9000/715 workstation.

	ratio		time SDP		time heuristic	
# clauses	mean	std	mean	std	mean	std
180	0.9657	0.0050	3.40	0.39	0.27	0.02
200	0.9698	0.0038	3.53	0.22	0.28	0.02
220	0.9717	0.0030	3.74	0.39	0.29	0.01
240	0.9729	0.0046	3.79	0.35	0.30	0.02
260	0.9752	0.0049	3.95	0.30	0.32	0.01
280	0.9724	0.0041	4.24	0.36	0.34	0.01
300	0.9750	0.0037	4.28	0.34	0.35	0.01
400	0.9780	0.0024	4.83	0.34	0.40	0.01

Table 5.2: Average solution times and approximation ratio for the SDP relaxation of MAX2SAT and for the for the Goemans-Williamson heuristic on 100-variable problems.

The column 'ratio' indicates the average ratio of the best obtained heuristic solution to $n - opt(\mathrm{SP}_{M2S})$. The ratio is consistently higher than 0.965 with low standard deviations (both in time and quality), indicating that near-optimal solutions are produced. It appears that the actual performance of the algorithm is much better than the expected worst case. Still, the solutions are not yet proven to be optimal. In general, computational experiences suggest that finding *good* solutions of MAX2SAT problems is not too difficult (note though

```
procedure node_procedure ($\Phi, lb, unsat$);
    if ($ub - unsat = 1$) unit_resolution($\Phi$);
    ($lb_{sdp}$, $ub_{sdp}$):= SDP_relaxation($\Phi$);                (*)
    $ub := \min\{ub,\ unsat + ub_{sdp}\}$;                            (*)
    $lb := \max\{lb,\ unsat + \max\{0, lb_{sdp}\}\}$;                 (*)
    if ($ub - lb < 1$) return;
    $p$ :=branch_rule($\Phi$);
    Set $p \equiv true$ and update $\Phi$, $unsat$;
    if ($ub - unsat \geq 1$) node_procedure ($\Phi, lb, unsat$);
    Set $p \equiv false$ and update $\Phi$, $unsat$;
    if ($ub - unsat \geq 1$) node_procedure ($\Phi, lb, unsat$);
return;
```

Figure 5.1: Branch and cut framework for MAX2SAT

that not all procedures provide a bound to measure the quality of the solution). The real computational challenge lies in proving *optimality* [80]. This is the subject of the next section.

5.4.2 A branch and cut framework

The SDP relaxations can be used in a branch and cut framework. The framework we have used for our numerical experiments is described in this section, with reference to Figure 5.1. This algorithm is a straightforward generalization of the DPLL algorithm (Section 3.3) suited for MAX-SAT problems.

At any node in the branching tree, the current set of clauses (obtained after partial assignment of the variables) is denoted by Φ, and lb and ub contain lower and upper bounds on the minimal number of unsatisfiable clauses respectively. The value $unsat$ is a counter for the number of unsatisfied clauses by the current partial assignment. Note that lb and $unsat$ are *local* variables that are valid in the current branch only; on the other hand, ub is a global variable which is valid for the whole search tree. At termination of the procedure, ub contains the optimal value of the instance.

Before calling `node_procedure` the values lb, $unsat$, and ub must be initialized. One can take $lb := 0$, $unsat := 0$, $ub := n$. Following [13], unit resolution is applied if $ub - unsat = 1$. Subsequently, the semidefinite relaxation of the current formula is solved to obtain upper and lower bounds ub_{sdp} and lb_{sdp}. The current bounds ub and lb are then updated (taking $unsat$ into account). If $ub - lb < 1$, then the best known solution hitherto cannot be improved upon in the current branch, hence the algorithm backtracks. Otherwise a variable x is determined to branch on, which is set to $true$ and $false$ respectively. Since the presence of unit clauses improves the tightness of the bounds computed (see Trevisan et al. [127]), the branching rule for fixing variables is as follows:

Branching Rule 5.1 *Choose the variable with the* maximal occurrence in the longest clauses.

The formula Φ and *unsat* are subsequently updated; if $ub - unsat < 1$ this branch need not be further explored. In the other case `node_procedure` is recursively called.

In each node, the steps marked ($*$) can be repeated adding violated cuts as discussed in Section 5.3.4 to the relaxation, to obtain tighter bounds.

5.4.3 Numerical experiments

Let us now present some numerical results for the branch and cut SDP algorithm of the previous section. The results presented here are of a preliminary nature, and were obtained *without* adding extra cuts. The MAX2SAT benchmark problems are taken from Borchers and Furman [13]. As before, all reported CPU-times are in seconds on a HP-9000/715 work station with 160MB internal memory. The SDP branch and cut method presented here are compared to

- an extended DPLL algorithm, EDPLL, (see also Borchers and Furman [13]). This is basically the DPLL algorithm (Section 3.3), modified to accommodate MAX-SAT solving (resulting in the algorithm explained in the previous section, obviously without the SDP relaxation included);
- a mixed integer linear programming approach using the commercial solver Minto [103].

The respective CPU-times are shown in Table 5.3.

	SDP		EDPLL	Minto
# clauses	time	nodes	time	time
100	84	82	1.36	12.9
150	69	64	5.1	18.0
200	91	70	395	67.3
250	118	92	2218	128
300	170	128	29794	687
350	127	91	>12hr	2339
400	56	40	>12hr	1550
450	276	210	>12hr	12634
500	205	144	>12hr	8677
2500	331	184	not run	not run
5000	663	399	not run	not run

Table 5.3: Solution times (in seconds) of MAX2SAT benchmark problems ($m = 50$) for different algorithms

It is immediately clear that the SDP approach is distinctly superior to the other two approaches if the clauses/variables ratio exceeds 5. The reason seems to be that the SDP relaxation becomes tighter as this ratio grows, as discussed in Section 5.3.3. Note also that the SDP branch and cut algorithm solved each of the problems in a few minutes. It therefore has a very robust performance in comparison to the other two methods. On the other hand, even though the approximate solutions in the root node are optimal or close to optimal, it appears that actually proving optimality requires a fair number of branches. Thus, even when an optimal solution is available, actually *proving* its optimality requires substantial computational effort.

The second set of test problems consists of *weighted* MAX2SAT problems from Borchers and Furman [13]. The same observations hold as for the unweighted problems, although the difference is now somewhat less pronounced. All the algorithms perform somewhat better on these problems. The results are shown in Table 5.4.

	SDP		EDPLL	Minto
# clauses	time	nodes	time	time
100	101	125	1.36	12.9
150	101	108	2.04	16.3
200	58	61	23.5	34.1
250	137	117	235	171
300	61	44	874	149
350	161	122	40285	2155
400	100	82	20233	579
450	53	44	>12hr	1420
500	118	76	>12hr	3153

Table 5.4: Solution times (in seconds) for weighted MAX2SAT benchmark problems ($m = 50$) for different algorithms

5.4.4 Cutting planes

The results from the previous section for MAX2SAT can be improved upon (at least in terms of node counts) by adding some of the cuts described in Section 5.3.4 to the relaxations. The influence of added cuts is illustrated in Table 5.5. These results were obtained using the SDP solver CUTSDP (Karisch [85]) in the branching scheme described in Section 5.4.2. The solution times are for proving optimality only, and are given for two MAX2SAT instances from Table 5.3 and two from Table 5.4 (weighted). The solver CUTSDP uses a primal-dual predictor-corrector algorithm. It also solves sparse Newton systems at each iteration of the solution of MAX2SAT relaxation, but the algorithm still requires additional computations involving the dense primal matrix variable. As a consequence it is less efficient than the dual scaling method. However, the purpose of this section is to illustrate the effect of cutting planes on the tightness of the relaxation and thus on the size of the search tree. The CUTSDP software automatically adds (some of) the violated triangle inequalities described in Section 5.3.4.

It is clear from Table 5.5 that the introduction of cuts reduces the size of the search tree significantly, but increases the solution time of the relaxations at the nodes. The total solution times are not improved in general, and all the solution times are worse than those reported in Table 5.3 and Table 5.4 for the dual scaling method without cuts.

	with cuts		without cuts	
# clauses	time	nodes	time	nodes
100	283	25	206	78
450	396	38	403	191
100 (weighted)	180	20	228	116
450 (weighted)	270	28	80	40

Table 5.5: Solution times (in seconds) for MAX2SAT benchmark problems ($m = 50$) for the CUTSDP method (with and without cuts) in a branching framework

Nevertheless, since the number of branching nodes can be reduced significantly, it is a challenge to extend the dual scaling method to use cuts and to find the optimal trade-off between stronger relaxations and increased solution times.

5.4.5 A note on solving SAT problems via MAX2SAT

The computational results reported in the previous sections indicate that the SDP relaxation can be quite effective for obtaining exact MAX2SAT solutions and this yields promise for the future. Unfortunately, it appears that with the current general purpose state-of-the-art implementations, solving larger problems is still beyond reach. To solve MAX2SAT problems with 100 variables and a varying number of clauses requires hours and more. Let us emphasize that by incorporating additional techniques in the algorithm, such as warm starts and early cutoffs, we expect the performance to improve significantly. However, using our current implementation, it is not feasible to solve 3SAT via a MAX2SAT approach, since (as pointed out in Section 5.2.1) a 3SAT formula with m variables and n clauses, has an associated MAX2SAT formulation with $m + n$ variables and $7n$ weighted clauses. Moreover, when relating the performance guarantee for MAX2SAT to the original 3SAT formula, it reduces to a .801 approximation algorithm (see Trevisan et al. [127]). By using a relaxation more suited for 3SAT instances, Karloff and Zwick showed that a $\frac{7}{8}$ approximation algorithm can be obtained [86]; see also Secion 5.8.

In the next section, we derive a semidefinite relaxation for the SAT problem in general. This relaxation, when specifically applied to 3SAT, will turn out to be a special case of the Karloff-Zwick relaxation.

5.5 A semidefinite relaxation of the SAT problem

5.5.1 A sufficient condition for unsatisfiability

Again, we consider the elliptic approximation of SAT problems as introduced in Chapter 2. Let Φ be a CNF formula with associated matrix A. Then

$$\mathcal{E}(w) = \{x \in \mathbb{R}^m \mid x^T A^T W A x - 2w^T A x \leq r^T w\},$$

where $w \in \mathbb{R}^n$, $W = \text{diag}(w)$ and $r_k = \ell(\mathbf{C}_k)(\ell(\mathbf{C}_k) - 2)$.

As stated before (Lemma 2.4.3), if $\ell(\mathbf{C}_k) \leq 2$ then the inequality may be replaced by equality. This is in fact the crucial observation for obtaining the MAX2SAT model derived in Section 5.3.1.

Let us recall Theorem 2.4.1 and Corollary 2.4.2. These state that $x \in \mathcal{E}(w)$ for all $w \geq 0$, $w \neq 0$, is a necessary and sufficient condition for any $x \in \{-1,1\}^m$ to be a satisfying assignment (for 2SAT the condition can be relaxed to $x \in \mathcal{E}_2(w)$ for any given $w \geq 0$). Thus, for any satisfying assignment $x \in \{-1,1\}^m$ it must hold that

$$x^T A^T W A x - 2w^T A x - r^T w \leq 0, \tag{5.2}$$

for all $w \geq 0$. Reversing this argument gives us a sufficient condition for unsatisfiability.

Corollary 5.5.1 *If for some $w \geq 0$ it holds that $x \notin \mathcal{E}(w)$ for all $x \in \{-1,1\}^m$, then Φ is unsatisfiable.*

Relaxing the integrality constraint to a spherical constraint we obtain an alternative sufficient condition for unsatisfiability.

Corollary 5.5.2 *If for some $w \geq 0$ it holds that $x \notin \mathcal{E}(w)$ for all x such that $x^T x = m$, then Φ is unsatisfiable.*

Observe that the condition of the second corollary is weaker than that of the first, since $x \in \{-1,1\}^m$ implies $x^T x = m$, but not the other way round.

Let us illustrate the usefulness of this approach with an example.

Example 5.5.3 We consider the well known *pigeon hole* formulas, which can be stated as follows: *given $h+1$ pigeons and h holes, decide whether it is possible to put each pigeon in at least one hole, while no two pigeons are put in the same hole.*
For more details on this problem and its SAT encoding see Section 5.6.2. The set of clauses in such a formula can be divided into a set of *long* clauses and a set of *short* clauses. It can be shown that for the instance with h holes and $h+1$ pigeons, when all long clauses are given a weight of one, and all short clauses a weight of $h - 1 + \frac{2}{h+1}$, the minimal value of (5.2) over the sphere $x^T x = m$ is equal to $4(h - 1 + \frac{1}{h+1}) > 0$. These values can be explicitly computed using the specific structure of the eigenspace of the matrix $A^T W A$ associated with the pigeon hole formulas. Thus by Corollary 5.5.2 it follows that the pigeon hole formulas are unsatisfiable. We conclude that, even using just a low dimensional weight vector, pigeon hole formulas can be shown to be contradictory. □

5.5.2 A certificate of unsatisfiability based on eigenvalues

In this section we reformulate Corollary 5.5.2 to obtain a condition for unsatisfiability in terms of eigenvalues. To this end we rewrite (5.2). Introducing an additional $\{-1,1\}$ variable x_{m+1} we obtain the inequality

$$x^T A^T W A x - 2x_{m+1} w^T A x - r^T w \leq 0. \tag{5.3}$$

Once again, if Φ is satisfiable then inequality (5.3) is satisfied by some $\{x_1, \ldots, x_{m+1}\} \in \{-1,1\}^{m+1}$ for all $w \geq 0$.

We can rewrite condition (5.3) as $\tilde{x}^T Q(w)\tilde{x} \leq 0$, where $\tilde{x} := [x_1, \dots, x_m, x_{m+1}]$ and $Q(w)$ is the $(m+1) \times (m+1)$ matrix:

$$Q(w) := \begin{bmatrix} A^T W A - \frac{r^T w}{m} I & -A^T w \\ -w^T A & 0 \end{bmatrix}.$$

This is valid, since $x \in \{-1, 1\}^m$ implies that $x^T(\frac{r^T w}{m} I)x = r^T w$. As before, we can further add a correcting vector $u \in \mathbb{R}^m$ to $Q(w)$ to obtain

$$\tilde{Q}(u, w) := \begin{bmatrix} A^T W A - \frac{r^T w}{m} I - \operatorname{diag}(u) & -A^T w \\ -w^T A & e^T u \end{bmatrix}.$$

It is easy to verify that $\tilde{x}^T Q(w)\tilde{x} = \tilde{x}^T \tilde{Q}(u, w)\tilde{x}$.

By Corollary 5.5.1 we are interested in minimizing $\tilde{x}^T \tilde{Q}(u, w)\tilde{x}$ over the $m+1$–dimensional $\{-1, 1\}$–vectors. Following Corollary 5.5.2 we relax the integrality constraint to a single spherical constraint $\tilde{x}^T \tilde{x} = m + 1$. In particular, if $w \geq 0$ and u exist, such that the minimal value of $\tilde{x}^T \tilde{Q}(u, w)\tilde{x}$ under the spherical constraint is positive, then Φ cannot be satisfiable. This is equivalent to finding a pair $(w \geq 0, u)$ such that the minimal eigenvalue of $\tilde{Q}(u, w)$ is positive.

Definition 5.5.4 *A pair $(w \geq 0, u)$ is called a (u, w)-*certificate of unsatisfiability *if the minimal eigenvalue of $\tilde{Q}(u, w)$ is positive.*

Note that given a (u, w)-certificate of unsatisfiability, its validity can be verified in polynomial time by computing the minimal eigenvalue of $\tilde{Q}(u, w)$.

5.5.3 An eigenvalue optimization problem

The problem of finding a (u, w)-certificate of unsatisfiability can be expressed in terms of an eigenvalue optimization problem as follows.

$$(\mathrm{SD}_{SAT}) \qquad \begin{array}{ll} \sup & (m+1)\lambda \\ \text{s.t.} & \tilde{Q}(u, w) \succeq \lambda I, \\ & w \geq 0. \end{array}$$

If $\tilde{Q}(u, w) \succeq \lambda I$, then $\tilde{Q}(u, w) - \lambda I \succeq 0$. Thus it follows that $(w \geq 0, u)$ is a (u, w)-certificate of unsatisfiability, if (and only if) $\lambda > 0$. We call the optimal value of optimization problem (SD_{SAT}) the *gap* of the formula Φ (not to be confused with *duality gap*).

Definition 5.5.5 *The gap of a formula Φ is defined as the optimal value of the optimization problem (SD_{SAT}).*

$$gap(\Phi) := \sup_{w \geq 0, u} \; (m+1)\lambda_{\min}\left(\tilde{Q}(u, w)\right).$$

Thus, by Corollary 5.5.2, we have the following corollary.

Corollary 5.5.6 *If a formula Φ has a positive gap, it has a (u, w)-certificate of unsatisfiability and therefore Φ is a contradiction.*

We will show that the converse is also true if Φ is a *2-SAT* formula. Furthermore, the formulas corresponding to a specific type of covering problems, which include the notorious *pigeon hole problems* and *mutilated chess boards*, have a positive gap. Let us emphasize that having a (u, w)-certificate of unsatisfiability is still merely a sufficient condition for unsatisfiability.

5.5.4 The dual relaxation

We can obtain the dual of the optimization problem (SD_{SAT}) via the primal–dual pair (SP, SD) (see Section 5.2.2). It can be simplified to the following *semidefinite feasibility problem*:

$$(\mathrm{SP}_{SAT})\qquad \begin{array}{ll} \text{find} & Y \in \mathbb{R}^{m\times m},\ y \in \mathbb{R}^m \\ \text{s.t.} & a_k^T Y a_k - 2a_k^T y \le r_k, \quad 1 \le k \le n, \\ & \mathrm{diag}(Y) = e, \\ & Y \succeq yy^T. \end{array}$$

Since (SP_{SAT}) does not have an objective function, we adopt the convention that its optimal objective value is zero if a feasible solution exists, while it is $+\infty$ if (SP_{SAT}) is infeasible.

To verify that (SP_{SAT}) is indeed the dual of (SD_{SAT}), note that the constraint on the diagonal of Y follows by dualizing the constraints associated with the correcting vector, while the first set of constraints is obtained by rewriting the condition

$$\mathbf{Tr} \begin{bmatrix} a_k a_k^T - \frac{r_k}{m} I & -a_k \\ -a_k^T & 0 \end{bmatrix} \begin{bmatrix} Y & y \\ y^T & 1 \end{bmatrix} \le 0,$$

and using that $\mathrm{diag}(Y) = e$. The last constraint of (SD_{SAT}),

$$Y \succeq yy^T, \tag{5.4}$$

is referred to as the *semidefinite constraint.* It follows using the well known *Schur complement* reformulation (see e.g. [75]):

$$\begin{bmatrix} Y & y \\ y^T & 1 \end{bmatrix} \succeq 0 \Leftrightarrow Y - yy^T \succeq 0.$$

When a formula Φ is satisfiable with an assignment x, the solution $Y = xx^T$, $y = x$ is feasible in its associated dual relaxation (SP_{SAT}); note that then

$$a_k^T Y a_k - 2a_k^T y = (a_k^T x)^2 - 2a_k^T x \le r_k,$$

which is exactly the elliptic representation of clause $\mathbf{C}_k$ (see (eq. 2.5)). Furthermore, $x \in \{-1, 1\}^m$, hence $\mathrm{diag}(Y) = e$, while (5.4) clearly is satisfied as well.

Rather than via dualizing (SD_{SAT}), (SP_{SAT}) can be derived directly from the elliptic approximations of clauses in the following way. Expanding (2.5), it can be written as

$a_k^T x x^T a_k - 2a_k^T x \leq r_k$. This equation can be linearized by replacing the variables x_i by *vectors* $v_i \in \mathbb{R}^{m+1}$ with the additional requirement that $\|v_i\| = 1$, adding a homogenizing vector v_{m+1}, and letting $Y = V^T V$ and $y = V^T v_{m+1}$ (here V is the matrix with the vectors v_i as columns). The constraint on the diagonal of Y follows immediately, and the semidefinite constraint follows using that $[V \ v_{m+1}]^T [V \ v_{m+1}] \succeq 0$ and the Schur complement. If Φ allows the satisfying assignment x, a feasible solution in terms of V is constructed by setting all entries of the vectors v_i to zero, except the first entries: the first entry of the vectors v_i ($1 \leq i \leq m$) is set to x_i, while the first entry of v_{m+1} is set to one.

Finally, note that (SP$_{SAT}$) is closely related to the MAX3SAT relaxation proposed by Karloff and Zwick [86]. It appears to be slightly weaker, since in their formulation the constraints are further disaggregated; see also Section 5.8. In addition, we are interested in proving unsatisfiability, rather than in MAX–SAT solutions.

5.5.5 Properties of the gap relaxations

Let us now consider (SD$_{SAT}$) and (SP$_{SAT}$) to derive a number of properties of these formulations. First note that (SD$_{SAT}$) is strictly feasible, since there exist $w \geq 0$, u and λ such that the matrix $\tilde{Q}(u, w) - \lambda I$ is positive definite. Thus *perfect duality* holds, implying that unboundedness of (SD$_{SAT}$) implies infeasibility of (SP$_{SAT}$).

Corollary 5.5.7 *For any formula Φ, gap(Φ) is either zero or infinity. If gap(Φ) $= \infty$, then there exists a (u, w)-certificate of unsatisfiability, implying the unsatisfiability of Φ.*

We have an even stronger duality result.

Lemma 5.5.8 *For the primal–dual pair (SP$_{SAT}$, SD$_{SAT}$) exactly one of the following two duality relations holds:*

1. *(SP$_{SAT}$) is feasible and (SP$_{SAT}$, SD$_{SAT}$) have complementary optimal solutions;*
2. *(SP$_{SAT}$) is strictly infeasible and (SD$_{SAT}$) is unbounded.*

Proof:

1. Suppose (Y, y) is a feasible solution of (SP$_{SAT}$). The all–zero solution $u = 0$, $w = 0$, $\lambda = 0$ is feasible for (SD$_{SAT}$). This constitutes a complementary (and therefore optimal) solution. Thus strong duality holds.
2. If (SP$_{SAT}$) is infeasible, we conclude from the perfect duality relation that (SD$_{SAT}$) is unbounded. Thus it allows a solution ($w \geq 0, u$) such that $\lambda > 0$. Using this solution an improving direction for the objective function of (SD$_{SAT}$) can be constructed, since $\lambda_{\min}\left(\tilde{Q}(\alpha u, \alpha w)\right) = \alpha \lambda_{\min}\left(\tilde{Q}(u, w)\right)$. This implies that (SP$_{SAT}$) is *strictly* infeasible (see Section 5.2.2). □

Since either complementary solutions exist for (SD$_{SAT}$) and (SP$_{SAT}$), or (SP$_{SAT}$) is strictly infeasible, we have the following corollary (for a proof, see [38]).

Corollary 5.5.9 *Using semidefinite programming, it can be decided in polynomial time which of the two duality relations holds. Thus the existence of a (u, w)-certificate of unsatisfiability can be established in polynomial time.*

Finally, we have a result on monotonicity of the gap.

Lemma 5.5.10 *Let Φ be a CNF-formula and let $\Psi \subseteq \Phi$. Then it holds that $gap(\Psi) \leq gap(\Phi)$.*

Proof: Consider (SD_{SAT}). Add to (SD_{SAT}) the constraints that $w_k = 0$ for all clauses $\mathbf{C}_k$ which occur in Φ only. Solving this modified version of (SD_{SAT}), gap(Ψ) is obtained. Obviously, it is a more restricted version of gap(Φ), the lemma therefore follows. □

This leads immediately to the following corollary.

Corollary 5.5.11 *The gap is monotone under unit resolution.*

Proof: Unit resolution can be regarded as the addition of unit clauses to a formula. By the previous lemma, the gap cannot decrease. □

In the next sections the gap is investigated in more detail for some specific SAT problems.

5.6 The gap for various SAT problems

5.6.1 The gap for 2SAT problems

In this section we prove that for 2SAT formulas Φ gap(Φ) $= \infty$ if and only if Φ is unsatisfiable. As stated before, it is well known that 2SAT problems are in fact solvable in linear time [6]. Therefore we do not claim that the algorithm presented in this section is computationally attractive (though it does run in polynomial time); our intention is to highlight some properties of the gap approach. We make use of the notion of minimal unsatisfiability.

Definition 5.6.1 (Minimal unsatisfiability) *An unsatisfiable CNF formula Φ is said to be* minimal unsatisfiable *if a satisfiable formula is obtained by omitting any given clause from Φ.*

By Lemma 5.5.10 we can restrict ourselves to the case that Φ is a minimal unsatisfiable formula, as any unsatisfiable subformula has a minimal unsatisfiable subformula.

Lemma 5.6.2 *If a formula Φ is minimal unsatisfiable, then $Az \not\geq 0$ for all $z \neq 0$, $z \in \mathbb{R}^m$.*

Proof: If the condition in this lemma does not hold, the formula has a linear autarky (see Definition 2.2.4), contradicting the fact that it is minimal unsatisfiable. □

From this lemma we have the following corollary.

Corollary 5.6.3 *If Φ is minimal unsatisfiable, then A is of full rank.*

Now we can prove the key lemma of this section; the main theorem follows.

Lemma 5.6.4 *Let Φ be a minimal unsatisfiable 2SAT formula. Then $gap(\Phi) = \infty$.*

Proof: From the semidefinite constraint (5.4) we conclude that $a^TYa \geq (a^Ty)^2$ for any vector a. This implies that

$$(a_k^Ty)^2 - 2a_k^Ty \leq a_k^TYa_k - 2a_k^Ty \leq r_k \equiv 0,$$

since $r_k = 0$ for 2SAT (for all $k = 1, \ldots, n$), from which it follows that $0 \leq a_k^Ty \leq 2$. The minimal unsatisfiability of Φ implies that $y = 0$. Hence, $a_k^TYa_k = 0$ for all $k = 1, \ldots, m$. Since $Y = V^TV \succeq 0$ it must hold that $0 = a_k^TYa_k = \|Va_k\|^2$, or $Va_k = 0$ implying that $Ya_k = 0$. The a_k's must therefore lie in the nullspace of and since A is of full rank ($\text{rank}(A) = m$) this implies $Y = 0$, contradicting the condition $\text{diag}(Y) = e$ of (SP$_{SAT}$). We conclude that (SP$_{SAT}$) is infeasible, implying that $\text{gap}(\Phi) = \infty$. □

Theorem 5.6.5 *Let Φ be any 2SAT formula. It holds*

$$gap(\Phi) = \begin{cases} \infty & \textit{if } \Phi \textit{ unsatisfiable;} \\ 0 & \textit{if } \Phi \textit{ satisfiable.} \end{cases}$$

Proof: Let Φ be a unsatisfiable 2SAT formula and let Ψ be a minimal unsatisfiable subformula of Φ. By Lemma 5.6.4, we have that $\text{gap}(\Psi) = \infty$ and Lemma 5.5.10 implies that $\text{gap}(\Phi) = \infty$. Conversely, assume that Φ is a satisfiable 2SAT formula. Using any satisfying assignment x, a feasible solution $Y = xx^T$, $y = x$ to (SP$_{SAT}$) can be constructed, implying that $\text{gap}(\Phi) = 0$. □

If $\text{gap}(\Phi) = 0$ we can use the dual solution (Y, y) to construct a satisfying assignment x. First we set the entries of x corresponding to nonzero entries of y to $x = \text{sgn}(y)$ as from the proof of Lemma 5.6.4 we know that y is an autarky. For all clauses that are not yet satisfied, a satisfying assignment can be constructed by considering Y^*, which denotes the matrix Y restricted to the rows and columns corresponding to the zero entries of y. Note that $a_k^Ty = 0$ implies that $a_k^TYa_k = 0$, so $Y_{ij} = -a_{ki}a_{kj}$ where $I_k \cup J_k = \{i, j\}$. Thus we can assume that each of the rows and columns of Y^* contains an off-diagonal element that is equal to ± 1. Fixing all ± 1 elements, such a matrix Y^* can be completed to a rank one $\{-1, 1\}$ matrix that is feasible in (SP$_{SAT}$). From this matrix a $\{-1, 1\}$ solution x can easily be deduced. This construction is equivalent to the one given in [46].

To finish this section, let us consider the relation of (SP$_{SAT}$) and (SP$_{M2S}$). The latter is obtained by incorporating the constraints $a_k^TYa_k - 2a_k^Ty \leq 0$ in the objective function. Accordingly, if (SP$_{SAT}$) is feasible, there exists a solution of (SP$_{M2S}$) with nonpositive objective value. Feige and Goemans [46] show that for the extended MAX2SAT relaxation (i.e. the triangle inequalities are included), (SP$_{M2S}$) has objective value zero if and only if the instance under consideration is feasible.

5.6.2 The gap for a class of covering problems

In this section we consider SAT encodings of a particular class of covering problems, and show that these can be shown to be contradictory by our SDP approach. Let V be a set of m propositional variables. Let $\mathcal{S} = \{S_1, \ldots, S_N\}$ and $\mathcal{T} = \{T_1, \ldots, T_M\}$ be sets of subsets of V. Both $\mathcal{S}$ and $\mathcal{T}$ form a partition of V. Furthermore, let us assume that $M < N$. Consider the following CNF formula Φ_{CP}.

$$\bigvee_{i \in S_k} p_i, \qquad 1 \leq k \leq N, \tag{5.5}$$

$$\neg p_i \vee \neg p_j, \quad i, j \in T_k, i \neq j, \ 1 \leq k \leq M. \tag{5.6}$$

An equivalent generic formulation is considered by Impagliazzo et al. [76]. First, let us give two examples of problems that fit in this format.

Pigeon hole formulas. *Given $h+1$ pigeons and h holes, decide whether it is possible to put each pigeon in at least one hole, while no two pigeons must be put in the same hole.* We now argue that the standard encoding of the pigeon hole problem fits the format Φ_{CP}. For each pigeon–hole pair a proposition is introduced; thus, we obtain a total of $h(h+1)$ variables in the set V. The long (positive) clauses (5.5) now express that each pigeon is put in at least one hole; thus there are exactly $N = h+1$ of such clauses that all have length h. The short (negative) clauses (5.6) model that no two pigeons must be put in the same hole simultaneously; for each hole there is a set of short clauses, giving rise to $M = h$ separate sets T_k, each of size $h+1$. It is easy to see that each proposition occurs both exactly once in the sets S_k and exactly once in the sets T_k, thus both $\mathcal{S}$ and $\mathcal{T}$ are a partition of V.

The mutilated chess board problem. *Given a chess board of size $2s \times 2s$ squares. Two of its diagonally opposite vertices are removed. Decide whether it is possible to cover the resulting 'mutilated' chess board by rectangular dominoes of size 2×1 (i.e. a single domino covers exactly two adjacent squares), such that each square is covered exactly once.* The standard satisfiability coding for this problem is obtained by introducing a proposition for each pair of adjacent squares; thus we need $4(2s^2 - s - 1)$ variables. For each square there is a positive clause (of length 2, 3 or 4) expressing that it must be covered at least once, and a set of negative (2–)clauses expressing that it must be covered at most once. Taking a subset of this set of clauses, a formula of the form Φ_{CP} is obtained. For all the *black* squares we keep the positive clauses, while for all the *white* squares we use only the negative clauses. The first set corresponds to (5.5) and the second set to (5.6). See also the graph in Figure 5.2; the nodes correspond to individual variables, the drawn edges indicate the *positive* clauses (all cliques with drawn edges constitute one positive clause) and the dotted edges the *negative* clauses (each pair of nodes connected by a dotted edge corresponds to a negative clause). Again, it is easy to see that each variable occurs in exactly one of the $N = 2s^2$ positive clauses, and in exactly one of the $M = 2s^2 - 2$ sets of negative clauses.

Both of the above problems are easily concluded to be unsatisfiable. We verify that Φ_{CP} is unsatisfiable (when $M < N$), using *cutting planes* (this construction is well known, see e.g. [29]). To this end we introduce some more notation. By e_S (e_{ij}) we denote the vector with ones in the positions $i \in S$ (i and j) and zeros elsewhere. The integer linear feasibility formulation can be stated as

$$(\mathrm{IP}_{CP}) \quad \begin{array}{lll} \text{find} & x \in \{-1,1\}^m & \\ \text{s.t.} & e_{S_k}^T x \geq 2 - |S_k|, & 1 \leq k \leq N, \\ & e_{ij}^T x \leq 0, & i,j \in T_k, i \neq j,\ 1 \leq k \leq M. \end{array}$$

Obviously, it is easy to find a solution of the linear relaxation of this problem (i.e. when the integrality constraints are relaxed to $-e \leq x \leq e$), by setting all variables to 0. However, using cutting planes one can show that for $\{-1,1\}$-variables, the set of inequalities for a set T_k implies that

$$e_{T_k}^T x \leq 2 - |T_k|,\ 1 \leq k \leq M \tag{5.7}$$

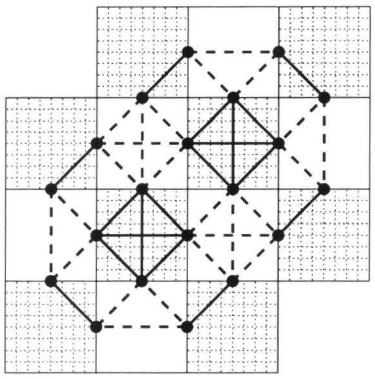


Figure 5.2: The mutilated chess board for $s = 2$.

(see [29]). A cutting plane is obtained by taking nonnegative combinations of the constraints, and subsequently adjusting the right hand side of the resulting inequality such that it is as tight as possible. To determine this right hand side, we use the fact that the variables must be binary [24].

For completeness, let us review the derivation of (5.7). Taking a subset $U \subseteq T_k$, $|U| = 3$, and summing the (three) inequalities associated with this set, the inequality $2e_U^T x \leq 0$ is obtained. Since for any $\{-1, 1\}$ vector x it holds that $e_U^T x$ is *odd*, the right hand side can be rounded down to the largest odd integer smaller than zero, thus we find that $e_U^T x \leq -1$. More generally, suppose we are given a set $U \subset T_k$ and an inequality $e_U^T x \leq 2 - |U|$. In addition let $j \in T_k \backslash U$, and denote $\overline{U} = U \cup \{j\}$. Summing for all $i \in U$ the inequalities $e_{ij}^T x \leq 0$ (with weight 1) and the initial inequality with weight $|U| - 1$, we obtain that $|U| e_{\overline{U}}^T x \leq (|U| - 1)(2 - |U|)$. Dividing both sides by $|U|$, and rounding the right hand side down to the nearest integer with same parity as $|\overline{U}|$ (thus the right hand side becomes $1 - |U|$, which is valid, since $((|U| - 1)/|U|)(2 - |U|) < 3 - |U|$) we obtain

$$e_{\overline{U}}^T x \leq 1 - |U| = 2 - |\overline{U}|.$$

We conclude that (5.7) is indeed implied by the inequalities in (IP$_{CP}$).

Summing all the inequalities (5.7), and using that $\mathcal{T}$ partitions V, we find that $-m \leq e^T x \leq 2M - m$. Similarly, taking the sum over the first set of inequalities in (IP$_{CP}$), we have $2N - m \leq e^T x \leq m$. Combining, we get

$$2N \leq e^T x + m \leq 2M, \tag{5.8}$$

from which it follows that $N \leq M$, implying the infeasibility of (IP$_{CP}$) and thus the unsatisfiability of Φ_{CP} when $M < N$. We conclude that using this cutting plane technique, a proof of unsatisfiability of Φ_{CP} can be constructed and verified in polynomial time. Surprisingly, other techniques often require large running times to solve formulas of this type. Indeed, Haken [63] proves that no short resolution proof of the unsatisfiability of pigeon hole formulas exists. The length of any resolution proof is exponential in the size of the formula. It is strongly conjectured that the mutilated chess board problem does not allow a polynomial size resolution proof either, despite the absence of 'long' clauses [133].

Note that to find the cutting plane proof sketched above efficiently, additional problem-specific information is used. In each step of the cutting plane proof, two linear inequalities are used to derive a new linear inequality. The cutting plane proof of the class (Φ_{CP}) is called a *tree-like* cutting plane proof [76], as each inequality is used only once. To construct such a short tree-like cutting plane proof additional searching is required, since the order in which linear inequalities are combined is crucial. For general CNF formulas, it is not clear how to efficiently find a (tree-like) cutting plane proof, even if it is known that one exists. It is our aim to show in this section that using our semidefinite programming approach formulas of the format Φ_{CP} are proven to be contradictory in polynomial time, while no additional problem-specific information whatsoever is required. This is due to the fact that (Φ_{CP}) allows a (u, w)-certificate of unsatisfiability whose existence can be computed in polynomial time (Corollary 5.5.9).

Let us consider the semidefinite relaxation of Φ_{CP}. The SDP relaxation can be denoted as (see also (SP_{SAT}))

$$(\text{SD}_{CP}) \qquad \begin{array}{lll} \text{find} & Y \in \mathbb{R}^{m \times m}, y \in \mathbb{R}^m & \\ \text{s.t.} & e_{S_k}^T Y e_{S_k} - 2e_{S_k}^T y \leq |S_k|(|S_k| - 2), & 1 \leq k \leq N \\ & e_{ij}^T Y e_{ij} + 2e_{ij}^T y \leq 0, & i, j, i \neq j \in T_k,\ 1 \leq k \leq M \\ & \text{diag}(Y) = e, & \\ & Y \succeq yy^T. & \end{array}$$

We prove the following theorem.

Theorem 5.6.6 *The semidefinite relaxation (SD_{CP}) of Φ_{CP} is infeasible (if $M < N$). Equivalently, gap(Φ_{CP}) $= \infty$.*

Proof: Note that from the semidefinite constraint (see (eq. (5.4)) it follows that $a^T Y a \geq (a^T y)^2$ for any m-vector a. Thus it follows that

$$(e_{S_k}^T y)^2 - 2e_{S_k}^T y \leq e_{S_k}^T Y e_{S_k} - 2e_{S_k}^T y \leq |S_k|(|S_k| - 2),$$

implying that

$$2 - |S_k| \leq e_{S_k}^T y \leq |S_k|,\ 1 \leq k \leq N. \tag{5.9}$$

Now we consider the inequalities corresponding to the sets T_k. Taking the sum over all the inequalities corresponding to set T_k, k fixed, we find that

$$e_{T_k}^T Y e_{T_k} + (|T_k| - 2)e_{T_k}^T \text{diag}(Y) + 2(|T_k| - 1)e_{T_k}^T y \leq 0.$$

To verify this, note that each diagonal element Y_{ii}, $i \in T_k$, occurs in exactly $|T_k| - 1$ inequalities; similarly, each linear term y_i, $i \in T_k$, occurs in exactly $|T_k| - 1$ inequalities as well. Simplifying this expression using that diag(Y) $= e$, we obtain

$$e_{T_k}^T Y e_{T_k} + 2(|T_k| - 1)e_{T_k}^T y \leq -|T_k|(|T_k| - 2).$$

Using the semidefinite constraint again, we conclude that

$$(e_{T_k}^T y)^2 + 2(|T_k| - 1)e_{T_k}^T y \leq e_{T_k}^T Y e_{T_k} + 2(|T_k| - 1)e_{T_k}^T y \leq -|T_k|(|T_k| - 2),$$

implying that

$$-|T_k| \leq e_{T_k}^T y \leq 2 - |T_k|, \; 1 \leq k \leq M.$$

Summing these inequalities we find that $-m \leq e^T y \leq 2M - m$ while from (5.9) we have that $2N - m \leq e^T y \leq m$, implying that $2N \leq e^T y + m \leq 2M$ (note that this is equivalent to what we obtained using cutting planes, see (5.8)). Thus we conclude that (SD$_{CP}$) is infeasible when $N > M$. □

Consequently, the Corollaries 5.5.7 and 5.5.9 yield the following corollary.

Corollary 5.6.7 *Using semidefinite programming, the unsatisfiability of Φ_{CP} can be decided in polynomial time.*

It may be noted that the proof of Theorem 5.6.6 and the cutting plane refutation of Φ_{CP} are essentially very similar. Indeed, the cutting planes (5.7) are automatically implied in (SD$_{CP}$).

Application to graph colouring

A famous result of Lovász is his 'sandwich' theorem [94], which states that for an undirected graph $G = (V, E)$ in polynomial time (using semidefinite programming), a number $\vartheta(G)$ can be computed which is bounded from above by the graph's colouring number $\gamma(G)$ (i.e. the minimal number of colours required to colour the vertices of the graph such that no two adjacent vertices have the same colour), and from below by its clique number $\omega(G)$ (i.e. the maximal complete subgraph of G). Applying our result from the previous section to the graph colouring problem (GCP) we obtain a similar result.

Let $G = (V, E)$ be an undirected graph and let C be a set of colours. We introduce a proposition for each vertex-colour combination. Then the GCP can be modelled as a formula Φ_{GCP} containing a set of $|V|$ long clauses, expressing that each vertex should be coloured by at least one colour, and a set of short clauses expressing that no two vertices must get the same colour. We can construct the semidefinite relaxation of Φ_{GCP} in the usual way; we refer to it as (SD$_{GCP}$). Now let C^* be the smallest set of colours for which (SD$_{GCP}$) is feasible. Such a set must exist, since for $|C^*| \geq |V|$ the GCP and thus its relaxation (SD$_{GCP}$) are trivially feasible. We have the following theorem.

Theorem 5.6.8 $\omega(G) \leq |C^*| \leq \gamma(G)$.

Proof: First note for any set of colour C with $|C| < \omega(G)$, Φ_{GCP} has a subformula of the form Φ_{CP}, hence by Theorem 5.6.6 and Lemma 5.5.10 it has gap infinity. This subformula corresponds to the set of clauses associated with a clique of size $|C| + 1$ or larger. Now consider set C^*. It holds that $|C^*| \geq \omega(G)$, since otherwise (SD$_{GCP}$) would be infeasible. Also, since removing one colour from C^* implies infeasibility of (SD$_{GCP}$) (by assumption), it holds that $|C^*| \leq \gamma(G)$. This proves the theorem. □

Hence, by applying a binary search on the size of C, a number similar to Lovász ϑ–number can be computed.

5.6.3 The gap for 3SAT problems

We have seen that two specific classes of CNF formulas can be efficiently handled by the gap approach. Let us now turn our attention to the most general class of CNF formulas, the 3SAT problems. First we have a negative result for pure 3SAT problems.

Lemma 5.6.9 *Let Φ be a pure 3SAT problem. It holds that gap $(\Phi) = 0$.*

Proof: It is easy to verify that the solution $Y = I$, $y = 0$ is a feasible assignment of (SP_{SAT}) since $a_k^T a_k = 3 = r_k$ for all clauses. By duality, $\mathrm{gap}(\Phi) = 0$. □

Corollary 5.6.10 *No pure 3SAT formula allows a (u, w)-certificate of unsatisfiability.*

Note that the proof of this lemma and the corollary easily extend to general CNF formulas in which no clauses of length one and/or two occur.

Of course the gap can be computed in nodes of a branching tree; during the branching process 2-clauses are created, thus allowing the SDP approach to provide a certificate of unsatisfiability in specific cases. Note that (also) in this respect, the SDP approach is stronger than the LP approach (as mentioned in Section 2.2). However, computing the gap is computationally rather expensive, and especially in a DPLL-like branching algorithm (including unit resolution, although this can be simulated by the semidefinite relaxation as well) the overhead will be substantial with the current state-of-the-art implementations. Two possibilities in this respect are:

1. Instead of using a primal-dual algorithm, it is possible to use a dual scaling algorithm to exploit sparsity of $\tilde{Q}$ to the full [9]. Furthermore, it can be used that all constraint matrices are of rank one, similar to the MAX2SAT case in Section 5.4. The computational experience in Section 5.4 with MAX2SAT problems indicates that for small sized problems this approach is competitive with other dedicated algorithms for the MAX2SAT problem.

2. For larger problems, spectral bundle methods can be used to solve the eigenvalue optimization problem (SD_{SAT}) [70]. These have been shown to be able to handle problems with thousands of variables. Such methods solve only (SD_{SAT}), so that the dual information is lost.

For now, let us slightly reformulate our gap relaxation to provide a minor result for 3SAT formulas. Consider again the elliptic representation $\mathcal{E}_k$ associated with clause $\mathbf{C}_k$ (see eq. (2.5)). By introducing a *parity*-variable, the inequality is turned into an equality constraint.

$$\mathcal{E}_k^p = \{x \in \mathbb{R}^n, 0 \leq s_k \leq 1 \mid (a_k^T x - 1)^2 + 4s_k = 4\},$$

where for all feasible $\{-1, 1\}$ assignments x it holds that $s_k \in \{0, 1\}$. The ellipsoid $\mathcal{E}_k^p$ has the semidefinite relaxation

$$a_k^T Y a_k - 2a_k^T y + 4s_k = 3.$$

Obviously, the trivial solution (Lemma 5.6.9), is still feasible when simply setting all s_k to 0. However, if we maximize the sum of the s_k's, a solution other than the trivial one may be obtained. Thus we define the *parity* gap. Consider the semidefinite optimization problem (SP$_{par}$).

$$(\mathrm{SP}_{par}) \qquad \begin{array}{rll} \max & \sum_{k=1}^{n} s_k & \\ \text{s.t.} & a_k^T Y a_k - 2a_k^T y + 4s_k = 3, & 1 \le k \le n, \\ & \operatorname{diag}(Y) = e, & \\ & Y \succeq yy^T, & \\ & 0 \le s_k \le 1, & 1 \le k \le n. \end{array}$$

Definition 5.6.11 *The optimal value of optimization problem (SP$_{par}$) is called the parity gap of a formula Φ.*

We then have the following lemma.

Lemma 5.6.12 *If a formula Φ has parity gap zero it is equivalent to an XOR-SAT formula and as such efficiently solvable.*

Proof: If the parity gap is zero, $s_k = 0$ for all $1 \le k \le n$. Suppose that a solution x exists for which at least one clause is satisfied in exactly two literals, hence $a_k^T x = 1$. Using this solution, a solution $Y = xx^T$, $y = x$ of (SP$_{par}$) can be constructed. Since $a_k^T Y a_k - 2a_k^T y = -1$, $s_k = 1$ implying that the parity gap is not equal to zero. If (SP$_{par}$) has objective value zero, no such solution exists. This implies that if a satisfying assignment exists, then it must satisfy each clause in exactly one or three literals. Thus the formula is equivalent to an XOR-SAT formula. Such formulas are solvable in polynomial time [111]; see Section 4.3. □

A specific class of formulas that appears likely to have parity gap zero, is the class of *doubly balanced formulas* (see Chapter 4). By definition, for doubly balanced formulas it holds that $A^T A$ is a diagonal matrix and $A^T e \equiv 0$. Eliminating the s_k variables from (SP$_{par}$) it can be rewritten as

$$(\mathrm{SP'}_{par}) \qquad \begin{array}{rll} \min & \mathbf{Tr} \begin{bmatrix} A^T A & -A^T e \\ -e^T A & 0 \end{bmatrix} \begin{bmatrix} Y & y \\ y^T & 1 \end{bmatrix} & \\ \text{s.t.} & -1 \le a_k^T Y a_k - 2a_k^T y \le 3, & 1 \le k \le n, \\ & \operatorname{diag}(Y) = e, & \\ & Y \succeq yy^T. & \end{array}$$

It holds that $opt(\mathrm{SP}_{par}) = \frac{3}{4}n - \frac{1}{4}opt(\mathrm{SP'}_{par})$. Note that the objective function of (SP'$_{par}$) is essentially the same objective function as that of (M2S) (see Section 5.3.1). Using formulation (SP'$_{par}$) it is straightforward to prove the next lemma.

Lemma 5.6.13 *A doubly balanced formula has parity gap zero.*

Proof: By the definition of doubly balancedness and the constraint on the diagonal of Y the objective function of (SP'$_{par}$) reduces to the constant $3n$, implying that the parity gap is equal to zero. □

We also have an autarky result.

Lemma 5.6.14 *If all $s_k > \frac{3}{4}$, $x = sgn(y)$ is a satisfiable assignment of Φ.*

Proof: Note that

$$(a_k^T y)^2 - 2a_k^T y + (4s_k - 3) \leq 0.$$

This implies that

$$1 - 2\sqrt{1 - s_k} \leq a_k^T y \leq 1 + 2\sqrt{1 - s_k}.$$

If $s_k > \frac{3}{4}$, then $a_k^T y > 0$. If all s_k have this property, then y is a linear autarky. □

Note that y might be a linear autarky while not all s_k are larger than $\frac{3}{4}$. In this respect, a slightly stronger formulation is obtained by using *a single* slack variable t for all clauses, rather than a separate slack variable s_k for each of the clauses. The objective then becomes to maximize t; all s_k's must be replaced by t, and equality must be replaced by inequality. A drawback of this approach is that an optimum of zero implies only that a polynomially solvable *sub*formula is present. On the other hand, solving this subformula first, may speed up solving the full formula (as demonstrated in Chapter 4).

While it appears that the semidefinite relaxation as described in this chapter is not quite strong enough to solve 3SAT problems by itself, it can be exploited in various other ways. A couple of these are mentioned in the next section.

5.7 Some computational experiences

5.7.1 Approximating MAX-SAT solutions

Both the primal and dual solutions obtained by solving the semidefinite relaxations can be used for heuristic purposes. The dual solution (Y, y) might be used to try to obtain good approximate MAX-SAT solutions. To illustrate the quality of the solutions thus obtainable, see Figure 5.3. We restricted ourselves to a set of random pure 3SAT formulas with 100 variables and a varying number of clauses, and used the dual formulation with a single slack variable. The SDPs were solved using the public-domain solver SeDuMi (Sturm [124]). Approximate solutions are constructed by (*i*) taking the dual solution and rounding y (the solid line), (*ii*) by applying a Goemans-Williamson-like randomized rounding procedure (the dashed line). For comparison we also included the solutions obtained by simply drawing random solutions (dash-dot); note that this in fact corresponds to the Karloff-Zwick algorithm for approximating (pure) MAX3SAT solutions. To obtain a good *ad hoc* lower bound on the optimal solution we applied a greedy weighted local search procedure, which is briefly described in Section 1.4.3. Considering the results, it is clear that the local search procedure gives the best results, but interestingly enough the drawn line is consistently above the dashed line, implying that the deterministic one-step procedure of rounding y gives better solutions than the randomized rounding procedure, which in turn is better than drawing random solutions.

Alternatively, using the *primal* optimal solution w, it is possible to identify hard subformulas. To this end, the sum of the weights is bounded (this corresponds to using a single slack variable in the dual formulation). Removing the clauses with 'small' weights, a subformula is obtained that appears to be the 'core' of the original formula. This yields a technique for finding approximately minimal unsatisfiable subformulas.

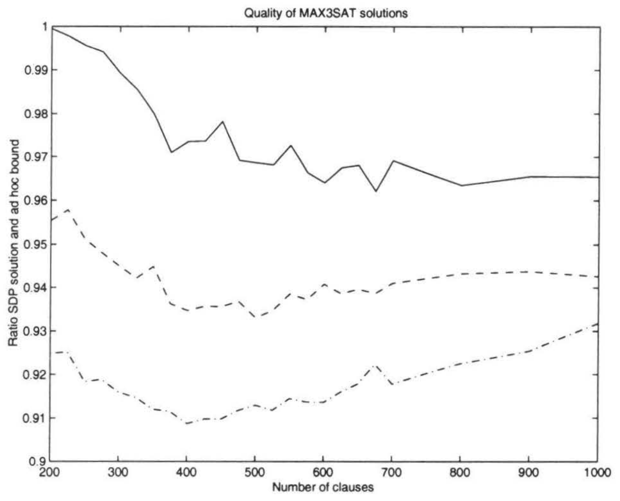


Figure 5.3: Comparison of MAXSAT solutions.

5.7.2 Reduction in size of search trees

Let us now consider the reduction of the search tree sizes when incorporating the relaxation in a branching algorithm. In Table 5.6 the average node counts for the DPLL with and without the SDP approach are given. The averages are taken over 20 problems of each size. On inspection of the table, we see that the size of the search trees decreases, but by a factor of two only. In addition, due to the generic method of counting nodes (i.e. a node is counted each time the branching rule is called) this number is deceptive. The number of SDPs that has to be solved is actually larger and therefore also included in the table. Thus it appears that the reduction of the search trees is somewhat disappointing; consequently, tighter SDP relaxations and/or improved branching rules are required, the latter possibly making use of the solution to the SDP. This is the subject of further research. In the next section we mention several approaches to obtain tighter relaxations.

5.8 Constructing tighter relaxations

So far we have considered a fixed quadratic clause-model based on elliptic approximations only. Obviously there exist many other quadratic clause-models. Generic procedures for deriving increasingly tighter models of growing dimensions for binary programming problems in general are given by Sherali and Adams [119] and Lovász and Schrijver [95]. These models eventually describe the convex hull. While these results are of great theoretical interest, it is not entirely clear how to implement them in practice, due to the exponential size of the resulting relaxations. For now, let us restrict ourselves to the most general valid quadratic models for individual clauses. Such models must satisfy the following condition. If x is a satisfying assignment then

$$q(x) = \sum_{i=1}^{m}\sum_{j=1}^{m} q_{ij}x_ix_j + \sum_{k=1}^{m} q_kx_k + q_0 \leq 0.$$

		DPLL	DPLL+SDP	
m	SAT/UNSAT	nodes	nodes	SDPs
50	UNSAT	26.9	16.4	23.4
	SAT	19.6	13.5	17.7
75	UNSAT	97.3	54.1	80.9
	SAT	38.1	26.4	32.8
100	UNSAT	293	160	235
	SAT	92.4	56.1	77.8
125	UNSAT	804	443	649
	SAT	211	122	171
150	UNSAT	1764	974	1415
	SAT	300	169	233

Table 5.6: Reduction of size of search trees using SDP.

As an example, let us consider the clause $\mathbf{C} = p_1 \vee p_2 \vee p_3$. Substituting the satisfying assignments of $\mathbf{C}$ in $q(x)$, a set of 7 inequalities is obtained. These inequalities, which are linear in the coefficients q_{ij} and q_k, define a cone. The extreme rays of this cone give rise to the following generic set of valid quadratic cuts.

$$\begin{aligned}
x_1x_2 + x_1x_3 - x_2 - x_3 &\leq 0\\
x_1x_2 + x_2x_3 - x_1 - x_3 &\leq 0\\
x_1x_3 + x_2x_3 - x_1 - x_2 &\leq 0\\
-x_1x_2 - x_1x_3 - x_2x_3 - 1 &\leq 0\\
-x_1x_2 + x_1 + x_2 - 1 &\leq 0\\
-x_1x_3 + x_1 + x_3 - 1 &\leq 0\\
-x_2x_3 + x_2 + x_3 - 1 &\leq 0
\end{aligned}$$

An equivalent set is given by Karloff and Zwick [86]. Each valid quadratic model of $\mathbf{C}$ is a positive linear combination of this set of inequalities. In particular, the elliptic approximation is obtained by taking the sum of the first three inequalities (see (2.5) and use that $x^2 = 1$ for $x \in \{-1, 1\}$). Thus, our relaxation is an aggregated version of the Karloff-Zwick relaxation.

Karloff and Zwick show that using the first three inequalities a 7/8 approximation algorithm for MAX3SAT problems can be obtained. Their relaxation has $4n$ inequality constraints where the coefficient matrices of the constraints are of rank 3. One can still solve the resulting problem by the dual scaling method (see [9]), but the assembly of the linear system at each iteration becomes more expensive, and its coefficient matrix becomes more dense. The question is therefore if these relaxations can be solved quickly enough to allow incorporation in a branch and cut scheme.

Generic sets of valid quadratic cuts for clauses of length four and longer can be derived in a similar manner. Note that the sizes of these sets grow with increasing clause lengths.

For example, the generic set associated with 4-clauses contains 23 inequalities. Halperin and Zwick [64] show that thus a .8721 approximation algorithm for MAX4SAT can be developed.

As stated before, the SDP relaxations can be further strengthened by adding valid inequalities. An example of such cuts are the *triangle inequalities* (see Section 5.3.4); note that many of these are in fact implied by the generic set given above (namely those concerning variables that occur jointly in some clause). Another possibility is to construct elliptic approximations of *pairs* of clauses, for instance

$$(a_k^T x - 1)(a_l^T x - 1) \leq (\ell(\mathbf{C}_k)) - 1)(\ell(\mathbf{C}_l) - 1). \tag{5.10}$$

All valid quadratic cuts may be derived for pairs of clauses as well.

6

A Nonlinear Approach To Combinatorial Optimization

We consider combinatorial optimization problems, modelled as linear programs with binary variables. We discuss a method to construct nonconvex optimization problems with continuous variables, such that its optimal solutions yield feasible binary solutions of the original problem. Two applications are examined: the Satisfiability problem and the Frequency Assignment Problem. Minimization algorithms for solving the nonconvex model are discussed. As an indication of the remarkable strength of the proposed method, we conclude with computational experiences on both real-life and randomly generated instances of the frequency assignment problem.

6.1 Introduction

Recent developments in optimization have lead to the application of nonlinear optimization techniques to solve combinatorial optimization problems. For example, in several papers nonlinear models are used to obtain effective heuristic algorithms (see e.g. Kamath et al. [82], Karmarkar et al. [88], Gu [61]) for specific combinatorial optimization problems. Other applications include semidefinite relaxations such as discussed in Chapter 5. In the master's thesis [135] a nonconvex quadratic model for the *frequency assignment problem* (FAP) is developed. The FAP belongs to the class of so-called *node covering* problems. This a class of NP-complete problems to which also the *stable set* and *graph colouring* problems belong. Such problems are usually modelled using binary variables. The model developed in [135] involves a nonconvex quadratic objective function, which is optimized over the unit hyper cube under some additional equality constraints. It exhibits the property that – provided that the FAP under consideration is feasible – its global optimal value is known, and any solution that is feasible in the original FAP is a globally optimal solution of the quadratic model and vice versa; we call this the *equivalence property.* In the present chapter an alternative derivation of the quadratic model is given and we show that any fractional solution that is feasible in the quadratic model can be efficiently rounded to a binary solution that is at least as good in terms of objective values; we refer to this as the *rounding property.* Thus, relaxing the binary variables to continuous variables yields no deterioration of the model. The model with continuous variables is essentially fully equivalent to the model with binary variables, as the rounding property implies that the global minima of both models coincide. Note that usually (for example in the linear case) relaxing the integrality constraints deteriorates the model in the sense that it is difficult to obtain a satisfactory binary solution from a (fractional) solution of the relaxed problem. In general, the relaxation provides bounds on the optimal value of the original problem only.

To find approximately optimal solutions of the model, in [135] a second-order interior-point potential reduction algorithm is used. This algorithm is inspired by an algorithm developed by Karmarkar et al. [82, 88], who applied the method to several difficult optimization problems, including the *set covering* problem [88] and the *satisfiability problem* [82]. The model they use is nonconvex quadratic as well. Similar to the model developed in [135] it exhibits the equivalence property; however, it does not feature the rounding property. Obviously, due to the models' nonconvexity, in either case the algorithms do not have a guarantee of finding a global optimum, but when a global optimum is found, it is recognized as such by construction.

The aim of this chapter is twofold.

- We give an alternative derivation of the model for node covering problems, and prove the rounding property. Subsequently, we address the problem of constructing nonconvex models with continuous variables that feature the equivalence and rounding property for more general combinatorial optimization problems (namely, all combinatorial optimization problems that can be modelled as binary linear programs). It turns out that such models involve *pseudo-Boolean* and *multi-linear* functions, which are in fact typically those functions that feature the rounding property. To obtain models with the desired properties, a technique first given by Granot and Hammer [57] can be applied. Unfortunately, the size of the reformulation may be intractable. It may be emphasized that for many important problem classes the reformulation can be done efficiently; for instance, both the node covering problems studied earlier and the *satisfiability* (SAT) problem allow a linear-time reformulation. The model thus obtained for SAT problems, is in fact a well known and studied model [16, 55, 61, 140]. Alternatively, a generally efficient reformulation method can be obtained using the CNF representations of linear inequalities discussed in Appendix A.

- We develop an algorithm that is based on the observation that the rounding property and its associated procedure can essentially be interpreted as a gradient descent algorithm. Thus, an efficient first-order descent algorithm is obtained, which on the instances we consider outperforms the potential reduction algorithm.

This chapter is organized as follows. In the next section we describe the techniques to obtain the reformulation; we start with deriving the nonconvex model for node covering problems, and subsequently generalize the basic ideas to the general case. Some algorithmic approaches to the given minimization problem are discussed, and we consider two special cases for which the reformulation can be done efficiently. We conclude with computational results on the frequency assignment problem and some remarks.

6.2 Preliminaries and notation

We consider binary feasibility problems of the form (BFP):

$$\text{(BFP)} \quad \text{find } x \in \{0,1\}^m \text{ such that } Ax \leq b,\ Bx = d.$$

Here $A \in \mathbb{R}^{n \times m}$, $B \in \mathbb{R}^{p \times m}$, $b \in \mathbb{R}^n$, $d \in \mathbb{R}^p$ and it is assumed that all data are integral. Note that we use $\{0,1\}$ variables here, rather than $\{-1,1\}$ variables which we have used elsewhere in this thesis. Obviously, theoretically this is completely equivalent; for our present purpose we feel that $\{0,1\}$ variables allow a more transparent presentation. Note that the property that $x_i^2 = 1$ for any $x_i \in \{-1,1\}$ has as $\{0,1\}$ equivalent that $x_i^2 = x_i$, i.e. $\{0,1\}$ variables are idempotent.

Many combinatorial optimization problems can be put in the form (BFP) by modelling them as integer linear programming problems and, if required, adding a bound on the objective function value. It is our aim to model (BFP) as a (nonconvex) minimization problem (NCP) with continuous variables:

$$\text{(NCP)} \quad \begin{array}{ll} \min & \mathcal{P}(x) \\ \text{s.t.} & Cx \leq c. \end{array}$$

We require the following properties.

Equivalence property. *A $\{0,1\}$ solution x is feasible in (BFP) if and only if it is a global minimizer of (NCP) such that $\mathcal{P}(x) = 0$.*

Rounding property. *For any vector x such that $Cx \leq c$, via an efficient rounding procedure a binary vector $[x]$ can be obtained such that $C[x] \leq c$ and $\mathcal{P}([x]) \leq \mathcal{P}(x)$.*

We observe that in the case of box constraints (i.e. $Cx \leq x$ denotes $0 \leq x \leq e$), Granot and Hammer [57] call the function $\mathcal{P}(x)$ the *resolvent* of (BFP). They give a method to obtain such a resolvent; it is derived in Section 6.3.2. The term 'resolvent' emphasizes the close correspondence between logical formulas and feasibility (or optimization) problems with binary variables. Indeed, the method to obtain a resolvent can be interpreted as a method of constructing a SAT problem whose solutions coincide with the solutions of (BFP); the resolvent is in fact a logical formula in *disjunctive normal form* expressing which (partial) assignments cause infeasibility. An alternative (more efficient) method to construct a SAT problem to represent (BFP) is given in Appendix A.

In the next section we concentrate on deriving $\mathcal{P}(x)$ without reference to the logical interpretation. Only in Section 6.5.1 we show explicitly that linear systems corresponding to CNF formulas have a direct resolvent.

Let us introduce some additional notation.

- By Q_T the square matrix with row and column indices in T is denoted;
- $\rho(a)$ denotes the *support* (the set of nonzero coordinates) of the vector a.

Remark: To avoid potential confusion, it is emphasized that the matrix A above is in general *not* the clause-variable as used elsewhere in this thesis.

6.3 A nonconvex continuous model

We first consider a specific kind of binary feasibility problems, the *node covering* problems, for which a model of the form (NCP) is derived [135]. The derivation of a model with the desired properties can, in part, be interpreted as an application of Granot and Hammer's method. Subsequently, we derive their method for finding a resolvent of (BFP) in its general form. Interestingly, the rounding property is not explicitly mentioned by Granot and Hammer. Their interest lies in applying and developing techniques using binary variables rather than continuous ones. Only in later work by various authors continuous variables are used; see for example [14, 16, 55, 61].

6.3.1 A special class of binary feasibility problems

In accordance with [135], let us make some assumptions on the matrices involved in (BFP).

Assumption 1 *All elements of A and B are binary.*

Assumption 2 *The right hand side vector b equals the all-one vector e.*

Assumption 3 *The sets $\rho(b_k^T)$ are a* partition *of the index set $\{1, \ldots, m\}$.*

Assumption 4 *For each pair of an equality constraint k, $1 \leq k \leq p$, and an inequality constraint l, $1 \leq l \leq n$, we have that $|\rho(b_k^T) \cap \rho(a_l^T)| \leq 1$.*

Let us denote the binary feasibility problem that satisfies these assumptions as (BFP$_0$). By E_k we denote the support of equality constraint k, i.e. $E_k = \rho(b_k^T)$. Also, $J_k = \rho(a_k^T)$. From Assumption 1 it follows that the equality constraints are of the form

$$e_{E_k}^T x = d_k, \quad 1 \leq k \leq p.$$

Similarly, by Assumptions 1 and 2 the inequality constraints are of the form

$$e_{J_k}^T x \leq 1, \quad 1 \leq k \leq n.$$

Furthermore, Assumption 3 implies the following proposition.

Proposition 6.3.1 *The matrix B is totally unimodular.*

Proof: It is easy to verify that each square submatrix of B has determinant 0 or 1. □

Using a property of totally unimodular matrices we state a corollary.

Corollary 6.3.2 *All vertices of the polytope $Bx = d$ are integral (provided d is integral).*

Assumption 4 is required later on to prove the rounding property.

Let us now have a closer look at the inequalities $Ax \leq b$. If for some k, $1 \leq k \leq n$, we have that $|J_k| \geq 3$, the corresponding inequality is equivalent to a set of $\frac{1}{2}|J_k|(|J_k| - 1)$ inequalities: $x_i + x_j \leq 1$, $i < j \in J_k$. Note that inequality k can be obtained from this set of inequalities by means of *Chvátal cutting planes* [24]. Define the set

$$\mathcal{M} = \{(i, j) \mid \exists\, k \in \{1, \ldots, n\} \text{ with } i, j \in J_k,\ 1 \leq i < j \leq m\},$$

then we can rewrite (BFP_0) as

$$(\mathrm{BFP}_0^*) \quad \text{find } x \in \{0,1\}^m \text{ such that } Bx = d \text{ and } x_i + x_j \leq 1,\ (i,j) \in \mathcal{M}.$$

The relaxation of (BFP_0^*) is

$$(\mathrm{RFP}_0^*) \quad \text{find } x \in [0,1]^m \text{ such that } Bx = d \text{ and } x_i + x_j \leq 1,\ (i,j) \in \mathcal{M}.$$

A solution x to (RFP_0^*) may be (at least partly) fractional and cannot be easily rounded to a feasible binary solution. However, if we replace the constraints $x_i + x_j \leq 1$ by $x_i x_j = 0$, thus obtaining the following feasibility problem

$$(\mathrm{NFP}_0^*) \quad \text{find } x \in [0,1]^m \text{ such that } Bx = d \text{ and } x_i x_j = 0,\ (i,j) \in \mathcal{M},$$

a solution x to (NFP_0^*) can straightforwardly be rounded to the binary solution $\lceil x \rceil$ for which we have that $\lceil x_i \rceil \lceil x_j \rceil = 0$ for all $(i,j) \in \mathcal{M}$. From $\lceil x \rceil$ one easily can obtain one or more feasible solutions of (BFP_0), as long as Assumptions 1 and 3 hold. Since B is binary, it holds that $B\lceil x \rceil \geq d$. For each equality constraint, we can simply set variables with index in its support from one to zero, until it is satisfied (see also [135]). Note that this also directly follows using Proposition 6.3.1 and its corollary. Applying the rounding procedure amounts to optimizing a linear objective function (either 'min $e^T y$' or 'max $e^T y$'; in both cases the optimal value is $e^T d$, which is attained for each feasible solution) subject to the linear constraints $By = d$ and $0 \leq y \leq \lceil x \rceil$, where x denotes the (fractional) solution to (NFP_0^*). Since B is totally unimodular, any basic solution y must be binary.

Observe that the constraints $x_i x_j = 0$ in (NFP_0^*) are nonconvex; this makes them difficult to handle as *constraints*. Instead, they can be transferred to the objective function. Thus the following minimization problem is obtained:

$$(\mathrm{NQP}) \qquad \begin{array}{ll} \min & \sum_{(i,j)\in\mathcal{M}} x_i x_j \\ \text{s.t.} & Bx = d, \\ & 0 \leq x \leq e. \end{array}$$

Obviously, the optimal value of this minimization problem is nonnegative and when (BFP_0) is feasible, it is equal to zero. Problem (NQP) inherits from (NFP_0^*) that a fractional solution x with objective value zero can be rounded to (possibly many) different feasible solutions. Indeed, the number of solutions obtainable by rounding can be enormous (see [137, 138]). Note that for a given feasible binary solution the objective value is integral; it is equal to the number of constraint violations in (BFP_0^*).

We introduce the (symmetric) matrix Q (see also [135]):

$$Q = \operatorname{sgn}\left[A^T A - \operatorname{diag}(A^T A)\right], \tag{6.1}$$

where $\operatorname{diag}(A^T A)$ denotes the diagonal matrix containing the diagonal entries of the matrix $A^T A$. Note that the matrix Q is similar to the matrices involved in the elliptic approximations introduced in Chapter 2. In fact, the inequalities $Ax \leq e$ can be interpreted to be a 2CNF formula. Accordingly, the quadratic model (without taking into account the equality constraints) derived in this section is the $\{0,1\}$ equivalent of the elliptic representation of 2CNF formulas.

Due to Assumption 1, the matrix Q is binary. Denoting the entries of Q by Q_{ij}, we have the following relation.

Lemma 6.3.3 *It holds that $Q_{ij} = 1$, $i \leq j$, if and only if $(i,j) \in \mathcal{M}$.*

Proof: Denote by a_i the ith column of A. We prove only the 'if' part here; the proof for the 'only if' part is similar and left to the reader. Suppose that $Q_{ij} = \text{sgn}(a_i^T a_j) = 1$. By the definition of Q we know that $i \neq j$ and since A is binary and therefore nonnegative, there must exist a $k \in \{1,\ldots,n\}$ such that $i,j \in J_k$. Thus $(i,j) \in \mathcal{M}$. □

It is clear from this lemma that the number of nonzero entries of the matrix Q is equal to $2|\mathcal{M}|$. Due to the lemma, we can rewrite (NQP) as

$$(\text{NQP}) \qquad \begin{array}{ll} \min & \frac{1}{2}x^T Q x \\ \text{s.t.} & Bx = d, \\ & 0 \leq x \leq e. \end{array}$$

Let us summarize our results hitherto in a theorem.

Theorem 6.3.4 (Equivalence property) *Consider the problems (NQP) and (BFP$_0$). It holds that*

- *The optimal value of (NQP) is zero if and only if (BFP$_0$) is feasible.*
- *A fractional solution of (NQP) with objective value zero yields multiple feasible (binary) solutions of (BFP$_0$).*

If (BFP$_0$) is infeasible, a global minimum of (NQP) will not be equal to zero. However, due to Assumption 4 (NQP) features the *rounding property*. Hence all strict minimizers (local and global) of (NQP) have integral objective values; in addition, each minimizer with integral objective value yields one ore more binary solutions with the same objective value. We will derive these facts below.

Let us assume that x is a fractional feasible solution of (NQP). It is our aim to construct a feasible binary solution $\tilde{x} = x + \Delta x$, such that the objective value of $\tilde{x}$ is at least as good as that of x. This amounts to finding a vector Δx such that $x + \Delta x$ is binary and

$$\tfrac{1}{2}(\Delta x)^T Q(\Delta x) + q^T(\Delta x) \leq 0, \tag{6.2}$$

where q is the gradient of the objective function in x, i.e. $q = Qx$. Such a vector could be found by solving the following minimization problem.

$$(\text{QP}\Delta) \qquad \begin{array}{lll} \min & \frac{1}{2}(\Delta x)^T Q(\Delta x) + q^T(\Delta x) & \\ \text{s.t.} & B\Delta x = 0 & \\ & \Delta x_i \in \{-x_i, 1 - x_i\}, & i = 1,\ldots,m. \end{array}$$

Note that solving (QPΔ) is equivalent to solving (NQP). Consequently, it is in general intractable to solve it directly. Under Assumption 4 however, an approximate solution that satisfies (6.2), can be found by solving a sequence of at most p subproblems that all can be solved in polynomial time. We first state a proposition.

Proposition 6.3.5 *Under Assumption 4, $Q_{E_k} \equiv 0$, for all k, $1 \leq k \leq p$.*

Proof: Denoting the ith column of A by a_i, it holds that $Q_{ij} = \text{sgn}(a_i^T a_j)$, $i \neq j$, while by construction $Q_{ij} = 0$ for $i = j$. Let $i, j \in E_k$, $i \neq j$. Then, by Assumption 4, there does not exist an l such that $i, j \in J_l$, implying that $Q_{ij} = 0$. □

Now consider the linear relaxation of (QPΔ), restricted to the variables Δx_i, $i \in E_k$, for some $k \in \{1, \ldots, p\}$. Using the last proposition, a linear program (rather than a nonconvex quadratic program) emerges. We use the change of variables $y = (\Delta x) + x$, to find

$$(\text{LP}_k) \qquad \begin{array}{ll} \min & q_{E_k}^T y \\ \text{s.t.} & e_{E_k}^T y = e_{E_k}^T x \\ & 0 \leq y \leq e. \end{array}$$

It is well known that linear programming problems can be solved in polynomial time [87]. Furthermore, due to the total unimodularity of B, the next proposition follows.

Proposition 6.3.6 *(LP$_k$) allows an optimal solution y^* with $y_{E_k}^*$ binary.*

Note that, using the structure of its constraints, (LP$_k$) can be solved more efficiently than a general linear program. It is in fact the linear relaxation of an easy *knapsack* problem, whose structure allows an optimal solution with the property stated in the proposition. To find such a solution, it suffices to sort an array of $|E_k|$ elements. It is easily verified that the following strategy yields an optimal solution of (LP$_k$). Let $S_k = \{i_1, \ldots, i_{d_k}\}$ be the set of indices corresponding to the d_k lowest gradient entries q_i, $i \in E_k$, and let $y^* := x + (e_{S_k} - x_{E_k})$; this implies that $y_i^* = x_i$, $i \notin E_k$, $y_i^* = 1$, $i \in S_k$, $y_i^* = 0$, $i \in E_k \backslash S_k$. It is easy to see that $e_{E_k}^T y = d_k = e_{E_k}^T x$, thus feasibility is maintained. Now we are ready to prove the rounding property.

Theorem 6.3.7 (Rounding property) *Using any feasible solution x of (NQP), a feasible binary solution $\tilde{x}$ of (NQP) can be constructed, such that $\frac{1}{2}\tilde{x}^T Q \tilde{x} \leq \lfloor \frac{1}{2} x^T Q x \rfloor$.*

Proof: Since the sets E_k are disjoint (Assumption 3), we can apply the same procedure for the variables occurring in each set E_k separately. Choose a $k \in \{1, \ldots, p\}$ for which x_{E_k} is not a binary vector. Solve the corresponding problem (LP$_k$). Above, an optimal solution y^* to this problem is derived. It remains to show that the (optimal) objective value does not increase, or in other words that $q^T y^* \leq q^T x$. To this end, using the set S_k defined above, we introduce the scalar ξ:

$$0 \leq \max_{i \in S_k} q_i \leq \xi \leq \min_{i \in E_k \backslash S_k} q_i.$$

We have that

$$\begin{aligned} q^T y^* - q^T x = q_{E_k}^T y^* - q_{E_k}^T x &= q_{E_k}^T (y^* - x) = q_{S_k}^T (e - x) - q_{E_k \backslash S_k}^T x \\ &\leq \xi \left[e_{S_k}^T (e - x) - e_{E_k \backslash S_k}^T x \right] = \xi \left[d_k - e_{E_k}^T x \right] = 0. \end{aligned}$$

Now let $x := y^*$. Obviously, x_{E_k} is now binary. If x still contains fractional values, a new subproblem (LP$_k$) for an appropriate k must be solved. This is repeated until a binary solution $\tilde{x}$ has been constructed. Using the fact that any binary solution must have an integral objective value, we conclude that $\frac{1}{2}\tilde{x}^TQ\tilde{x} \leq \lfloor \frac{1}{2}x^TQx \rfloor$. □

The proof of the rounding property is constructive and thus provides us with an efficient *rounding procedure* to round any fractional solution to a binary solution with improved objective value. This procedure requires at most p iterations, where in each iteration an array of on average m/p elements needs to be sorted. Note that if $d = e$, only the minimum of an array of on average m/p elements must be found. In section 6.4.2 the method for approximately solving (QPΔ) is generalized to an efficient gradient descent algorithm for solving model (NQP). We have the following corollaries.

Corollary 6.3.8 *Given a fractional solution x with objective value less than one, the rounding property guarantees the existence of a binary solution with objective value zero, which is a feasible solution of (BFP$_0$). This solution can be constructed efficiently using the rounding procedure.*

Note that the order in which the variables are rounded may influence the quality of the solution obtained. It is not clear beforehand which is the optimal order.

Corollary 6.3.9 *Since Q is binary, any strict minimum (local or global) of (NQP) has an integral objective value and corresponding binary minimizer.*

It is easy to see that the objective value is equal to the number of constraint violations in (BFP$_0^*$) (as pointed out earlier). Note that the rounding property holds for any non-negative matrix $\tilde{Q}$ with the same nonzero structure as Q. This allows us to weight the constraints; if the constraint concerning the variables i_1 and j_1 is considered to be more important than the constraint concerning a pair of variables i_2 and j_2 we can set $\tilde{Q}_{i_1j_1}$ to a large value N, while setting $\tilde{Q}_{i_2j_2}$ to one. We will exploit this observation in Section 6.4.3.

Corollary 6.3.10 *Any (random) feasible solution x of (NQP) yields the upper bound $\lfloor \frac{1}{2}x^TQx \rfloor$ on the minimal number of inequality constraint violations in (BFP$_0^*$).*

This corollary also follows directly from the rounding property. For example, assuming that $d = \kappa e$, $|E_k| = m/p$, for all $k = 1, \ldots, p$, and $\kappa \in \mathbb{N}$, an easy to compute upper bound is

$$\text{maximum number of constraint violations} = \left\lfloor \left(\frac{\kappa p}{m}\right)^2 |\mathcal{M}| \right\rfloor. \tag{6.3}$$

Note that this is in fact the expected objective value of a random solution in which each variable is set to 1 with probability $\frac{\kappa p}{m}$ and to 0 with probability $1 - \frac{\kappa p}{m}$. Thus, the rounding procedure provides us with a deterministic procedure to verify a probabilistic statement. This also follows from the *probabilistic method* (see Alon and Spencer [2]); we come back to this in the next section. For specific applications (such as the Frequency Assignment Problem; see Section 6.5.2) the ratio $\kappa p/m$ is relatively small, thus guaranteeing that a substantial fraction, namely $1 - (\kappa p/m)^2$, of the constraints can be satisfied *in polynomial time*. In this sense, our rounding technique can be regarded as an approximation algorithm with a performance guarantee. Note that due to the nature of the rounding procedure always solutions are produced that satisfy the equality constraints.

Unfortunately, the performance guarantee, in most (non trivial) cases, will not be satisfactory since we are interested in finding feasible binary solutions of (BFP$_0$). More powerful minimization techniques for solving (NQP) must be used to obtain (hopefully) better solutions. Such techniques are developed in Section 6.4. To finish off this section, let us show that there exist instances for which the bound (6.3) is tight.

Example 6.3.11 Consider the instance that is constructed as follows. For $1 \leq k \leq 3 = p$, $E_k = \{2k-1, 2k\}$, and $\mathcal{M} = \{(1,3),(2,3),(1,4),(2,5)\}$. Furthermore, $\kappa = 1$. It is easy to verify that the assumptions 3-4 are satisfied. A solution of the feasibility problem of finding $x \in \{0,1\}^6$ with $e_{E_k}^T x = 1$, $1 \leq k \leq p$, and $x_i + x_j \leq 1$, $(i,j) \in \mathcal{M}$, is $x_2 = 1$, $x_4 = 1$, $x_6 = 1$. Constructing the associated quadratic model and substituting the solution $x_i = \frac{1}{2}$, we find that $(\kappa p/m)^2|\mathcal{M}| = 1$, implying that after rounding at most one constraint will be violated. Note that $x^TQx = x_1(x_3 + x_4) + x_2(x_3 + x_5)$. Now consider the rounding procedure for $k = 1$, i.e. solve (LP$_1$). This amounts to minimizing $x_1 + x_2$ subject to $x_1 + x_2 = 1$. Obviously, there are two binary optimal solutions. Choosing $x_1 = 1$, $x_2 = 0$ and propagating, a solution violating one constraint is obtained. Choosing $x_1 = 0$, $x_2 = 1$ and propagating, the feasible solution is found. □

6.3.2 The general case

In the previous subsection a nonconvex quadratic model equivalent to (BFP) has been derived under the assumption that the matrices A and B have a special structure. Let us now consider the more general case. But first, let us analyze the characteristics of model (NQP). We make three observations:

1. For all $k \in \{1, \ldots, p\}$ the following holds. If all variables x_i, $i \notin E_k$, are fixed, while only the variables x_i, $i \in E_k$, remain free, the nonconvex quadratic function x^TQx reduces to a *linear* function in the variables x_i, $i \in E_k$.

2. The matrix B is totally unimodular.

3. The sets E_k, $k = 1, \ldots, p$, partition the index set $\{1, \ldots, m\}$.

These observations are in fact crucial for obtaining the desired rounding property. Thus we aim for a model (NCP) that has similar properties. Note that for the first observation to hold, $\mathcal{P}(x)$ must be linear in each of its variables. Thus $\mathcal{P}(x)$ must be a *multilinear* function; i.e. it must be of the form

$$\mathcal{P}(x) = \sum_{k=1}^{M} c_k \prod_{i \in J_k} x_i, \tag{6.4}$$

where $c_k \in \mathbb{R}$, and the sets J_k are index sets. Multilinear functions are also known as *pseudo-Boolean* functions (Hammer et al. [57, 65, 66]). Clearly, the objective function of (NQP) is bilinear, hence it fits the form (6.4). The term 'multilinear' is explained by noting that such a function takes its extrema (both minima and maxima) at vertices of the unit hyper cube. For completeness, we include this as a theorem (see also e.g. [14, 16, 61]).

Theorem 6.3.12 (Rounding property) *Let x be such that $0 \leq x \leq e$. An efficient rounding strategy is available to construct a binary solution $[x]$ with the property that $\mathcal{P}([x]) \leq \mathcal{P}(x)$.*

Proof: If x is binary, then let $[x] = x$. Otherwise, there is an i such that $0 < x_i < 1$. Fix all variables at their current value, except the variable x_i. Then $\mathcal{P}(x)$ reduces to a linear function in one variable, that reaches its minimum on the interval $[0, 1]$ in either $x_i = 0$ or $x_i = 1$. Accordingly, fix x_i at its appropriate value. Repeating this procedure for all fractional variables, a binary solution is constructed such that $\mathcal{P}([x]) \leq \mathcal{P}(x)$. □

Thus, similar to the previous section, the rounding procedure to verify the rounding property can be interpreted as optimizing a linear function over a (in this case one dimensional) polytope with integral vertices. In the previous section, by making use of the 'nice' structure of a subset of the constraints, higher dimensional problems could be considered in the rounding procedure. In general, if the constraints (both equality and inequality) do not have such a structure, they can all be incorporated in the multilinear function $\mathcal{P}(x)$. Note that the Corollaries 6.3.8-6.3.10 can be straightforwardly generalized to the present case.

As mentioned before, the rounding property also follows from the *probabilistic method* [2]. For a given x, $0 \leq x \leq e$, the value of $\mathcal{P}(x)$ is the *expected value* of a binary solution y that is obtained by setting y_i to 1 with probability x_i and to 0 with probability $1 - x_i$. The probabilistic method specifies that there must exist a binary solution with this expected value. Moreover, the method of *conditional probabilities* yields such a solution in polynomial time [108]. This method coincides with the proof of Theorem 6.3.12 (albeit with different terminology).

Below we discuss how to construct a multilinear function that is the resolvent of a single inequality constraint. First, let us briefly consider equality constraints $Bx = d$. These can be processed in several ways. The preference for any of the methods below should depend on the particular structure of $Bx = d$.

- Replace by $Bx \leq d$ and $Bx \geq d$ and treat as inequality constraints.
- Add $(Bx - d)^T(Bx - d)$ to $\mathcal{P}(x)$, and use idempotency of the variables to eliminate the quadratic terms. Thus only bilinear and linear terms emerge; the rounding property remains intact.
- Use $Bx = d$ to eliminate a number of variables so as to reduce the problem size.
- If the equality constraints can be used to lift the rounding procedure into higher dimension (as in the special case of the previous section), then leave them intact.

Let us now address the problem of constructing a resolvent $\mathcal{P}(x)$ with the desired properties. This construction was first given in [66]. Let us first consider some examples.

Example 6.3.13 In the previous section inequalities of the form $x_1 + x_2 \leq 1$ were considered, with resolvent $x_1 x_2 = 0$. Clearly, it holds both for *continuous* and *binary* variables that $x_1 x_2 = 0$ implies that $x_1 + x_2 \leq 1$. This shows that the nonlinear formulation is stronger than the linear one, as the converse is true for binary variables only. Now consider the following inequality:

$$x_1 + x_2 + x_3 + x_4 \leq 2. \tag{6.5}$$

Again, we must have that $x_1 x_2 x_3 x_4 = 0$, but obviously this does not imply that (6.5) is satisfied, since at least *two* x_i need to be equal to zero. Inequality (6.5) may be replaced by the following four inequalities:

$$\begin{array}{lllll} & +x_2 & +x_3 & +x_4 & \leq 2 \\ x_1 & & +x_3 & +x_4 & \leq 2 \\ x_1 & +x_2 & & +x_4 & \leq 2 \\ x_1 & +x_2 & +x_3 & & \leq 2 \end{array} \tag{6.6}$$

Note that inequality (6.5) in fact can be interpreted as a *Chvátal cut* [24], that can be obtained by adding the four inequalities (6.6) and rounding the right hand side downwards. Recall that a cut forces the variables in the linear relaxation to integer values. For the integer formulation however, it is redundant. Accordingly, for a *binary* solution, we have that (6.6) is satisfied if and only if (6.5) is satisfied. Each of the inequalities in (6.6) is satisfied if at least *one* of the variables in it is zero, similar to the first case above. Thus we obtain the resolvent of (6.5):

$$x_2x_3x_4 + x_1x_2x_4 + x_1x_3x_4 + x_1x_2x_3 = 0.$$

If this equation is satisfied by a solution $0 \leq x \leq e$, then system (6.6) is satisfied, implying that (6.5) is satisfied. □

In the following we formalize what is exposed by the example. Let us first verify that all coefficients may be assumed to be nonnegative. Consider the inequality $a^Tx \leq a_0$. Define $J^+ = \{i : a_i > 0\}$, $J^- = \{i : a_i < 0\}$ and $J = J^+ \cup J^-$. Then it can be rewritten as

$$a_{J^+}^T x - a_{J^-}^T(e - x) = a_{J^+}^T x + |a_{J^-}^T|(e - x) \leq a_0 - e_{J^-}^T a.$$

Since $e - x$ is binary if x is binary, we may assume without loss of generality that $a \geq 0$. We first consider inequalities of a particular form. By e we denote an all-one vector of appropriate length.

Definition 6.3.14 *The* maximum violation ν *of an inequality* $e^Tx \leq r$ *is defined as*

$$\nu = \max_{x \in \{0,1\}^t} e^Tx - r = t - r.$$

Obviously, if $\nu \leq 0$, the associated inequality is trivially satisfied for any binary vector x. Therefore in the following we consider only inequalities that have a strictly positive maximum violation. Note that for the first inequality in the example $\nu = 1$, while for inequality (6.5) $\nu = 2$. For the inequalities (6.6) however, we have that $\nu = 1$.

Let us state and prove an easy lemma.

Lemma 6.3.15 *Let* $e^Tx \leq r$ *denote an inequality with maximum violation* ν. *Let* x *be such that* $0 \leq x \leq e$. *Then* $\nu \leq 1$ *if and only if*

$$p(x) = \prod_{i=1}^{t} x_i = 0 \Rightarrow e^Tx \leq r.$$

Proof: Assume the implication holds. Then $x_i = 0$ for some i. Thus $e^Tx \leq t - 1 \leq r$. It follows that $\nu = t - r \leq 1$. On the other hand, suppose that $\nu \leq 1$. Assume that for some x it holds that $p(x) = 0$, thus $x_i = 0$ for some i. Hence $e^Tx \leq t - 1 = \nu + r - 1 \leq r$. □

Lemma 6.3.15 suggests the approach to replace the inequality $a^Tx \leq a_0$ by an equivalent set of inequalities having binary coefficients and maximum violation one. This can be done by means of *minimal covers* (see e.g. [104]).

Definition 6.3.16 (Minimal cover) *A set M is called a* minimal cover *of inequality $a^T x \leq a_0$, if for all $i \in M$ the following holds:*

$$e_{M\setminus\{i\}}^T a \leq a_0 < e_M^T a.$$

For a given inequality, assume that we have found all its distinct minimal covers and stored these in the set $\mathcal{M} = \{M_1, M_2, \dots, M_{|\mathcal{M}|}\}$. With each minimal cover $M_j \in \mathcal{M}$ we associate an inequality

$$e_{M_j}^T x \leq |M_j| - 1,$$

and we denote the union of these inequalities by $Ux \leq Ue - e$. It is obvious that all inequalities in $Ux \leq Ue - e$ have a maximum violation of one. Furthermore, we can prove the following equivalency.

Lemma 6.3.17 *For binary x, $a^T x \leq a_0$ if and only if $Ux \leq Ue - e$.*

Proof: Assume we are given x such that $a^T x \leq a_0$. Suppose that for some $j = 1, 2, \dots, |\mathcal{M}|$ we have that $e_{M_j}^T x > |M_j| - 1$. This implies that $x \geq e_{M_j}$, from which it follows that $a^T x \geq a_{M_j}^T x = e_{M_j}^T a > a_0$. This contradicts the fact that $a^T x \leq a_0$. We conclude that $Ux \leq Ue - e$. Conversely, assume we are given an x such that $Ux \leq Ue - e$. Let $\overline{J} = \{i \in J : x_i = 1\}$. Suppose that $a^T x = a_{\overline{J}}^T x = e_{\overline{J}}^T a > a_0$. Sort the indices $i \in \overline{J}$ such that $a_{i_1} \geq a_{i_2} \geq \dots \geq a_{i_m}$. Denote by e_k the vector with ones in the first k positions, and zeros elsewhere. Now for some k it holds that $e_k^T a \leq a_0$, while $e_{k+1}^T a > a_0$. By construction, $\{i_1, i_2, \dots, i_{k+1}\} = M_j$ for some j, which leads again to a contradiction. □

Using these lemmas, we associate a multilinear function with an inequality $a^T x \leq a_0$ as follows.

$$P(x) = \sum_{j=1}^{|\mathcal{M}|} p_j(x) = \sum_{j=1}^{|\mathcal{M}|} \prod_{i \in M_j} x_i.$$

Hence each $p_j(x)$ is associated with a distinct minimal cover. We are ready to prove the equivalence property.

Theorem 6.3.18 (Equivalence property) *Given are an inequality $a^T x \leq a_0$ and its associated function $P(x)$. For any x such that $0 \leq x \leq e$, $P(x) \geq 0$. If $P(x) = 0$, then $a^T [x] \leq a_0$, where $[x]$ denotes a binary solution that is obtained by rounding all fractional elements of x either up or down. Vice versa, if a binary x satisfies $a^T x \leq a_0$, then $P(x) = 0$.*

Proof: By construction it is clear that $P(x) \geq 0$ for any $0 \leq x \leq e$. Also, if $P(x) = 0$ for some $0 \leq x \leq e$, then by construction $P([x]) = 0$ and thus $p_j([x]) = 0$ for all j. By Lemma 6.3.15 it follows that $U[x] \leq Ue - e$, and hence using Lemma 6.3.17, $a^T [x] \leq a_0$. The proof of the converse statement is left to the reader. □

We state two corollaries.

Corollary 6.3.19 *The degree of the multilinear function associated with an inequality $a^T x \leq a_0$ is equal to the size of its largest minimal cover.*

Corollary 6.3.20 *From a (partly) fractional vector x, $0 \leq x \leq e$, for which it holds that $P(x) = 0$, multiple binary solutions $[x]$ such that $a^T [x] \leq a_0$ can be constructed.*

Now it is easy to construct the resolvent of (BFP). Each inequality k is (if necessary) replaced by its equivalent set of inequalities with maximum violation one, and its associated multilinear function $P_k(x)$ is constructed. Summing the functions $P_k(x)$, we conclude that

$$\mathcal{P}(x) = \sum_{k=1}^{n} P_k(x)$$

is the resolvent of (BFP) and can be used as the objective function of (NCP). (BFP) is feasible if and only if the optimal value of (NCP) equals zero. By Corollary 6.3.20 a minimizer of (NCP) yields one or possibly multiple feasible solutions when it is rounded to a binary solution. If a binary solution has a positive objective value, it can be interpreted as follows.

Corollary 6.3.21 *The objective value of a binary solution x of (NCP) is equal to the number of minimal covers in the reformulation of (BFP) that is completely covered by x.*

Thus the objective value gives an upper bound on the number of constraint violations in the original formulation (BFP). Note that the formulation can be adjusted such that the objective value is equal to the number of constraint violations in the original problem; without going into details, to this end *extended covers* that cover sets of minimal covers need to be enumerated and evaluated. If the rounding property is applied to obtain approximation results this may be worthwhile. Here we do not pursue it.

Example 6.3.22 Consider the inequality

$$4x_1 + 6x_2 - 3x_3 - 5x_4 + 10x_5 \le 7.$$

We rewrite this inequality as

$$4x_1 + 6x_2 + 3(1 - x_3) + 5(1 - x_4) + 10x_5 \le 15.$$

The inequality has 5 minimal covers of sizes $2, 3, 3, 3$ and 4. The largest minimal cover has size 4; consequently, the polynomial $P(x)$ has degree 4. The associated multilinear function is

$$P_1(x) = x_1x_2(1 - x_3)(1 - x_4) + (1 - x_3)(1 - x_4)x_5 + x_1(1 - x_3)x_5 + x_1(1 - x_4)x_5 + x_2x_5.$$

An additional inequality is given by $3x_1 + 5x_2 + 2x_3 - 6x_5 \le 4$, with resolvent

$$P_2(x) = x_2(1 - x_5) + x_1x_3(1 - x_5).$$

Considering $\mathcal{P}(x) = P_1(x) + P_2(x)$ it is clear that x_2 must be equal to zero. After substitution of $x_2 = 0$, $\mathcal{P}(x)$ has 4 terms. Substituting $x_i = \frac{1}{2}$, $i = 1, 3, 4, 5$, we find $\mathcal{P}(x) = \frac{1}{2}$, implying that the rounding procedure must yield a feasible solution. Keeping to the lexicographic order we obtain the partial solution $\tilde{x} = (0, 0, 1, *, *)$, with $\mathcal{P}(\tilde{x}) = 0$, which yields four distinct feasible solutions. □

Let us briefly consider the complexity of finding all minimal covers of a given inequality. For a constraint of length m of the form $e^T x \le r$, we can say more beforehand about the size of its set of minimal covers.

Lemma 6.3.23 *The number of minimal covers of an inequality $e^T x \leq r$, is equal to*

$$|\mathcal{M}| = \begin{pmatrix} m \\ r+1 \end{pmatrix}.$$

Proof: To construct *all* minimal covers of $e^T x \leq r$, one needs to find all sets M_j such that $|M_j| = r + 1$. This amounts to finding all combinations of $r + 1$ elements out of m elements. □

For $r = \frac{1}{2}m$, this number is exponential in m. In specific applications performing the procedure as previously described is therefore computationally intractable. However, using the linear time algorithm to construct CNF equivalents of inequalities as discussed in Appendix A, a polynomial time reformulation procedure can be obtained (see also Section 6.5.1). The practical drawback of this algorithm is its requirement of the introduction of additional variables and constraints.

In general, to find all minimal covers of a given inequality, implicit enumeration of all assignments to the variables occurring in the inequality is required. This can be done by setting up a search tree in the usual way; at each node a variable x_i is set to one in its left branch and to zero in its right branch. First sorting the coefficients in descending order, the search tree can be kept relatively small by choosing variable x_i as branching variable at depth $i+1$. A branch is closed when the partial assignment is such that the constraint is violated. If a branch remains open, this can be used to prune the search tree.

We conclude this section with observing that in special cases by making use of techniques from Boolean algebra, the resolvent $\mathcal{P}(x)$ can be much simplified; we refer to Hammer [65]. In the next section we discuss two (non-exact) algorithms for obtaining global minimizers of (NCP). These algorithms involve continuous variables.

6.4 Two minimization algorithms

We describe two algorithms for solving nonconvex models of the form (NCP). We will be concentrating on the special case of Section 6.3.1. Let us stress that both algorithms can be adapted in a straightforward manner to be applied to general models of the form (NCP). Moreover, the algorithms can also be applied to models that do not stem from a specific binary application and/or do not feature the rounding property. For example, the algorithms have been used in the context of unsupervised neural network training (Trafalis et al. [126]).

Let us recall the model (NQP).

$$(\text{NQP}) \qquad \begin{array}{ll} \min & \frac{1}{2}x^T Q x \\ \text{s.t.} & Bx = d, \\ & 0 \leq x \leq e. \end{array}$$

We minimize (NQP) iteratively. In each step we attempt to find a new intermediate solution that is better than the previous one, until a minimum is reached. Thus a generic algorithm is the following.

Starting from a feasible point:

1. Find a descent direction for the objective function.
2. Find the minimum of the objective function along the descent direction to obtain the new iterate.
3. If the new iterate has an objective value smaller than one, then use the rounding procedure to solve the problem, else go to step 1.
4. If the new iterate has an objective value greater than one, while no improving direction can be found, the algorithm is trapped in a local minimum. Modify the problem in some way to avoid running in this local minimum again, and restart the process.

The crucial element of the algorithm is step 1; the computation of a descent direction. We will describe two methods to compute it. First we briefly discuss a potential reduction method, that is inspired by the method introduced by Karmarkar et al. [82, 88]. Full details on this method can be found in [135, 138]. This method makes use of second-order derivatives and is therefore computationally involved. The second method discussed uses only the gradient. It is inspired by the rounding procedure. The algorithm is related to the reduced gradient method (see for instance Bazaraa et al. [7]), but no use is made of basic solutions and by exploiting the special structure of the model we can gain in efficiency.

6.4.1 A potential reduction algorithm

To solve (NQP) we introduce a *logarithmic barrier* function

$$\psi(x) = \tfrac{1}{2}x^TQx - \mu\sum_{i=1}^{m}\log x_i(1-x_i), \tag{6.7}$$

where the barrier parameter μ is positive. Note that $\psi(x)$ consists of two parts. The first is the objective function of (NQP), the second part is the logarithmic barrier function, which is mainly important for numerical reasons. It also exhibits good empirical properties. For brevity we denote the constraints $0 \leq x \leq e$ (and possible additional constraints) by $Cx \leq c$. Without loss of generality, we assume that the equality constraints $Bx = d$ are eliminated, or relaxed to inequality constraints. For techniques to deal with the equality constraints, see [135].

The idea of the algorithm is that minimizing $\psi(x)$ under appropriate constraints is equivalent to solving (NQP). Consider the nonconvex minimization problem

$$(\mathrm{NQP}_\psi) \qquad \begin{array}{ll} \min & \psi(x) \\ \text{s.t.} & Cx \leq c. \end{array}$$

Since solving (NQP$_\psi$) is in general NP-complete, in each iteration it is approximated by a quadratic optimization problem over an ellipsoid, which can be solved in polynomial time. Let x^i be the current interior solution and let $z^i = c - Cx^i$ be its slack vector. Using the notation $Z = \text{diag}(z^i)$, the Hessian and gradient of ψ in x are

$$\begin{aligned} h_\psi &= Qx^i + \mu C^T Z^{-1} e; \\ H_\psi &= Q + \mu C^T Z^{-2} C. \end{aligned}$$

Denoting $\Delta x = x - x^i$, the second order Taylor expansion of ψ around x^i is given by:

$$\psi^{(2)}(x) = \tfrac{1}{2}(\Delta x)^T H_\psi \Delta x + h_\psi^T \Delta x + \psi(x^i).$$

As approximation of the polytope the *Dikin ellipsoid* [40]) is used, which is given by

$$\mathcal{E}(r) = \{\Delta x \in \mathbb{R}^m \mid (\Delta x)^T C^T Z^{-2} C \Delta x \leq r^2\},$$

where for $r < 1$ the ellipsoid is inscribed in the feasible region. Note that this is a different ellipsoid from the one we introduced in Section 2.4; that ellipsoid *circumscribes*, rather than inscribes, the feasible region. We are now ready to formulate the polynomially solvable subproblem:

$$(\text{NQP}_\mathcal{E}) \quad \begin{array}{rl} \min & \tfrac{1}{2}(\Delta x)^T H_\psi (\Delta x) + h_\psi^T (\Delta x) \\ \text{s.t.} & (\Delta x)^T C^T Z^{-2} C (\Delta x) \leq r^2. \end{array}$$

This problem is known in the literature as a *trust region* problem (see e.g. Sörensen [121]). The optimal solution Δx^* to $(\text{NQP}_\mathcal{E})$ is a descent direction of $\psi^{(2)}(x)$ from x^i. A vector Δx^* is an optimal solution of $(\text{NQP}_\mathcal{E})$ if and only if $\lambda \geq 0$ exists, such that:

$$(H_\psi + \lambda C^T Z^{-2} C)\Delta x^* = -h_\psi \tag{6.8}$$

$$\lambda \left((\Delta x^*)^T C^T Z^{-2} C (\Delta x^*) - r^2\right) = 0 \tag{6.9}$$

$$H_\psi + \lambda C^T Z^{-2} C \quad \text{is positive semidefinite.} \tag{6.10}$$

For a proof of the optimality conditions, see Karmarkar et al. [88]. The algorithm uses an effective search strategy to find a multiplier λ and corresponding solution Δx^*, such that the optimality conditions are satisfied and the radius r of the inscribed ellipsoid is acceptable. This method is closely related to Sörensen's algorithm. It may require verifying (6.10) and subsequently solving the linear system (6.8) repeatedly. After a descent direction has been obtained, the new iterate x^{i+1} is computed by applying a line search to find the minimum of the potential function along the descent direction.

To benefit from the computational effort required to compute the new iterate as much as possible, one or more, possibly problem-specific, rounding schemes are applied to obtain a (hopefully globally optimal) binary solution. This process is repeated until either the problem is solved, or a local minimum is reached.

6.4.2 A gradient descent method

This algorithm is inspired by the rounding property. It uses first-order derivatives only, whereas in the potential reduction algorithm also second-order derivatives are used. Again, let x^i denote the current feasible solution. Our aim is to produce a descent direction Δx. Consider:

$$(\mathrm{QP}_i) \quad \begin{array}{ll} \min & \frac{1}{2}(\Delta x)^T Q \Delta x + q^T \Delta x \\ \text{s.t.} & B\Delta x = 0 \\ & -x^i \leq \Delta x \leq e - x^i, \end{array}$$

where $q = Qx^i$ is the gradient of the objective function. Note that (QP_i) is in fact the linear relaxation of $(\mathrm{QP}\Delta)$ (see Section 6.3.1). Using decomposition and the rounding property, an approximately optimal solution to this problem can be computed in polynomial time. Considering only the variables with index in the support $\rho(b_k^T) = E_k$ of a given equality constraint k, an optimal solution can be constructed in a straightforward manner, since then the quadratic term in the objective function is automatically annihilated. We refer to this procedure as *local gradient search*; it performs the procedure to solve (LP_k) (see Section 6.3.1). It is summarized, in a slightly modified form, in Figure 6.1. The procedure `sort`, that is called in line 2, sorts the entries of the given vector in *increasing* order. In line 3 τ is set equal to the d_kth smallest element of q. Note that a (nonnegative) parameter ε is introduced in the fourth line. This parameter prevents that too many variables are set to zero at once, thus forcing more variables to positive values. Intuitively this prevents the algorithm from running into a local minimum too easily.

```
procedure local_gradient ($x^i, \varepsilon, k$)
    $q := Qx^i$;
    $q_{sort} :=$ sort $(q_j \mid j \in E_k)$;
    $\tau := q_{sort}(d_k)$;
    $S_k := \{j \in E_k \mid q_j \leq \tau + \varepsilon\}$;
    $y := x^i + \frac{d_k}{|S_k|}(e_{S_k} - x^i_{E_k})$;
return ($y$)
```

Figure 6.1: The `Local Gradient` algorithm

Another possibility is to do a *global gradient search*. Neglecting the quadratic term in (QP_i), an efficiently solvable linear programming problem is obtained:

$$(\mathrm{LP}_i) \quad \begin{array}{ll} \min & q^T \Delta x \\ \text{s.t.} & B\Delta x = 0 \\ & -x^i \leq \Delta x \leq e - x^i. \end{array}$$

Our approach is to solve (LP_i) and then to find the minimum of the objective function along the search direction Δx^*, where Δx^* denotes the optimal solution to (LP_i). Due to Assumptions 2-4 (see Section 6.3.1), this solution can be computed in strongly polynomial time. It is obtained by executing the procedure `local_gradient` for each $k = 1, \ldots, p$, with $\varepsilon = 0$. Note that if we give ε a (small) positive value, an approximate solution of (LP_i) is obtained that, again intuitively, may better suit our purposes. To some extent, ε performs the role of the barrier parameter μ. The function of both is to prevent the iterates from converging to a vertex too fast. Observe that one might also choose to use the gradient of the potential function $\psi(x)$ (6.7) as objective function in (LP$_i$). This has no influence on the efficiency with which a descent direction can be computed.

The objective function along the search direction $x^i + \Delta x^*$ as a function of α is can be convex or concave. By simple calculations we compute the optimal value for α:

$$\theta(\alpha) = (x^i + \alpha\Delta x^*)^T Q(x^i + \alpha\Delta x^*) = ((\Delta x^*)^T Q\Delta x^*)\alpha^2 + 2(q^T\Delta x^*)\alpha + \text{constant}.$$

Therefore if $(\Delta x^*)^T Q\Delta x^* \leq 0$ (i.e. $\theta(\alpha)$ is concave), we take α equal to one, since that is the maximum step size maintaining feasibility. Otherwise, if $(\Delta x^*)^T Q\Delta x^* > 0$ we take

$$\alpha := \min\left\{1, -\frac{q^T\Delta x^*}{(\Delta x^*)^T Q\Delta x^*}\right\}.$$

It is easily understood that $\alpha \geq 0$ since $q^T\Delta x^* \leq 0$.

If the gradient descent algorithm is employed to minimize the potential function ψ, one has to be careful in computing the optimal step length. Either the above sketched analytic approach may be used, with appropriate measures taken to ensure that the iterates stay in the interior of the feasible region, or a line search must be carried out to obtain a good approximation of the optimal step length.

The gradient descent algorithm runs as follows. Starting from a feasible point, we apply the global gradient search until no sufficiently improving direction can be found. During this process the parameter ε must be decreased to zero. Subsequently, we switch to the local gradient search, until either the problem is solved or a minimum is reached. In the latter case, the fractional solution is rounded to a binary solution, using the rounding property.

6.4.3 Local minima

There are several heuristics available to deal with local minima; see also [135]. In the computational results reported in this chapter, we use the following approach. Let x^k be a local minimizer of the problem under consideration. The idea is to determine all constraints that are violated by $\lceil x^k \rceil$, and to increase the corresponding weights. Let

$$\mathcal{V} = \{(i,j) \mid (i,j) \in \mathcal{M} \text{ with } x_i^k x_j^k > 0\},$$

and modify the objective function of the problem (NQP):

$$\min \quad N \sum_{(i,j)\in\mathcal{V}} x_i x_j + \sum_{(i,j)\in\mathcal{M}\setminus\mathcal{V}} x_i x_j,$$

where N is a large number. Substituting the current locally optimal solution in this modified objective function, the objective value will now be increased to N times the old one. If N is sufficiently large the current solution is no longer a local minimum. The minimization procedure can be continued until another minimum is found. Observe that this procedure may be regarded as a *local search* procedure. In a sense it is assumed that the current solution is close to an optimal solution, and the algorithm will subsequently converge to solutions that are 'near' the current solution. Note that, if at some point in the process the algorithm gets stuck in a region where there appear to be many local minima, one might decide to restart the algorithm from a new starting point.

6.5 Two specific applications

In this section we discuss two specific applications of the reformulation technique described in Section 6.3. It may be stressed that even though any problem of the form (BFP) can be transformed to a problem of the form (NCP), in general this will be worthwhile only if all constraints involved have one or at most few minimal covers. Moreover, the model is particularly suited to solve *feasibility* problems rather than problems in which some linear objective function needs to be optimized. In the next subsection Satisfiability (SAT) problems are considered and in the subsection thereafter we turn our attention to the Frequency Assignment Problem (FAP).

6.5.1 The satisfiability problem

Let us consider the satisfiability problem in CNF as introduced in Chapter 1. It is easy to see that using the notion of minimal covers a multilinear model for the SAT problem is derived that is equivalent to the unweighted variant of the model (WPR) used by among others Gu [61]; see also the Sections 2.3 and 4.4.2. Considering the linear inequality associated with a clause, it is easy to verify that it has maximum violation $\nu = 1$, hence Lemmas 6.3.15 and 6.3.17 and Theorem 6.3.18 apply. Thus we have the following implication. Let $0 \leq x \leq e$, then

$$P_k(x) = \prod_{i \in I_k} (1 - x_i) \prod_{j \in J_k} x_i = 0$$

if and only if $\mathbf{C}_k$ is satisfied by $[x]$. As each inequality has exactly one minimal cover, it follows that for a given CNF formula one can straightforwardly construct a nonconvex minimization problem as described in Section 6.3.2. As pointed out there, for a given truth value assignment x, $\mathcal{P}(x)$ is equal to the number of unsatisfied clauses. We give an example.

Example 6.5.1 For a specific class of contradictory formulas, the resolvent $\mathcal{P}(x)$ has the interesting property that it is exactly equal to one. It is therefore immediately clear that these formulas are not satisfiable, and that each truth value assignment satisfies all except one clause. Let the formula F_ℓ be the formula containing all 2^ℓ distinct clauses on ℓ variables that can be constructed using the propositions $p_1, \ldots, p_\ell$. Clearly, F_ℓ is not satisfiable. Now we show that the polynomial $\mathcal{P}_\ell(x)$ associated with F_ℓ is equal to one, by induction on ℓ. For $\ell = 1$ we have $F_1 = p_1 \wedge \neg p_1$, which implies that $\mathcal{P}_1(x) = (1 - x_1) + x_1 = 1$. Now assume that the claim holds for ℓ. Consider $F_{\ell+1}$. Clearly, we have

$$F_{\ell+1} = (p_{\ell+1} \vee F_\ell) \wedge (\neg p_{\ell+1} \vee F_\ell).$$

This implies that

$$\mathcal{P}_{\ell+1}(x) = (1 - x_{\ell+1})\mathcal{P}_\ell(x) + x_{\ell+1}\mathcal{P}_\ell(x).$$

Since $\mathcal{P}_\ell(x) \equiv 1$ we find that also $\mathcal{P}_{\ell+1}(x) \equiv 1$. In fact this is an example of formulas having positive polynomial representation (PPR) as introduced in Chapter 4. □

Gu [61] proposes several algorithms for solving the 'global optimization version' of the satisfiability problem. One of these algorithms in fact uses the rounding procedure. The function $\mathcal{P}(x)$ is iteratively minimized by in each iteration choosing a variable and setting this to either 0 or 1, according to which gives the biggest improvement in terms of objective value. Note that this algorithm may be considered as a 'branching rule'; thus the global optimization algorithm is a branching algorithm (without backtracking). Backtracking can be incorporated to obtain a complete algorithm. The algorithms Gu propose prove to be quite effective. SAT instances up to a size of 50000 clauses and 5000 variables are solved in a matter of (fractions of) seconds [61]. Interestingly, Gu also compares his results with those obtained by Kamath et al. [82] who use the second-order potential reduction approach that is described in Section 6.4.1 to minimize the function $x^T(e-x)$ subject to the linear constraints associated with the clauses. It appears that this method is outperformed by Gu's algorithms.

Another interesting observation is that the approximation algorithm for maximum satisfiability problems proposed by Johnson [78] follows directly from the rounding property (this was earlier observed in [16, 140]). Using the polynomial representation of a pure ℓSAT formula (i.e. all clauses have length exactly ℓ) with n clauses and substituting $x = \frac{1}{2}e$, this solution has an objective value equal to $n2^{-\ell}$ (this is also the expected value of a random solution in which each variable is set to 1 with probability $\frac{1}{2}$ and to 0 with probability $\frac{1}{2}$). Via the rounding procedure a binary solution with objective value smaller than or equal to $n2^{-\ell}$ is obtained, implying that at least $(1-2^{-\ell})n$ clauses are satisfied. This is the same bound as Johnson obtains. In fact the algorithms coincide although Johnsons does not make (explicit) use of a polynomial representation of the satisfiability problem. For general maximum satisfiability problems (i.e. unit clauses may be present), the approximation ratio is $\frac{1}{2}$. Goemans and Williamson [55] have obtained $\frac{3}{4}$ approximation algorithms making use of the polynomial representation in conjunction with randomized rounding of the solution of a linear programming relaxation of MAX-SAT. Their result has been improved upon by Asano [5] who has obtained a 0.77 approximation algorithm.

Incidentally, it has been shown recently for pure 3SAT that no polynomial time algorithm with a better performance guarantee than Johnson's algorithm exists (unless $\mathcal{P} = \mathcal{NP}$) [68]; i.e. no efficient algorithm can approximate any given instance of MAX3SAT within a ratio of $\frac{7}{8}$. As pointed out in Section 5.8 Karloff and Zwick obtained an optimal approximation algorithm for MAX3SAT. Thus there exist $\frac{7}{8}$ approximation algorithms both for instances in which all clauses have length three or less and for instances in which all clauses have length three or more. Whether these approximation algorithms can be combined to obtain a $\frac{7}{8}$ approximation algorithm for MAX SAT in general is still an open question.

6.5.2 The frequency assignment problem

As a second application, we consider the Frequency Assignment Problem (FAP) which is described in Section 1.3.3. We consider one particular class of the FAP:

given a set of frequencies $\mathcal{F}$, a set of links $\mathcal{L}$ and a set of interference constraints, assign a frequency $f \in \mathcal{F}$ to each link $l \in \mathcal{L}$ such that all the interference constraints are satisfied, and the number of distinct frequencies used is minimal.

In order to model the FAP, we introduce some additional notation.

$\mathcal{F}_l$	:	frequency domain of link $l \in \mathcal{L}$ ($\mathcal{F}_l \subseteq \mathcal{F}$);
$f \hookrightarrow l$	:	the assignment of frequency $f \in \mathcal{F}_l$ to link $l \in \mathcal{L}$;
D	:	matrix indicating the required frequency distance for any two links.

If $f \hookrightarrow l$ and $g \hookrightarrow k$, then it must hold that $|f-g| \geq d_{lk}$. Hence, if $d_{lk} > 0$, then links l and k can not be assigned the same frequency.

Let us point out that the graph coloring problem, a well known NP-complete problem, can be solved as an FAP, implying that the FAP is NP-complete. Given the problem of colouring the nodes of a graph $G = (V, E)$ this can be reduced to an instance of the FAP by letting $\mathcal{L} \equiv V$ and defining $d_{lk} = 1$ if $(l,k) \in E$; otherwise $d_{lk} = 0$. The minimal number of frequencies required to construct an interference-free assignment is equal to the minimal number of colours required to colour the nodes of G. The FAP is more complicated than the graph coloring problem in the sense that an interference constraint does not just express that a pair of links must be assigned different frequencies, but it also specifies a minimal required distance.

We have the following decision variables:

$$\begin{aligned} x_{lf} &= \begin{cases} 1 & \text{if } f \hookrightarrow l, \\ 0 & \text{otherwise,} \end{cases} \qquad l \in \mathcal{L},\ f \in \mathcal{F}_l. \\ z_f &= \begin{cases} 1 & \text{if the frequency } f \text{ is assigned to at least one link,} \\ 0 & \text{otherwise,} \end{cases} \qquad f \in \mathcal{F}. \end{aligned}$$

Thus, with reference to Section 1.3.3, the x_{lf} variables are associated with the propositional variables p_{lf}, the z_f variables with the q_f. We can straightforwardly associate a linear model with the SAT encoding. First, we need to assign a frequency to each link.

$$e_{l\mathcal{F}_l}^T x = 1, \qquad l \in \mathcal{L}. \tag{6.11}$$

Note that a linear inequality associated with a clause should have an inequality sign rather than an equality sign, but taking its interpretation into account it is obvious that equality suffices. The interference constraints are as follows:

$$x_{lf} + x_{kg} \leq 1,\ (l,k) \in \mathcal{L}, f \in \mathcal{F}_l, g \in \mathcal{F}_k, \text{ such that } |f-g| < d_{lk}. \tag{6.12}$$

Denoting the equality constraints (6.11) by $Bx = e$ and the inequality constraints (6.12) by $Ax \leq e$, the FAP can be expressed as the $\{0,1\}$ feasibility problem to find $x \in \{0,1\}^m$ such that $Ax \leq e$ and $Bx = e$.

The model can be extended to restrict the number of frequencies used. To this end, we add the variables z_f to the model and the constraint $e^T z = F_{\max}$, where $F_{\max}$ specifies the number of frequencies to be used. Furthermore, we add the following constraints to the model:

$$x_{lf} - z_f \leq 0, \qquad l \in \mathcal{L},\ f \in \mathcal{F}_l. \tag{6.13}$$

Now let the set of linear inequalities (6.13) be denoted by $x - R^T z \leq 0$. The extended feasibility problem is expressed as

$$\text{find } (x,\ z)^T \in \{0,1\}^{m+|\mathcal{F}|} \text{ such that } \quad \begin{cases} Ax \leq e \\ x - R^T z \leq 0 \\ Bx = e \\ e^T z = F_{\max}. \end{cases}$$

Note that this problem no longer can be directly transformed to a SAT problem. The bound on the number of frequencies to use disturbs the required structure. However, the Assumptions 1-4 (Section 6.3.1) are satisfied by this model, thus enabling us to use formula (6.1) to obtain a nonconvex quadratic model for the FAP. To verify this, note that $x - R^T z \leq 0$ can be rewritten as $x + R^T(e - z) \leq R^T e = e$, where we use that R contains exactly one nonzero element in each column.

Let us now consider Corollary 6.3.10 and apply it to the FAP, in other words, let us determine the quality guarantee of the rounding procedure. To this end we take a closer look at the number of constraints present; the upper bound on the objective value that can be obtained using the rounding property is related to these numbers. We denote the number of interference constraints by $|d|$, i.e. $|d|$ is the number of pairs of links for which $d_{lk} > 0$. Assuming that all $|\mathcal{F}|$ frequencies are available for each link, these constraints are modelled using $\gamma|\mathcal{F}||d|$ constraints of type (6.12), where the value $1 \leq \gamma \leq |\mathcal{F}|$ depends on the average required frequency distance. For $\gamma = 1$, the FAP reduces to a graph colouring problem; for $\gamma = |\mathcal{F}|$ none of the interference constraints can be satisfied. The number of constraints of type (6.13) is equal to $|\mathcal{L}||\mathcal{F}|$.

For the moment, we consider only the feasibility version of the FAP (i.e. without the z_f variables). By initially setting $x_i = 1/|\mathcal{F}|$ for all i, one can obtain a binary solution that satisfies at least a fraction $1-(1/|\mathcal{F}|)^2$ of the total number of constraints. (In the notation of Section 6.3.1 (see eq. 6.3), $\kappa = 1$, $m/p = |\mathcal{F}|$.) To get a better understanding of its implications, this performance guarantee should be related to the number of interference constraints $|d|$; thus we find that at least $(1 - \gamma/|\mathcal{F}|)|d|$ interference constraints can be satisfied in polynomial time. For example, if $\gamma/|\mathcal{F}| \approx .1$ (which is the case for the large benchmarks solved in the next section), then it is guaranteed that at least about 90% of the interference constraints are satisfied in polynomial time. In practice, the solutions obtained using the rounding procedure may be, and frequently are, even better.

If we also take into account the bound on the number of frequencies to be used, substituting the average solution $(x,\ z)$ gives the objective value $(1/|\mathcal{F}|)(\gamma|d| + |\mathcal{L}|F_{\max})$. To give a flavour of the implications, assuming that $F_{\max} = \frac{1}{2}|\mathcal{F}|$, $|d| = 5|\mathcal{L}|$, solutions can be constructed that satisfy at least 80% of the interference constraints.

To illustrate the rounding property and algorithms of Section 6.4 let us consider an example. In the next section we turn to computational results on larger instances of the FAP.

Example 6.5.2 Let us again consider the instance depicted in Figure 1.3 in Chapter 1. It is defined by

$$\mathcal{L} = \{1,2,3,4\} \text{ and } \mathcal{F} = \{2,4,7,9\};$$

$$\mathcal{F}_1 = \mathcal{F}_3 = \{2,4,9\},\ \mathcal{F}_2 = \mathcal{F}_4 = \mathcal{F}.$$

The required frequency distances d_{lk} are given by the following matrix:

$$D = \begin{pmatrix} - & 4 & 6 & 1 \\ & - & 0 & 4 \\ & & - & 0 \\ & & & - \end{pmatrix}.$$

The number of constraints of type (6.12) is 27, while the number of constraints of type (6.13) is 14. We do not take the latter into account in this example. Let us apply the algorithms discussed in Section 6.4 to this (very small) instance. We take as initial solution $x^0_{lf} = 1/|\mathcal{F}_l|$, for all $l \in \mathcal{L}, f \in \mathcal{F}_l$. The objective value of x^0 is 161/72, implying that an assignment can be constructed that violates at most 2 constraints.

- The potential reduction algorithm (Section 6.4.1) finds after two iterations the fractional solution $x^2 = (.91\ .02\ .08\ ;\ .20\ .00\ .12\ .70\ ;\ .22\ .02\ .76\ ;\ .06\ .48\ .38\ .09)$, with potential value 1.64. Here the ';' separates the variables corresponding to different links. By, for each link, setting the variable with maximal value to 1, the feasible assignment $(2 \hookrightarrow 1, 9 \hookrightarrow 2, 9 \hookrightarrow 3, 4 \hookrightarrow 4)$ is obtained. If no such rounding scheme is applied, in the third iteration a solution x^3 is found with potential value .87, guaranteeing the existence of a feasible solution (by the rounding property). The algorithm needs another 7 iterations to find a (near-binary) solution with potential value less than 10^{-3}.

- The gradient descent algorithm (Section 6.4.2) finds a global optimum after three iterations. The parameter ϵ is taken to be equal to zero. First two 'convex steps' are made (step sizes 29/41 and 43/116). The objective value of the intermediate solution x^2 is 1.66. Subsequently the algorithm takes a 'concave step' (i.e. step size 1) to the global optimum which yields the feasible assignment that was also found using the potential reduction method. Thus, the instance is completely solved by the *global gradient search* and there is no need to switch to the local variant.

- Let us now apply the rounding procedure to the initial solution x^0. The reader may want to verify that by subsequently rounding the variables for links 1, 2, 3 and 4, we arrive via the intermediate objective values 49/24, 17/12 and 3/4, at either the feasible assignment given above or the (also feasible) assignment $(9 \hookrightarrow 1, 2 \hookrightarrow 2, 2 \hookrightarrow 3, 7 \hookrightarrow 4)$. It is interesting to note that by rounding the variables in reverse order, an infeasible assignment might be obtained. This concludes the example. □

6.5.3 Computational results

The aim of this section is to provide an indication of the effectiveness and robustness of the proposed algorithms. To this end, we report on some computational experience on the FAP. In Table 6.1 the characteristics of the instances considered are given. See for more information on these instances the benchmark overview paper [21]. CELAR (Centre d'ELectrique d'ARmement) provided a number of test instances. All other instances were generated using the TU Delft developed test problem generator *GRAPH* (van Benthem [10]). All the tests were run on a HP9000/720 work station, 150 mHz. The implementation of the potential reduction algorithm is the same as used in [135, 137, 138]; it is implemented in MATLAB™ and it uses a number of FORTRAN routines provided by the linear programming interior point solver LIPSOL [142]. The same parameter settings and rounding schemes as in the above mentioned papers were employed.

The gradient descent algorithm is implemented in MATLAB™ as well. Some preliminary experience with applying it to minimize potential function $\psi(x)$ (eq. (6.7)) indicated that thus no better results are obtained than when using the 'plain' gradient (i.e. without

name	$\lvert\mathcal{L}\rvert$	$\lvert d\rvert$	# frq.	# vars.	name	$\lvert\mathcal{L}\rvert$	$\lvert d\rvert$	# frq.	# vars.
G16.6	16	28	6	232	TUD200.1	200	1071	12	4015
G20.6	20	36	6	411	TUD200.3	200	1060	*18	3449
G20.10	20	80	10	296	GRAPH01	200	1034	18	3484
G24.6	24	74	6	518	GRAPH02	400	2045	14	7336
G26.8	26	72	8	387	GRAPH08	680	3417	*20	12838
G30.8	30	84	8	562	GRAPH09	916	4789	16	18070
G36.12	36	204	12	530	GRAPH14	916	4180	*10	18382
G40.8	40	170	8	651	CELAR01	916	5090	16	18124
G50.12	50	240	12	592	CELAR02	200	1135	14	4026
G100.12	100	502	10	1078	CELAR03	400	2560	14	7970

Table 6.1: Characteristics of the instances. Given are, for each instance, its name, its number of links and interference constraints, its optimal (* best known) solution and the number of variables required to model it. The number of frequencies available for each link is typically 44.

barrier). Therefore we use the plain objective function. In the implementation, the parameter ε is initially set to .25, and after each *global gradient search* it is decreased by a factor 2. In the *local gradient search* it is set to zero. The algorithm switches from global to local gradient search when the first improves the objective value by less than .1.

The computational tests were carried out as follows. The algorithm was provided with the value of the best known solution (i.e. the optimal value reported in Table 6.1), and ran until either a solution with the desired number of frequencies was found, or the number of local minima encountered exceeded some preset maximum $K_{\max}$. The value of $K_{\max}$ was set to $K_{\max} = 20$. A local minimum was dealt with as described in Section 6.4.3, with the value of N set to 100.

Each of the 'smaller' instances (G*.*) was solved with both algorithms, using 20 different starting points. The results of these runs are given in Table 6.2. The success of the algorithms is to some extent dependent on the quality of the starting point. The percentages of the total number of runs in which an optimal solution was found are given ('Succespercentage'). The minimal, average and maximal solution times are also reported. It appears that the gradient descent algorithm is at least as efficient as the potential reduction algorithm and certainly faster. Even though one might expect the second-order derivatives to contain more information than the first-order, the gradient algorithm also seems to be more robust. The facts that the gradient algorithm consumes less computer memory and its execution is faster allow us to solve larger instances than using the potential reduction algorithm. This, combined with its excellent performance on the smaller problems, justifies us to restrict ourselves to using the gradient descent algorithm to solve the larger problems; the reader is referred to [135, 138] for results on the larger problems using the potential reduction algorithm. In these papers further preprocessing methods are discussed, from which the potential reduction algorithm greatly benefits. Using the gradient algorithm, we succeeded in solving all the instances; the results are reported in

	solved		min		mean		max	
name	pr	grad.	pr	grad.	pr	grad.	pr	grad.
G16.6	95%	95%	1.3	0.4	43.5	2.7	236.0	6.0
G20.10	90%	85%	3.0	0.8	6.6	9.3	13.6	32.2
G26.8	60%	85%	6.6	0.8	109.6	7.3	714.3	21.7
G20.6	100%	90%	4.0	0.7	42.1	5.1	128.0	14.3
G24.6	100%	100%	6.5	0.8	14.6	1.8	37.0	4.2
G36.12	85%	80%	70.6	1.1	96.3	10.6	153.7	41.2
G30.8	65%	70%	6.4	4.0	21.5	12.4	62.3	29.4
G40.8	90%	100%	48.0	1.6	151.5	7.1	910.2	36.6
G50.12	10%	35%	325.2	1.4	769.9	13.0	1214.6	21.9
G100.12.	15%	55%	73.6	13.6	251.1	59.0	535.8	201.1

Table 6.2: 'Success-percentage', and minimal, average and maximal solution times for the potential reduction algorithm (pr) and the gradient descent algorithm (grad.).

Table 6.3. In a small number of cases we improved on the best known solutions reported in [125], which were at the time the best known solutions. These are indicated with an *.

6.6 Concluding remarks

As the computational results reported in the previous section indicate, the gradient descent algorithm is capable of solving large frequency assignment problems (containing over 18000 variables) within the hour on a HP9000/720 work station. These results are obtained using an experimental MATLAB™ implementation. Recently, an efficient implementation became available (Pasechnik [106]). Computation times using this implementation are 10-200 shorter; see Table 6.4. Comparing the results to those in the overview paper by Tiourine et al. [125] it appears that the gradient descent algorithm outperforms the fastest methods mentioned there, which are taboo search and simulated annealing. More importantly, these methods do not always find optimal solutions, while the gradient method does. In the same paper some comments on the effectiveness of the various approaches are made. Although comparison is hard since different hardware and different programming languages are used, some tentative conclusions are drawn. These rank the potential reduction method along with well known approximation algorithms such as simulated annealing, local search and genetic algorithms. Hence we conclude that the gradient descent algorithm introduced here is a valuable tool for solving hard combinatorial optimization problems as well.

name	solved	min	mean	max
TUD200.3 (18)	85%	37	150	312
TUD200.3 (16*)	60%	65	182	363
TUD200.3 (14*)	25%	84	233	418
GRAPH01	100%	10	67	287
TUD200.1	85%	30	273	591
CELAR02	100%	15	48	115
GRAPH02	95%	26	168	769
CELAR03	15%	465	891	1304
GRAPH08 (20)	100%	266	863	1647
GRAPH08 (18*)	100%	361	481	716
GRAPH09	70%	606	1503	2551
CELAR01	20%	1733	2468	3203
GRAPH14	100%	93	753	1920

Table 6.3: 'Success-percentage', and minimal, average and maximal solution times for the gradient method on the larger test instances.

name	time
GRAPH01	0.60
CELAR02	2.25
GRAPH02	4.82
CELAR03	4.37
GRAPH08	7.29
GRAPH09	5.55
CELAR01	16.98
GRAPH14	3.13

Table 6.4: Solution times using Pasechnik's implementation of the gradient method. Table taken from [106].

Appendix A

Representing linear inequalities by CNF formulas

We consider the construction of a propositional formula whose satisfying solutions correspond to feasible solutions of the inequality

$$a^T x = \sum_{i=1}^{m} a_i x_i \leq a_0,$$

where without loss of generality we assume that $a \geq 0$. Let us first give a brief outline of the idea of the transformation, which is essentially quite simple. On the lowest level, propositions and clauses are added to represent the single terms $a_i x_i$, using the binary representations of the a_i. The required logical expressions are derived from binary multiplication, and their CNF translations are added to the set of clauses. Then, on the next level, propositions are added to represent the sum (in binary notation) of disjoint pairs of terms, and logical expressions derived from binary addition are added to the set of clauses. On the subsequent level sums of, again disjoint, *pairs of pairs* of terms are represented in a similar way. This process is repeated until the top level is reached where a set of propositions is added that represents the sum, in binary notation, over all the terms of the inequality under consideration. During the process the logical expressions on a certain level operate exclusively on the propositions introduced on that level and those introduced one level lower. Finally, logical expressions that operate on the top level propositions are derived to ensure that the right hand side a_0 of the inequality is not exceeded. To this end the binary representation of a_0 is used. For example, let $a_0 = 26$ and denote the top level proposition corresponding to bit $i+1$ (i.e. the bit that contributes 2^i to the sum) by p_i. We define $\mathcal{B}(a_0)$ to be the set containing all powers of two that contribute to the binary representation of a_0. Hence $\mathcal{B}(26) = \{1, 3, 4\}$ and it follows that $\neg p_i$ for $i \geq 5$, $p_2 \rightarrow \neg(p_3 \wedge p_4)$ and $p_0 \rightarrow \neg(p_1 \wedge p_3 \wedge p_4)$.

The resulting formula is satisfiable if and only if the original inequality allows a feasible binary solution. A satisfiable assignment restricted to the propositions associated with the original x_i variables yields a feasible binary solution of the inequality.

Below a formal recursive definition of the construction is given. Subsequently, we discuss it in more detail and show that its complexity is linear. Let us introduce some notation. By $a_{\max}$ we denote the maximum entry of the vector a. Furthermore, we let M be such that $2^{M-1} \leq a_{\max} < M$ or equivalently, $M \leq 1 +^2 \log(a_{\max})$. We associate propositions

p_{x_i} with the (binary) variables x_i, $i = 1, \dots, m$, and introduce propositions $\overline{p}_j^{(i)}$ for all $j = 1, \dots, M$, $i = 1, \dots, m$. In the following I is a set of indices. We use the sets $\lfloor I \rfloor \subseteq I$ and $\lceil I \rceil \subset I$ for which the following holds:

$$\lfloor I \rfloor \cup \lceil I \rceil = I;\ \lfloor I \rfloor \cap \lceil I \rceil = \emptyset;\ |\lfloor I \rfloor| \geq |\lceil I \rceil|;\ |\lfloor I \rfloor| - |\lceil I \rceil| \leq 1.$$

The sets $\lfloor I \rfloor$ and $\lceil I \rceil$ are a partition of I. The propositions representing the sum $\sum_{i \in I} a_i x_i$, are denoted by $\{p_j^{(I)}\}$ where $j = 0, 1, \dots, M_I$, with $M_I = M + \log(|I|)$.

The transform of a linear inequality $a^T x \leq a_0$ is given by

$$trans(a^T x \leq a_0) = trans(a^T x) \wedge trans(\leq a_0).$$

The transformation of the sum of $a_i x_i$ over the index set $I \neq \emptyset$ is recursively defined as

$$trans(\sum_{i \in I} a_i x_i) = \begin{cases} trans(\sum_{i \in \lfloor I \rfloor} a_i x_i) \wedge trans(\sum_{i \in \lceil I \rceil} a_i x_i) \wedge \mathcal{T}^+(\{p_j^{(I)}\}, \{p_j^{(\lfloor I \rfloor)}\}, \{p_j^{(\lceil I \rceil)}\}), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{if } |I| > 1; \\ trans(a_i) \wedge \mathcal{T}^*(\{p_j^{(i)}\}, \{\overline{p}_j^{(i)}\}, p_{x_i}), \text{ if } I = \{i\}. \end{cases}$$

Here the operator $\mathcal{T}^+(\{p_j^{(U)}\}, \{p_j^{(V)}\}, \{p_j^{(W)}\})$ performs the *addition* of two numbers in binary notation $p_j^{(V)}$ and $p_j^{(W)}$, yielding the sum $p_j^{(U)}$. The operator $\mathcal{T}^*(\{p_j^{(i)}\}, \{\overline{p}_j^{(i)}\}, p_{x_i})$ performs the *multiplication* of a number in binary notation $\overline{p}_j^{(i)}$ with a binary number p_{x_i}. Later on we give explicit expressions for these operators. Furthermore, we have that

$$trans(a_i) = \bigwedge_{j \in \mathcal{B}(a_i)} \overline{p}_j^{(i)} \wedge \bigwedge_{j \notin \mathcal{B}(a_i)} \neg \overline{p}_j^{(i)}. \tag{A.1}$$

Finally, let us define $\mathcal{M} = \{1, \dots, m\}$. To process the right hand side a_0, we define

$$trans(\leq a_0) = \bigwedge_{j \notin \mathcal{B}(a_0)} \left(p_j^{(\mathcal{M})} \rightarrow \neg \bigwedge_{k \in \mathcal{B}(a_0): k > j} p_k^{(\mathcal{M})} \right).$$

We have the following theorem.

Theorem A.1 *Assuming that $a_{\max}$ is a priori bounded, the complexity of the transformation introduced above is linear in the length of the inequality.*

To prove the theorem, let us take a closer look at the number of additional variables and clauses we need to introduce to perform the transformation. To this end, we first specify the number of additional variables and clauses the binary multiplication and addition require. The following figure explains how to obtain the transformation operator $\mathcal{T}^+(\cdot)$. Let $U = V \cup W$.

$$\begin{array}{ccccccc} & p_{M_U-1}^{(V)} & p_{M_U-2}^{(V)} & \cdots & p_2^{(V)} & p_1^{(V)} & p_0^{(V)} \\ & p_{M_U-1}^{(W)} & p_{M_U-2}^{(W)} & \cdots & p_2^{(W)} & p_1^{(W)} & p_0^{(W)} & + \\ \hline p_{M_U}^{(U)} & p_{M_U-1}^{(U)} & p_{M_U-2}^{(U)} & \cdots & p_2^{(U)} & p_1^{(U)} & p_0^{(U)} \end{array} \tag{A.2}$$

For example, we have that $p_0^{(U)}$ is *true* if and only if *exactly one* of the propositions $p_0^{(V)}$ and $p_0^{(W)}$ is *true*. This is expressed by (A.3). Thus the transformation operator $\mathcal{T}^+(\{p_j^{(U)}\},\{p_j^{(V)}\},\{p_j^{(W)}\})$ with $U = V \cup W$, V and W nonempty, can be expressed as (using *carry* propositions $q_{j,j+1}$)

$$\left(p_0^{(U)} \leftrightarrow (p_0^{(V)} \leftrightarrow \neg p_0^{(W)})\right) \wedge \tag{A.3}$$

$$\left(q_{01}^{(U)} \leftrightarrow (p_0^{(V)} \wedge p_0^{(W)})\right) \wedge \tag{A.4}$$

$$\bigwedge_{j=1,\dots,M_U} \left(p_j^{(U)} \leftrightarrow (p_j^{(V)} \leftrightarrow p_j^{(W)} \leftrightarrow q_{j-1,j}^{(U)})\right) \wedge \tag{A.5}$$

$$\bigwedge_{j=1,\dots,M_U-1} \left(q_{j,j+1}^{(U)} \leftrightarrow \left((p_j^{(V)} \wedge p_j^{(W)}) \vee (p_j^{(V)} \wedge q_{j-1,j}^{(U)}) \vee (p_j^{(W)} \wedge q_{j-1,j}^{(U)})\right)\right) \wedge \tag{A.6}$$

$$\left(q_{M_U-1,M_U}^{(U)} \leftrightarrow p_{M_U}^{(U)}\right). \tag{A.7}$$

The logical expressions (A.3-A.7) can be readily rewritten to CNF formulas using De Morgan's laws. Then, taking $N = \max\{|V|,|W|\}$, $2N$ new variables and at most $14N-7$ clauses are introduced.

The transformation operator $\mathcal{T}^*(\cdot)$ is obtained in a similar way, yielding

$$\mathcal{T}^*(\{p_j^{(i)}\},\{\overline{p}_j^{(i)}\},p_{x_i}) = \bigwedge_{j=0}^{M_i} \left(p_j^{(i)} \leftrightarrow (\overline{p}_j^{(i)} \wedge p_{x_i})\right). \tag{A.8}$$

Note that, using unit resolution, this expression in conjunction with (A.1) reduces to

$$\bigwedge_{j\in\mathcal{B}(a_i)} \left(p_j^{(i)} \leftrightarrow p_{x_i}\right) \wedge \bigwedge_{j\notin\mathcal{B}(a_i)} \neg p_j^{(i)}.$$

From this we see that the propositions $p_j^{(i)}$, $j \in \mathcal{B}(a_i)$, can be eliminated by substituting them by p_{x_i}. Thus in the computations below these are not counted. Now let us denote by var_m and cl_m the numbers of additional variables and clauses to transform an inequality of length m. The following lemma bounds these numbers. Observe that the lemma implies Theorem A.1.

Lemma A.2 *We have the following upper bounds on var_m and cl_m.*

$$\begin{aligned} var_m &\leq 2m(1+\log(a_{\max})); \\ cl_m &\leq 8m(1+2\log(a_{\max})). \end{aligned}$$

Proof: If $m = 2^r$ for some natural number r the required number of variables and clauses can easily be computed:

$$var_{2^r} = \sum_{i=1}^{r} 2^{r-i}(2(M+i-1)) = 2(2^r(M+1)-(M+r+1)); \tag{A.9}$$

$$cl_{2^r} = \sum_{i=1}^{r} 2^{r-i}(14(M+i-1)-7) = 7(2^r(2M+1)-2(M+r)-1). \tag{A.10}$$

To compute var_m for arbitrary m, we use the following recursive formula:

$$var_m = var_{2^k} + var_{m-2^k} + 2\ell(var_{2^k}), \tag{A.11}$$

where $k = \max(\mathcal{B}(m))$ and $\ell(var_{2^k})$ denotes the length of the binary representation of var_{2^k}. By construction, we have that $\ell(var_{2^k}) = M + k$. Substituting this and (A.9) in (A.11) we obtain

$$var_m = 2^{k+2} + 2^{k+1}(M-1) - 2 + var_{m-2^k}.$$

Using that $\mathcal{B}(m - 2^k) = \mathcal{B}(m)\backslash\{k\}$, we derive the following upper bound on var_m:

$$var_m \leq 2[m(M+1) - (M + |\mathcal{B}(m)| + \min(\mathcal{B}(m))].$$

In a similar manner an upper bound on the number of clauses can be derived:

$$cl_m \leq 7[m(2M+1) - 2(M + |\mathcal{B}(m)| + \min(\mathcal{B}(m))) + 1] + M_{\mathcal{M}} - |\mathcal{B}(a_0)|,$$

where the term $M_{\mathcal{M}} - |\mathcal{B}(a_0)|$ is the number of clauses required to process the right hand side. The bounds stated in the lemma follow easily. □

Using specific problem–dependent structures, a number of redundant additional variables and clauses can usually be directly eliminated; we refer to [136].

Binary linear programs with constant objective function can be transformed to CNF by applying the procedure described above to each of the inequalities separately. Note that it is important to keep track of negative coefficients: if a variable x_i has a negative coefficient, then its associated proposition p_{x_i} must be negated in (A.8). Binary LPs with non-constant objective function can be transformed to CNF by adding a bound on the objective function and treating it as a linear inequality. Performing a binary search on this bound and repeatedly solving the corresponding satisfiability problems an optimal solution can be obtained. Since only the subset of clauses corresponding to the artificial bound on the objective function changes, the CNF formulas involved differ very little.

We conclude this appendix by mentioning a modification of the transformation that restricts the number of additional clauses. Consider again Figure (A.2). For example, we need for $p_0^{(U)}$ to be *true*, that either $p_0^{(V)}$ *or* $p_0^{(W)}$ is *true*, which is expressed by (A.3), i.e. we require *equivalence*. However, this may be relaxed to $p_0^{(U)} \leftarrow (p_0^{(V)} \leftrightarrow \neg p_0^{(W)})$, as *implication* suffices. Similarly, we can replace the first equivalences by implications in expressions (A.4-A.7). Thus the number of additional clauses is, roughly speaking, halved [136]. Observe that the idea that is used here is similar to Wilson's modification [139] of the Tseitin construction to construct CNF equivalents of logical formulas.

Appendix B

Taylor approximations and elliptic approximations

In this appendix we explain the relation between the elliptic approximations introduced in Section 2.4 and the nonlinear models such as discussed in Section 2.3. Rather than making direct use of the linear inequalities associated with the clauses, the elliptic approximation is derived as a second-order approximation of an exact nonlinear model of satisfiability problems. For all notation, see the above mentioned sections.

Let us consider the second order Taylor approximation of the function $\mathcal{T}(x)$. By straightforward calculus, we find the following expression of the first order partial derivative of $\mathcal{T}(x)$.

$$\frac{\partial}{\partial x_i}\mathcal{T}(x) = \sum_{k=1}^{n} \frac{\partial G}{\partial F_k}\frac{\partial F_k}{\partial x_i}.$$

The gradient of $\mathcal{T}(x)$ in $\overline{x}$ is denoted by $\nabla\mathcal{T}(\overline{x})$. We have the following expression for the second order partial derivative:

$$\frac{\partial^2}{\partial x_j \partial x_i}\mathcal{T}(x) = \sum_{k=1}^{n}\left[\sum_{l=1}^{n} \frac{\partial^2 G}{\partial F_l \partial F_k}\frac{\partial F_l}{\partial x_j}\frac{\partial F_k}{\partial x_i} + \frac{\partial G}{\partial F_k}\frac{\partial^2 F_k}{\partial x_j \partial x_i}\right].$$

The Hessian of $\mathcal{T}(x)$ in $\overline{x}$ is denoted by $\Delta\mathcal{T}(\overline{x})$. The second-order Taylor approximation of $\mathcal{T}(x)$ in $\overline{x}$ is given by

$$\mathcal{T}(x) \approx \mathcal{T}(\overline{x}) + (\nabla\mathcal{T}(\overline{x}))^T(x-\overline{x}) + \tfrac{1}{2}(x-\overline{x})^T(\Delta\mathcal{T}(\overline{x}))(x-\overline{x}).$$

If $\mathcal{T}(x)$ is convex (concave), its approximation is convex (concave) as well. If it is not convex, its approximation can be made convex using a property of $\{-1,1\}$ vectors. This is explained later on.

To get some insight in the structure of the Taylor approximation we observe the following. In general, the partial derivative

$$\frac{\partial F_k}{\partial x_i}$$

will be nonzero only if the associated proposition p_i occurs in the associated clause $\mathbf{C}_k$. Similarly, the partial derivative

$$\frac{\partial^2 F_k}{\partial x_i \partial x_j}$$

will be nonzero only if the associated propositional variables p_i and p_j occur simultaneously in clause $\mathbf{C}_k$. Thus, the gradient and Hessian are directly related (not unexpectedly) to occurrences and joint occurrences of variables in clauses.

As an example we consider the second-order Taylor approximation of the weighted polynomial representation (WPR) of the example in Section 2.3; see also Section 4.4.2. It holds that (cf. (2.4))

$$\frac{\partial P_k}{\partial x_j} = \frac{\partial}{\partial x_j}\prod_{i=1}^{m}(1 - a_{ki}x_i) = -a_{kj}\prod_{i\neq j}(1 - a_{ki}x_i),$$

and, for $j \neq l$,

$$\frac{\partial^2 P_k}{\partial x_j \partial x_l} = a_{kj}a_{kl}\prod_{i\neq j,l}(1 - a_{ki}x_i).$$

Computing the partial derivatives in the center $\overline{x} \equiv 0$ of the $\{-1,1\}$ hyper cube, we obtain

$$(\nabla\mathcal{P})_i = \sum_{k=1}^{n} w_k a_{ki} = (A^T w)_i,$$

where $w = [w_1, \dots, w_n]$, and

$$(\Delta\mathcal{P})_{ij} = -\sum_{k=1}^{n} w_k a_{ki} a_{kj} = -(A^T W A)_{ij},$$

where $i \neq j$ and $W = \operatorname{diag}(w)$. For $i = j$, $(\Delta\mathcal{P})_{ii} = 0$, due to the multilinearity of $P_k(x)$. This implies that $\Delta\mathcal{P}$ is indefinite. However, consider the second-order Taylor approximation around the center of the $\{-1,1\}$ hyper cube:

$$\mathcal{P}^{(2)}(x;w) = w^T Ax - \tfrac{1}{2}x^T(A^T WA - \operatorname{diag}(A^T WA))x.$$

By $A^TWA - \operatorname{diag}(A^TWA)$ we denote the matrix A^TWA with all its diagonal entries set to zero (note the similarity with equation (6.1)). Multiplying $\mathcal{P}^{(2)}(x;w)$ by -2 and expanding it, using that (by assumption) $e^TWe = e^Tw = 1$, we obtain

$$\begin{aligned} -2\mathcal{P}^{(2)}(x;w) &= -2w^TAx + x^TA^TWAx - x^T\operatorname{diag}(A^TWA)x \\ &= (Ax-e)^TW(Ax-e) - 1 - x^T\operatorname{diag}(A^TWA)x. \end{aligned}$$

Let us now consider the last term; it is constant for any $\{-1,1\}$ vector x, since (using Lemma 2.4.5)

$$x^T\operatorname{diag}(A^TWA)x = \sum_{i=1}^{m}\sum_{k=1}^{n} w_k a_{ki}^2 x_i^2 = \sum_{k=1}^{n} w_k \sum_{i=1}^{m} a_{ki}^2 = \sum_{k=1}^{n} w_k \ell(\mathbf{C}_k).$$

This property is in fact exactly the one mentioned in Section 2.4.4; there we observe that it can be utilized to obtain various equivalent models for binary quadratic optimization. In this case we use it to rewrite $\mathcal{P}^{(2)}(x;w)$ as a 'nice' convex quadratic function. Now it is our aim to find a value for th_{true}, such that it holds that

$$(Ax-e)^TW(Ax-e) \leq th_{true}$$

for any satisfying assignment $x \in \{-1,1\}^m$. Analogous to the derivation in Section 2.4, such a th_{true} can be found by using the integer linear programming formulation (IP_{SAT}). For any satisfying assignment x, it holds that $Ax \geq b$, while on the other hand it is easy to see that $a_k^T x \leq \ell(\mathbf{C}_k)$, hence $Ax \leq 2e - b$. Combining and subtracting e we find that

$$b - e \leq Ax - e \leq -(b - e),$$

from which it follows that (using that $w \geq 0$)

$$(Ax - e)^T W (Ax - e) \leq (b - e)^T W (b - e).$$

Rewriting this expression, we find that for any satisfying assignment x

$$x^T A^T W A x - 2w^T A x \leq b^T W (b - 2e).$$

Recalling the definition of the quadratic function $\mathcal{Q}(x; w)$ (see equation (2.7)) and observing that the right hand side is equivalent to $r^T w$ (since $b_k = 2 - \ell(\mathbf{C}_k)$), we have again derived the weighted elliptic approximation that is first specified in Theorem 2.4.1.

It may be noted that other nonlinear SAT representations such as mentioned in the examples in Section 2.3, give rise to similar elliptic approximations [96, 97].

Bibliography

[1] F. Alizadeh. *Combinatorial optimization with interior point methods and semi-definite matrices.* PhD thesis, University of Minnesota, Minneapolis, USA, 1991.

[2] N. Alon and J.H. Spencer. *The probabilistic method.* John Wiley and Sons, New York, 1992.

[3] K.M. Anstreicher and M. Fampa. A long–step path following algorithm for semidefinite programming problems. Technical report, Working paper, Department of Management Sciences, University of Iowa, Iowa City, USA, 1996.

[4] Y. Asahiro, K. Iwama, and E. Miyano. Random generation of test instances with controlled attributes. In Johnson and Trick [79], pages 377–393.

[5] T. Asano. Approximation algorithms for MAX SAT: Yannakakis vs. Goemans-Williamson. In *Proceedings of the 3rd Israel Symposium on Theory and Computing Systems*, pages 24–37, 1997.

[6] 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.

[7] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear programming: Theory and algorithms.* John Wiley and Sons, New York, 1993.

[8] B. Benhamou and L. Saïs. Theoretical study of symmetries in propositional calculus and applications. In *Proc. of the 11th Conference on Automated Deduction*, pages 281–294, 1992.

[9] S.J. Benson, Y. Ye, and X. Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. Technical report, Computational Optimization Lab, Department of Management Science, University of Iowa, Iowa City, USA, 1997.

[10] H.P. van Benthem. *GRAPH*: Generating Radio link frequency Assignment Problems Heuristically. Master's thesis, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1995.

[11] C.E. Blair, R.G. Jeroslow, and J.K. Lowe. Some results and experiments in programming techniques for propositional logic. *Computers and Operations Research*, 13(5):633–645, 1986.

[12] M. Böhm and E. Speckenmeyer. A fast parallel SAT-solver — efficient workload balancing. *Annals of Mathematics and Artificial Intelligence*, 17:381–400, 1996.

[13] B. Borchers and J. Furman. A two–phase exact algorithm for MAX–SAT and weighted MAX–SAT problems. *Journal of Combinatorial Optimization*, pages 299–306, 1999.

[14] E. Boros. Maximum renamable Horn sub–CNFs. Technical Report 5–97, RUTCOR Research Report, Rutgers University, 1997.

[15] E. Boros, Y. Crama, P.L. Hammer, and M. Saks. A complexity index for satisfiability problems. *SIAM Journal on Computing*, 23:45–49, 1992.

[16] E. Boros and A. Prékopa. Probabilistic bounds and algorithms for the maximum satisfiability problem. *Annals of Operations Research*, 21:109–126, 1989.

[17] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic minimization algorithms for VLSI minimization.* Kluwer Academic Publishers, 1985.

[18] R.E. Bryant. Graph–based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C–35(8), 1986.

[19] M. Buro and H. Kleine Bühning. Report on a SAT competition. *EATCS Bulletin*, 49:143–151, 1993.

[20] S.R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *The Journal of Symbolic Logic*, 52(4):916–927, 1987.

[21] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio Frequency Assignment Benchmarks. *Constraints*, 4(1):79–89, 1999.

[22] V. Chandru, C.R. Coullard, P.L. Hammer, M. Montanez, and X. Sun. On renamable Horn and generalized Horn functions. *Annals of Mathematics and Artificial Intelligence*, 1:33–47, 1990.

[23] V. Chandru and J.N. Hooker. Extended Horn sets in propositional logic. *Journal of the Association for Computing Machinery*, 38:205–221, 1991.

[24] V. Chvátal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4:305–337, 1973.

[25] M. Conforti and G. Cornuéjols. A class of logic problems solvable by linear programming. *Journal of the ACM*, 42(5):1107–1113, 1995.

[26] 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.

[27] S.A. Cook. A short proof of the pigeon hole principle using extended resolution. *SIGACT News*, pages 28–32, Oct.–Dec. 1976.

[28] S.A. Cook and R.A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44:36–50, 1979.

[29] W. Cook, C.R. Coullard, and G. Turan. On the complexity of cutting plane proofs. *Discrete Applied Mathematics*, 18:25–38, 1987.

[30] J.M. Crawford. Solving satisfiability problems using a combination of systematic and local search. Extended abstract, presented at *Second DIMACS Challenge*, Rutgers University, NJ, 1993.

[31] J.M Crawford and L.D. Auton. Experimental results on the crossover point in random 3SAT. *Artificial Intelligence*, 81, 1996.

[32] J.M. Crawford, M.J. Kearns, and R.E. Schapire. The minimal disagreement parity problem as a hard satisfiability problem. Manuscript, 1995.

[33] N. Creignou. Complexity versus stability for classes of propositional formulas. *Information Processing Letters*, 68:161–165, 1998.

[34] M. D'Agostino and M. Mondadori. The taming of the cut. Classical refutations with analytic cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.

[35] D. van Dalen. *Logic and structure.* Springer–Verlag, Berlin, 3rd edition, 1994.

[36] M. Davis, M. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[37] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:210–215, 1960.

[38] E. de Klerk. *Interior point methods for semidefinite programming.* PhD thesis, Delft University of Technology, Delft, The Netherlands, 1997.

[39] C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Mathematical Programming*, 62:557–574, 1993.

[40] I.I. Dikin. Iterative solution of problems of linear and quadratic programming. *Doklady Akademiia Nauk SSSR*, 174:747–748, 1967. Translated into English in *Soviet Mathematics Doklady* 8, 674–675.

[41] 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.

[42] D. Du, J. Gu, and P.M. Pardalos, editors. *Satisfiability problem: Theory and applications*, volume 35 of *DIMACS series in Discrete Mathematics and Computer Science.* American Mathematical Society, 1997.

[43] O. Dubois. Lecture held at DIMACS conference, Rutgers University, New Brunswick, NJ, March 1996.

[44] O. Dubois, P. Andre, Y. Boufkhad, and J. Carlier. SAT versus UNSAT. In Johnson and Trick [79], pages 415–436.

[45] L. Faybusovich. Semi–definite programming: a path–following algorithm for a linear–quadratic functional. *SIAM Journal on Optimization*, 6(4):1007–1024, 1996.

[46] U. Feige and M. Goemans. Approximating the value of two prover proof systems with applications to MAX 2SAT and MAX DICUT. In *Proc. Third Israel Symposium on Theory of Computing and Systems*, pages 182–189, 1995.

[47] J. Franco and M. Paull. Probabilistic analysis of the Davis–Putnam procedure for solving satisfiability. *Discrete Applied Mathematics*, 5:77–87, 1983.

[48] J. Franco and R. Swaminathan. Toward a good algorithm for determining unsatisfiability of propositional formulas. Technical report, Computer Science Department, University of Cincinnati, Cincinnati, Ohio, USA, 1995. To appear in *Journal of Global Optimization*.

[49] J.W. Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, Department of Computer Science, University of Pennsylvania, USA, 1995.

[50] G. Gallo and M.G. Scutellá. Polynomially solvable satisfiability problems. *Information Processing Letters*, 29:221–227, 1988.

[51] M.R. Garey and D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman and company, San Francisco, 1979.

[52] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP–complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[53] A. Van Gelder and Y.K. Tsuji. Satisfiability testing with more reasoning and less guessing. In Johnson and Trick [79], pages 559–586.

[54] P. Goel. An implicit enumeration algorithm to generate test sets for combinatorial logic circuits. *IEEE Transactions on Computers*, C–30(3):215–222, 1981.

[55] M.X. Goemans and D.P. Williamson. New $\frac{3}{4}$-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7(4):656–666, 1994.

[56] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[57] F. Granot and P.L. Hammer. On the use of boolean functions in 0-1 programming. *Methods of Operations Research*, 12:154–184, 1971.

[58] J.F. Groote and H. van Maaren. Equivalence of the concave optimisation method and d'Agostino tableaux for propositional logic. In *Proceedings of the 11th international symposium on computer and information sciences (ISCIS–XI)*, pages 41–51, 1996.

[59] J.F. Groote and J.P. Warners. The propositional formula checker HeerHugo. Technical Report SEN-R9905, Centre for Mathematics and Computer Science (CWI), Amsterdam, The Netherlands, 1999. Accepted for publication in the *SAT2000* issue of the *Journal of Automated Reasoning*.

[60] J. Gu. Local search for the satisfiability (SAT) problem. *IEEE Transactions on Systems, Man and Cybernetics*, 23(4):1108–1129, 1993.

[61] J. Gu. Global optimization for satisfiability (SAT) problem. *IEEE Transactions on Knowledge and Data Engineering*, 6(3):361–381, 1994.

[62] J. Gu, P.W. Purdom, J. Franco, and B.W. Wah. Algorithms for the satisfiability (SAT) problem: a survey. In Du et al. [42], pages 9–151.

[63] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[64] E. Halperin and U. Zwick. Approximation algorithms for MAX 4–SAT and rounding procedures for semidefinite programs. Technical report, Department of Computer Science, Tel-Aviv University, Tel-Aviv, Israel, 1998.

[65] P.L. Hammer. Boolean elements in combinatorial optimization. *Annals of Discrete Mathematics*, 4:51–71, 1979.

[66] P.L. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas.* Springer-Verlag, 1968.

[67] J. Harrison. Stållmarck's algorithm as a HOL derived rule. In J. von Wright, J. Grundy, and J. Harrison, editors, *Proceedings of TPHOLs'96*, volume 1125 of *LNCS*, pages 221–234, 1996.

[68] J. Håstad. Some optimal in–approximability results. In *Proceedings of the 28th annual ACM Symposium on the Theory of Computing*, pages 1–10, 1997.

[69] B. He, E. de Klerk, C. Roos, and T. Terlaky. Method of approximate centers for semi–definite programming. *Optimization Methods and Software*, 7:291–309, 1997.

[70] C. Helmberg and F. Rendl. A spectral bundle method for SDP. Technical Report ZIB Preprint SC–97–37, Konrad–Zuse–Zentrum, Berlin, 1997.

[71] J.N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12:217–239, 1988.

[72] J.N. Hooker. Resolution vs. cutting plane solution of inference problems: some computational experience. *Operations Research Letters*, 7(1):1–7, 1988.

[73] J.N. Hooker and C. Fedjki. Branch–and–cut solution of inference problems in propositional logic. *Annals of Mathematics and Artificial Intelligence*, 1:123–139, 1990.

[74] J.N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15(3):359–383, 1995.

[75] R.A. Horn and C.R. Johnson. *Matrix analysis.* Cambridge University Press, 1985.

[76] R. Impagliazzo, T. Pitassi, and A. Urquhart. Upper and lower bounds for tree–like cutting plane proofs. In *Proceedings of the 9th Annual IEEE Symposium on Logic in Computer Science*, pages 220–228, 1994.

[77] R.G. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[78] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[79] D.S. Johnson and M.A. Trick, editors. *Cliques, Coloring and Satisfiability: Second DIMACS implementation challenge*, volume 26 of *DIMACS series in Discrete Mathematics and Computer Science.* American Mathematical Society, 1996.

[80] S. Joy, J. Mitchell, and B. Borchers. A branch and cut algorithm for MAX–SAT and weighted MAX–SAT. In Du et al. [42].

[81] S. Joy, J. Mitchell, and B. Borchers. Solving MAX–SAT and weighted MAX–SAT using branch-and-cut. Technical report, Rensselaer Polytechnic Institute, 1998. Submitted.

[82] A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. Computational experience with an interior point algorithm on the satisfiability problem. *Annals of Operations Research*, 25:43–58, 1990.

[83] A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. A continuous approach to inductive inference. *Mathematical Programming*, 57:215–238, 1992.

[84] A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. An interior point approach to Boolean vector function synthesis. In *Proceedings of the 36th MSCAS*, pages 185–189, 1993.

[85] S. Karisch. CUTSDP – A toolbox for a cutting–plane approach based on semidefinite programming. User's guide/version 1.0. Technical Report IMM–REP–1998–10, Department of Mathematical Modelling, Technical University of Denmark, 1998.

[86] H. Karloff and U. Zwick. A 7/8–approximation algorithm for MAX3SAT? In *Proceedings of the 38th Symposium on the Foundations of Computer Science*, pages 406–415, 1997.

[87] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[88] N. Karmarkar, M.G.C. Resende, and K.G. Ramakrishnan. An interior point algorithm to solve computationally difficult set covering problems. *Mathematical Programming*, 52:597–618, 1991.

[89] O. Kullmann. A systematical approach to 3-SAT-decision, yielding 3-SAT-decision in less than 1.5045^n steps. Technical report, Johann Wolfgang Goethe–Universität, Fachbereich Mathematik, 60054 Frankfurt, Germany, 1995.

[90] O. Kullmann. Investigations on autark assignments. Technical report, Johann Wolfgang Goethe–Universität, Fachbereich Mathematik, 60054 Frankfurt, Germany, 1998. Submitted.

[91] T. Larrabee. Efficient generation of test patterns using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 11(1):4–15, 1992.

[92] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. Technical report, LaRIA, Université de Picardie Julves Verne, Amiens, France, 1999.

[93] Greentech Computing Limited. GT6 algorithm solves the extended DIMACS 32-bit parity problem. Note available from http://www.research.att.com/ ~kautz/ challenge/, 1998.

[94] L. Lovász. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, 1979.

[95] L. Lovász and A. Schrijver. Cones of matrices and set–functions and 0–1 optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991.

[96] H. van Maaren. Elliptic approximations of propositional formulae. Technical Report 96–65, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1996. To appear in *Discrete Applied Mathematics*.

[97] H. van Maaren. On the use of second order derivatives for the satisfiability problem. In Du et al. [42], pages 677–687.

[98] H. van Maaren, J.F. Groote, and M. Rozema. Verification of propositional formulae by means of convex and concave transforms. Technical Report 95–74, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1995.

[99] H. van Maaren and J.P. Warners. Bounds and fast approximation algorithms for binary quadratic optimization problems with application to MAX 2SAT and MAX CUT. Technical Report 97–35, Delft University of Technology, The Netherlands, 1997.

[100] B. Mazure, L. Saïs, and E. Grégoire. Boosting complete techniques thanks to local serach methods. *Annals of Mathematics and Artificial Intelligence*, 22:319–331, 1998.

[101] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 459–465, San Jose, CA, 1992.

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

[103] G.L. Nemhauser, M.W.P. Savelsbergh, and G.C. Sigismondi. MINTO, a mixed integer optimizer. *Operations Research Letters*, 15(1):47–58, 1994.

[104] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.

[105] Yu. Nesterov and M.J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research*, 22(1):1–42, 1997.

[106] D.V. Pasechnik. An interior point approximation algorithm for a class of combinatorial optimization problems: implementation and enhancements. Technical report, Department of Technical Mathematics and Informatics, Faculty of Information Technology and Systems, Delft University of Technology, 1998.

[107] R. Puri and J. Gu. A BDD SAT solver for satisfiability testing: An industrial case study. *Annals of Mathematics and Artificial Intelligence*, 17:315–337, 1996.

[108] P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *J. Comput. System. Sci.*, 37:130–143, 1988.

[109] M.G.C. Resende and T.A. Feo. A GRASP for Satisfiability. In Johnson and Trick [79], pages 499–520.

[110] J.A. Robinson. A machine–oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[111] T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Symposium on the Theory of Computing*, pages 216–226, 1978.

[112] I. Schiermeyer. Pure literal look ahead: An O(1.497^n) 3-satisfiability algorithm (extended abstract). In J. Franco, G. Gallo, H. Kleine Büning, E. Speckenmeyer, and C. Spera, editors, *Workshop on the Satisfiability problem, Siena*, University of Köln, 1996.

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

[114] M.H. Schulz, E. Trischler, and T.M. Sarfert. Socrates: A highly efficient automatic test pattern generation system. *IEEE Transactions on Computer-Aided Design*, 7:126–137, 1988.

[115] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In Johnson and Trick [79], pages 521–532.

[116] B. Selman, H. Kautz, and D. McAllester. Ten challenges in propositional reasoning and search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, Nagoya, Aichi, Japan, 1997.

[117] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, San Jose, CA, 1992.

[118] Y. Shang and B.W. Wah. A discrete Lagrangian-based global-search method for solving satisfiability problems. *Journal of Global Optimization*, 10:1–40, 1997.

[119] H.D. Sherali and W.P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for 0–1 programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

[120] J.P. Marques Silva and K.A. Sakallah. GRASP: A new search algorithm for propositional satisfiability. Technical Report CSE-TR-292-96, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan, USA, 1996.

[121] D.C. Sörensen. Newton's method with a model trust region modification. *SIAM Journal on Numerical Analysis*, 19:409–426, 1982.

[122] G. Stållmarck. A proof theoretic concept of tautological hardness. Incomplete manuscript, 1994.

[123] G. Strang. *Linear algebra and its applications.* Harcourt Brace Jovanovich, third edition, 1988.

[124] J.F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Technical report, Communications Research Laboratory, McMaster University, Hamilton, Canada, 1998.

[125] S. Tiourine, C. Hurkens, and J.K. Lenstra. An overview of algorithmic approaches to frequency assignment problems. Technical report, CALMA project, Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 1995.

[126] T. Trafalis, T. Terlaky, J.P. Warners, A.J. Quist, and C. Roos. Unsupervised neural network training via a potential reduction approach. Technical Report 96–172, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1996.

[127] L. Trevisan, G. Sorkin, M. Sudan, and D. Williamson. Gadgets, approximation and linear programming. In *Proceedings of the 37th Symposium on the Foundations of Computer Science*, pages 617–626, 1996.

[128] M.A. Trick. Second DIMACS challenge test problems. In Johnson and Trick [79], pages 653–657.

[129] G.S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, part 2:115–125, 1968. Reprinted in J. Siekmann and G. Wrightson (editors), *Automation of reasoning* vol. 2, Springer–Verlag Berlin, 1983.

[130] T.E. Uribe and M.E. Stickel. Ordered binary decision diagrams and the Davis–Putnam procedure. In *Proc. of First Conference on Constraints in Computational Logic*, volume 845 of *LNCS*, pages 34–49, 1994.

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

[132] A. Urquhart. The complexity of propositional proofs. *The Bulletin of Symbolic Logic*, 1(4):425–467, 1995.

[133] A. Urquhart. Open problem posed at SAT'98, May 10–14, Paderborn, Germany. 1998.

[134] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM review*, 38:49–95, 1996.

[135] J.P. Warners. A potential reduction approach to the Radio Link Frequency Assignment Problem. Master's thesis, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1995.

[136] J.P. Warners. A linear–time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68:63–69, 1998.

[137] J.P. Warners, T. Terlaky, C. Roos, and B. Jansen. Potential reduction algorithms for structured combinatorial optimization problems. *Operations Research Letters*, 21:55–64, 1997.

[138] J.P. Warners, T. Terlaky, C. Roos, and B. Jansen. A potential reduction approach to the frequency assignment problem. *Discrete Applied Mathematics*, 78(1–3):251–282, 1997.

[139] J.M. Wilson. Compact normal forms in propositional logic and integer programming formulations. *Computers and Operations Research*, 17(3):309–314, 1990.

[140] M. Yannakakis. On the approximation of maximum satisfiability. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–9, 1992.

[141] H. Zhang. SATO: An efficient propositional solver. In W. McCune, editor, *Automated Deduction — CADE-14*, LNAI 1249, pages 272–275. Springer, 1997.

[142] Y. Zhang. Solving large–scale linear programs by interior point methods under the MATLAB environment. Technical Report TR96–01, Department of Mathematics and Statistics, University of Baltimore County, Baltimore, Maryland 21222–5398, 1996.

Samenvatting

Het *satisfiability problem* (SAT) ('vervullings probleem') uit de propositie logica ligt aan de basis van veel als moeilijk beschouwde problemen binnen verschillende vakgebieden, zoals kunstmatige intelligentie, informatica, elektrotechniek en besliskunde. Het probleem betreft de vraag of een logische formule waar gemaakt kan worden door het toekennen van geschikte logische waarden aan de variabelen, of dat zij een tegenspraak bevat. Het SAT probleem is NP-compleet. Derhalve zijn in het algemeen geen (theoretisch) efficiënte oplosmethodes beschikbaar; het wordt onwaarschijnlijk geacht dat zulke methodes bestaan. Wel bestaan er verschillende specifieke klasses van SAT die een efficiënt (polynomiaal) algoritme toestaan.

In de afgelopen jaren is de belangstelling voor SAT aanzienlijk toegenomen. Dit is het gevolg van een tweetal ontwikkelingen. Ten eerste hebben de hedendaagse computers een zodanige snelheid en geheugencapaciteit dat men tegenwoordig in staat is om met de klassieke algoritmes enorme problemen op te lossen; ten tweede zijn nieuwe effectieve oplosmethoden ontdekt. De klassieke algoritmes behoren doorgaans tot de *complete* methodes. Deze zijn altijd in staat te beslissen of een formule wel of geen tegenspraak bevat. Veel van de nieuwe algoritmes zijn *incompleet*. Dat wil zeggen dat ze niet in staat zijn om te herkennen dat een formule een tegenspraak bevat; als een formule echter vervulbaar is, blijken dergelijke methodes vaak veel effectiever dan de complete methodes. Als gevolg van al deze ontwikkelingen zijn specifieke SAT instanties vaak relatief eenvoudig oplosbaar; het zij met behulp van een van de beschikbare algoritmes danwel door ze te herkennen als behorende tot een klasse van theoretisch efficiënt oplosbare problemen. Derhalve, mede gegeven de expressieve kracht van de propositielogica, is het mogelijk om SAT coderingen van praktijkproblemen van aanzienlijke omvang op te lossen.

Om het hoofd te bieden aan SAT problemen van immer toenemende omvang, wordt steeds onderzoek gedaan naar het verbeteren van oplosmethodes en implementaties. Ongelukkigerwijs zijn er nog altijd problemen (ook relatief kleine) die zeer moeilijk oplosbaar zijn. De algemene verwachting is dat om ook vooruitgang te boeken bij het oplossen van deze problemen, nieuwe technieken moeten worden ontwikkeld. Zulke technieken zullen, beter dan de huidige, in staat moeten zijn om bepaalde structuren te benutten danwel gebaseerd moeten zijn op nieuwe benaderingen. In dit proefschrift worden dergelijke (zowel complete als incomplete) technieken ontwikkeld, waarbij gebruik wordt gemaakt van methoden uit de *logica*, *kunstmatige intelligentie* en *mathematische optimalisering.*

In de eerste twee hoofdstukken van dit proefschrift worden de benodigde noties en notaties geïntroduceerd en wordt getracht de lezer een indruk te geven van de stand van zaken op het gebied van de SAT algoritmiek. Zodoende wordt een aantal interessante fenomenen geïdentificeerd. Deze fungeren als leidraad voor het onderzoek beschreven in de hoofdstukken drie tot en met zes. Voorts wordt in hoofdstuk twee de elliptische approximatie afgeleid; deze is van eminent belang voor de ontwikkelde technieken. Door bepaalde eigenschappen van binaire variabelen te benutten, kunnen uit deze approximatie verschillende modellen worden afgeleid, elk met een eigen aard en daaruit voortvloeiende toepassingen. In elk van de hoofdstukken wordt een dergelijk specifiek model nader belicht en gebruikt in de ontwikkeling van nieuwe algoritmes.

In hoofdstuk 3 wordt een van de meest gebruikte algoritmes onder de loep genomen: het DPLL-algoritme. In dit algoritme worden impliciet alle mogelijke oplossingen van een formule getest (in een zogenaamde zoekboom) om zo te ontdekken of een aan alle voorwaarden voldoet. We trachten de effectiviteit van dit algoritme te verbeteren, door met behulp van elliptische approximaties de impliciete structuur van een formule te benutten. Inderdaad wordt op deze wijze de grootte van de zoekbomen gereduceerd.

In hoofdstuk 4 gaan we een stap verder door, wederom gemotiveerd door elliptische approximaties, expliciet een specifieke structuur te zoeken in formules. Indien een formule de gezochte structuur heeft, kan zij eenvoudig worden opgelost met een speciaal algoritme. Het blijkt dat formules met de bedoelde structuur juist moeilijk zijn voor de meer gebruikelijke algoritmes. In het algemeen kan de structuur herkend worden met behulp van een geschikt lineair optimaliseringsprobleem. Een bepaalde klasse van testproblemen, die met de voorheen beschikbare technieken onopgelost was gebleven, blijkt tamelijk eenvoudig oplosbaar met behulp van de voorgestelde technieken.

Ook in hoofdstuk 5 wordt gebruik gemaakt van technieken uit de mathematische optimalisering. Recentelijk is er een grote belangstelling voor het gebied van de semidefiniete programmering. Dit is een generalisatie van lineaire programmering. Het is gebleken dat semidefiniete programmerings formuleringen theoretisch vaak sterke eigenschappen hebben. Ook wanneer toegepast op SAT problemen is dit het geval. We voeren experimenten uit met het baanbrekende Goemans-Williamson approximatie algoritme om een indruk te krijgen van haar praktische toepasbaarheid om effectief SAT problemen exact op te lossen. Vervolgens formuleren we een optimaliseringsprobleem, gebaseerd op elliptische approximaties, waarmee bepaalde contradicties herkend kunnen worden. Met behulp van deze formulering kan een aantal probleem-klasses efficiënt worden opgelost.

In hoofdstuk 6 tenslotte worden combinatorische optimaliseringsproblemen gemodelleerd als minimaliseringsproblemen van niet-convexe functies over de eenheidskubus. Elke oplossing van het originele combinatorische optimaliseringsprobleem is een minimum van de herformulering en vice versa. Daarnaast heeft de herformulering de eigenschap dat elke gebroken oplossing eenvoudig kan worden afgerond naar een geheeltallige oplossing die tenminste even goed is. Met behulp van effectieve minimaliseringstechnieken kunnen oplossingen worden gevonden welke goede of zelfs optimale oplossingen zijn van het originele probleem. Een en ander wordt geïllustreerd middels het frequentie toewijzings probleem.

Het proefschrift besluit met twee appendices en een lijst met referenties.

Curriculum vitae

Joost Warners was born on January 8th, 1971 in Borsele. In 1989 he finished his secondary education at College Blaucapel in Utrecht. From 1989 to 1995 he studied technical mathematics at the Faculty of Technical Mathematics and Informatics of the Delft University of Technology. In the year 1994 until early 1995 he worked within the international CALMA project. This culminated in his Master's Thesis *A Potential Reduction Approach to the Radio Link Frequency Assignment Problem*, which in 1996 was awarded the annual prize of the Netherlands Society for Statistics and Operational Research (VVS).

In the summer of 1995 he started his Ph.D. research on the Satisfiability Problem. The research was initially performed at Delft university of Technology and at the Department of Philosophy at Utrecht University; from 1996 onwards the work was carried out in Delft and at the Centre for Mathematics and Computer Science (CWI) in Amsterdam. Currently Joost lives in Utrecht with his wife (and ...?) and he still holds an FC Utrecht season ticket.

Titles in the IPA Dissertation Series