Lehrstuhl für Theoretische Informatik
und Grundlagen der Künstlichen Intelligenz
Fakultät für Informatik
Technische Universität München

Diplomarbeit in
Informatik

Efficient Solving of Combinatorial Problems using SAT-Solvers

Stefan Kugele
`kugele@cs.tum.edu`

Aufgabensteller: Univ.-Prof. Dr. Helmut Veith
Betreuer: Dipl.-Ing. Christian Schallhart
Abgabedatum: 12. Oktober 2006

Erklärung

Ich versichere, dass ich diese Diplomarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Stefan Kugele
München, 12. Oktober 2006

Kurzfassung

In dieser Diplomarbeit beschäftigen wir uns mit Boolean Satisfiability (SAT), einem klassischen Suchproblem der Informatik. Stephen A. Cook und Leonid Levin zeigten, dass dieses Problem **NP**-vollständig ist, das heißt, dass sich viele, sehr unterschiedlich aufgebaute und gleichzeitig praktisch zentrale Suchprobleme auf SAT in einem streng formalen Sinn durch sogenannte Reduktionen zurückführen lassen. Im Rahmen dieser Diplomarbeit stellen wir eine Methode vor, um beliebige **NP**-vollständige Probleme mit Hilfe von modernsten, dem aktuellen Stand der Technik entsprechenden, SAT-Solvern zu lösen.

Zunächst verwenden wir einen auf Binärer Suche aufbauenden Ansatz, der als Entscheidungs-Prozedur SAT-Solver verwendet, um die Chromatische Zahl schwieriger Graphen zu finden. Mit dieser Technik haben wir Erfolg und können für bisher ungelöste Benchmarkprobleme optimale Färbungen berechnen.

Die Idee der Graph-Färbbarkeit wird als Grundlage des zweiten vorgestellten Problems verwendet: Inverse RNA-Faltung bezeichnet den Vorgang, um aus einer 2-dimensionalen Strukturbeschreiben (Sekundärstruktur) zu einer—biologischen Einschränkungen unterliegenden—Basenbelegung zu gelangen. Wir führen eine Reduktion ein, um die inverse RNA Faltung sehr schnell mit Hilfe von SAT-Solvern durchzuführen. Diese Reduktion wird in einem zweiten Schritt erweitert und angepasst. Um die Anzahl der möglichen Lösungen zu reduzieren, benutzen wir ein realistischeres Energiemodell. Wir verwenden hierzu Solver für Pseudo Boolesche Optimierungsprobleme (PBO), um den inversen RNA Faltungsprozess im Hinblick auf Energiestufen zu optimieren.

Letztlich beschäftigen wir uns mit einem Problem aus der Zahlentheorie, und zwar den Ramsey Zahlen $R(m, n)$. Obwohl diese Problem $\mathbf{coNP}^{\mathbf{NP}}$-hart zu sein scheint, werden wir für kleine Eingabewerte $R(m, n)$ errechnen. Ein rekursiver Algorithmus um die Grenzen für $R(m, n)$ zu ermitteln vervollständigt diese Arbeit.

Abstract

In this thesis we focus on Boolean Satisfiability (Sat), a classical search problem in computer science. Stephen A. Cook and Leonid Levin showed, that this problem is **NP**-complete, i.e., great many differently structured and at the same timepractically relevant search problems can be translated to Sat in a strictly formal sense by so-called reductions. In this thesis, we present a methodology to solve arbitrary **NP**-complete problems by dint of state-of-the-art Sat-solvers.

First, we use a binary search approach and state-of-the-art Sat-solvers as decision procedure to find the chromatic number of hard graph coloring instances. We apply the presented technique successfully and are able to find for formerly unsolved benchmark problems the optimal coloring.

The idea of graph coloring is taken as a basis for our second example: Inverse RNA folding denotes the process of obtaining a base assignment—restricted by biological constraints—given a 2-dimensional structure description (Secondary Structure). We introduce a reduction to perform inverse RNA folding very quickly with the help of Sat-solvers. This reduction is enhanced and adopted in a second step. To reduce the number of possible solutions we use a more realstic energy model. More specifically, we use solvers for Pseudo Boolean Optimization (PBO) problems, to optimize the inverse RNA folding process with respect to energy levels.

Finally, we deal with a number theoretical problem, namely the Ramsey numbers $R(m, n)$. Although this problem seems to be $\mathbf{coNP^{NP}}$-hard, we will calculate for small inputs $R(m, n)$. A recursive algorithm to determine bounds for $R(m, n)$ completes this thesis.

Acknowledgments

I want to thank the following persons who have supported me and thus made this thesis possible.

First and foremost, I am indebted to Christian Schallhart and Professor Helmut Veith. I have been working together with Christian for more than two years. For the SEP (research project during the master program) as well as for my diploma thesis, he provided me with help in all scientific and practical problems. Without him, this work would not have been possible. I give my special thanks to Helmut, for his constructive and helpful comments and new ideas on my work.

Last but not least, I thank everyone who contributed to this thesis but whom I did not mention.

Contents

1 Introduction **1**
1.1 Motivation 7
1.2 Outline 7

2 Experimental Setup and Format Definitions **8**
2.1 Sat-Solver 8
2.2 Format Definitions 9
2.2.1 DIMACS Format for CNF-Formula Encoding 9
2.2.2 PBO and LP Format for (Pseudo) Boolean Equations Encoding 10
2.3 Used Hardware Configuration 12

3 Chromatic Numbers **14**
3.1 Introduction 14
3.2 Reduction to Sat 14
3.2.1 Variable Mappings 15
3.3 MinColoring as Optimization Problem 16
3.3.1 Binary Search Algorithm 16
3.4 Experimental Results 17

4 RNA Problems **22**
4.1 Introduction 22
4.2 Reduction to SAT 24
4.2.1 Interpretation as Coloring Problem 25
4.2.2 Biological Constraints 26
4.2.3 Variable Mappings 26
4.3 Molecular Switch 29
4.4 Energy Parameters 31
4.5 RNA and Linear Programming 32
4.5.1 Introduction 32
4.5.2 Pseudo-Boolean Representation 35

5 Ramsey Numbers **41**
5.1 Introduction 41
5.2 Interpretation as Graph Theoretic Problem 41

5.2.1 Complexity Considerations 44
5.2.2 Reduction to SAT 46
5.3 Algorithm to determine Bounds for $R(m, n)$ 50
5.4 Graphs with neither a CLIQUE nor an INDEPENDENTSET 50
5.5 Estimation of the Instance Size 56
5.6 Experimental Results 56

6 Conclusion and Future Work **59**

A Chromatic Numbers **61**

B RNA Encodings **76**
B.1 Example CNF Encoding for the tRNA$^{\mathrm{Phe}}$ Molecule 76
B.2 Example IP Encoding for only one Base Stacking 79

C Ramsey Numbers **83**

List of Figures **85**

List of Tables **87**

List of Algorithms **88**

Bibliography **89**

Chapter 1

Introduction

Suppose that you are organizing a public viewing event for the FIFA World Cup 2006$^{\mathrm{TM}}$for a group of 400 international soccer fans. Because of the limited space around the screen, you have to choose 100 of visitors. They will receive places in the viewing area. To complicate matters, the Department of the Interior has provided you with a list of pairs of incompatible visitors, i.e., visitors who are prepared to use violence against each other. Then no pair from this list is allowed to appear in your final choice. This is an example of what computer scientists call an **NP**-complete problem, since it is easy to check whether a given choice of one hundred visitors proposed by a co-worker is satisfactory, i.e., no pair taken from your co-worker's list also appears on the list from the Department of the Interior. However the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing 100 viewers from the 400 applicants is greater than the number of atoms in the known universe!

One of the outstanding problems in computer science is whether there are computational problems which have quickly verifiable solutions, but which require an intractable long time to be solved by any deterministic procedure. Problems like the one listed above certainly seem to be of this kind, but so far no one was able to prove that these problems are really as hard as they appear, i.e., that they cannot be solved efficiently by a computer. Stephen Cook [9] and Leonid Levin [25] formulated the **P** (i.e., easy to find a solution) versus **NP** (i.e., easy to verify a given solution) problem independently in 1971. **P** is the class of tractable problems, since it is easy to find a solution for a given problem instance. In the case of **NP**, we only know that it is easy to verify a given solution—but we do not know whether it is easy to find such a solution. Thus **NP** contains problems which resisted an algorithmic approach so far and appear to be computational intractable. This characterization of tractable and intractable problems dates from workings from Alan Cobham [8] and Jack Edmonds [14]. Even earlier letters from Kurt Gödel to John von Neumann [19] addressed this topic. A formal definition of these classes reads as follows:

Definition 1 (Decision problem). A decision problem *is a problem which has as result for a given input instance either yes or no.*

Definition 2. ***P*** *is the complexity class containing decision problems which can be solved by a deterministic Turing machine [42] in an amount of time that is polynomial in the size of the input.*

Definition 3 (Alternative I). ***NP*** *is the complexity class containing all decision problems whose satisfying solution has polynomial size and can be verified in polynomial time.*

Definition 4 (Alternative II). ***NP*** *is the complexity class containing decision problems which can be solved by a non-deterministic Turing machine in an amount of time that is polynomial in the size of the input.*

In the latter definition, *non-deterministic* means that the Turing machine can make non-deterministic computation steps. Unlike to a deterministic Turing machine whose next action is uniquely defined, the non-deterministic Turing machine may have a choice between several next actions. In this way, the Turing machine guesses at each computation step the right action, i.e., it choses the action which eventually leads to an accepting state, if such an action exists. A non-deterministic Turing machine acceps an input if there is some sequence of choices, which are made in a non-deterministically way, that leads to an accepting state. It is sufficient to have a single accepting path in the computation tree. An input is rejected, if and only if all possible computation paths lead to an rejecting state. This is different to a deterministic Turing machine which has a single determined computation path, and not a tree. A deterministic Turing machine accepts or rejects an input if the only computation sequence leads to an accepting state or an rejecting, respectively. Thus, it is obvious that a deterministic Turing machine is a special case of a non-deterministic Turing machine since the single computation path of the deterministic machine is also included as a path in the computation tree of the non-deterministic machine. Hence, it follows that **P** is a subset of **NP** since determinism is a special case of non-determinism. Whether this subset is proper or not is the big question ($\mathbf{P} \overset{?}{\subsetneq} \mathbf{NP}$).

In Computer Science the notion of a *reduction* has been formulated and in fact, lies at the very heart of complexity theory. Let us recall what a reduction is.

Definition 5 (Karp Reduction [23]). *Let A, B be decision problems. We say that B reduces to A if there is a poly-time transformation (reduction) R which, for every instance x of B, produces a corresponding instance $R(x)$ of A, such that $x \in B \Leftrightarrow R(x) \in A$. In symbols we write $B \leq_p A$.*

$B \leq_p A$ means that problem A is at least as hard as problem B: In Figure 1.1 on the facing page we show how to use an algorithm for A to solve problem B given a reduction R from B to A. Use reduction R to transform input instance x ($R(x)$) in order to have a correct input instance for algorithm A. A returns either *yes* or *no*

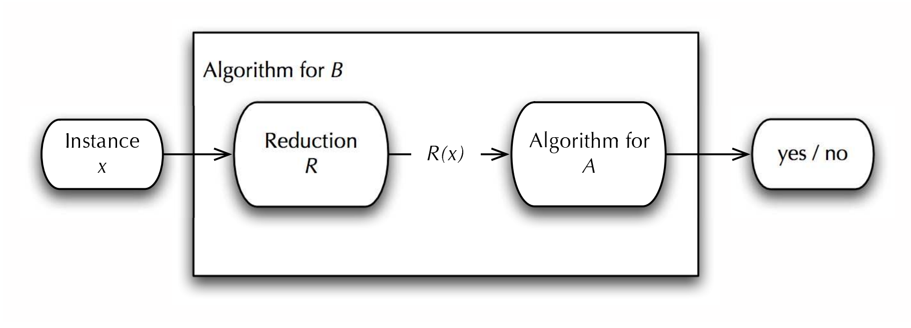


Figure 1.1: Reduction from B to A. According to [34] page 160.

which is the result for problem instance x. Using the concept of reductions, we can define further essential concepts of complexity theory: *hardness* and *completeness*.

Definition 6 (Hardness). *Let $\mathcal{C}$ be a complexity class and L a language. L is $\mathcal{C}$-hard with respect to $\leq_P$ if $L' \leq_P L$ holds for all $L' \in \mathcal{C}$.*

Definition 7 (Completeness). *Let $\mathcal{C}$ be a complexity class and L a language. We say that L is $\mathcal{C}$-complete if L is $\mathcal{C}$-hard with respect to $\leq_P$ and $L \in \mathcal{C}$ holds.*

In fact, problems with membership in **NP** occur in many real life applications. Starting from protein folding in biology, chemical synthesis, reconstruction of 3D images in medical applications going to problems in physics, statistics, finance and many more.

The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) has named seven Prize Problems. The Scientific Advisory Board of CMI selected these problems, focusing on important classic questions that have resisted solution over the years. The Board of Directors of CMI designated a \$7 million prize fund for the solution to these problems, with \$1 million allocated to each problem. The question whether $\mathbf{P} = \mathbf{NP}$ or $\mathbf{P} \neq \mathbf{NP}$ is one of the selected problems.

It is commonly believed that $\mathbf{P} \neq \mathbf{NP}$ holds and therefore one cannot hope to solve **NP**-problems effectively in general (since **P** is the class of efficiently solvable problems). However, there are some approaches dealing with **NP**-complete problems—none of them is perfect but each of them is an essential technique to practically handle these problems.

1. Special heuristics that are optimized for specific problems.
2. Approximation algorithms which return solutions of guaranteed quality. That leads to a whole new branch of complexity classes characterized by means of the poly-time approximability of the respective problems.

3. Calculation of the exact solution as efficient as possible having in mind that the algorithms we are using have exponential worst case run-time.

Our approach is based on solving the BOOLEAN SATISFIABILITY (SAT) problem as efficiently as possible. This is the basis for solving many other problems, e.g., Bounded Model Checking (BMC) [5] which uses SAT techniques very successfully. Our method is based on a two step process: first we reduce (transform) instances of a certain problem domain (e.g. bio-chemistry, number theory, etcetera) into a SAT formula (SAT domain). This formula acts as input for a state-of-the-art SAT-solver which has two possible results: either it returns a satisfying assignment or it states that the input formula is unsatisfiable. In the first case, the result has to be transformed back to the original problem domain. This two step process is illustrated in Figure 1.2. Formally, we use the following definition:

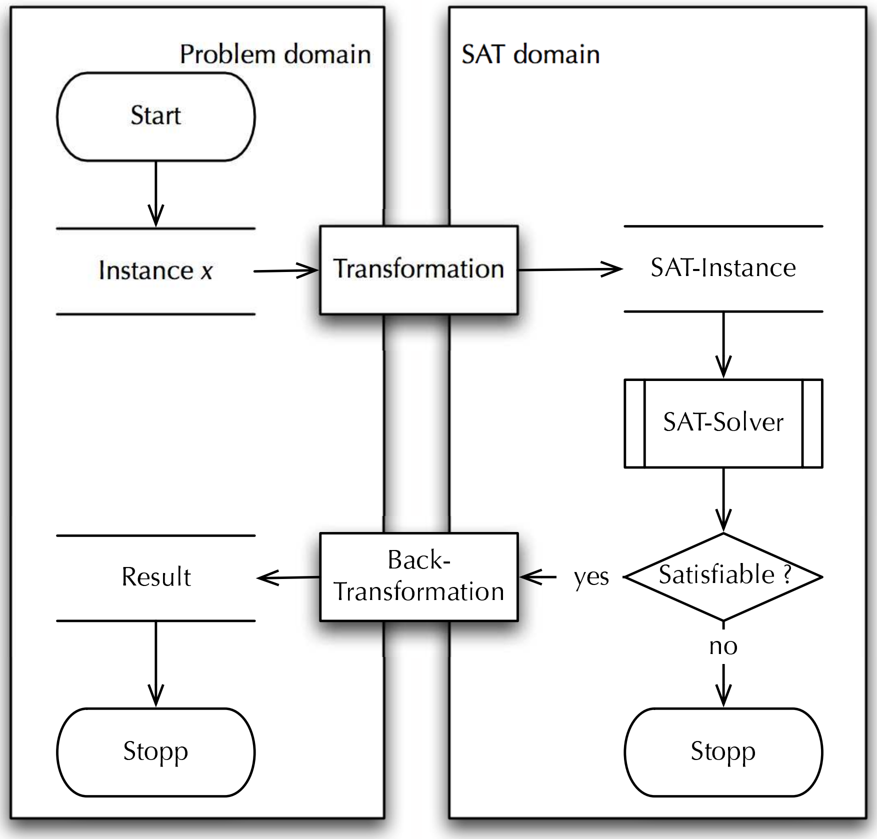


Figure 1.2: Solving problem domain specific instances by first reducing them to SAT, then using a SAT-solver to search for a result. If such a result exists, transform it back into the problem domain to interpret the result.

Definition 8 (Satisfiability). *A Boolean expression ϕ is* satisfiable *if and only if there* *exists at least one variable assignment to the variables of ϕ under which ϕ evaluates to* TRUE.

Definition 9 (SATISFIABILITY Problem). *An instance of the problem is defined by a Boolean expression written using only* AND, OR, NOT, *variables, and parentheses. The question is: given the expression, is there some assignment of truth values to the variables such that the expression will evaluate to* TRUE *under the assignment?*

In general, all SAT-solvers work on instances that have a certain normal form—the so-called *Conjunctive Normal Form* (CNF)

Definition 10 (Conjunctive Normal Form). *A Boolean formula is in* conjunctive normal form *(CNF) if it is a conjunction of clauses, where each clause is a disjunction of literals.*

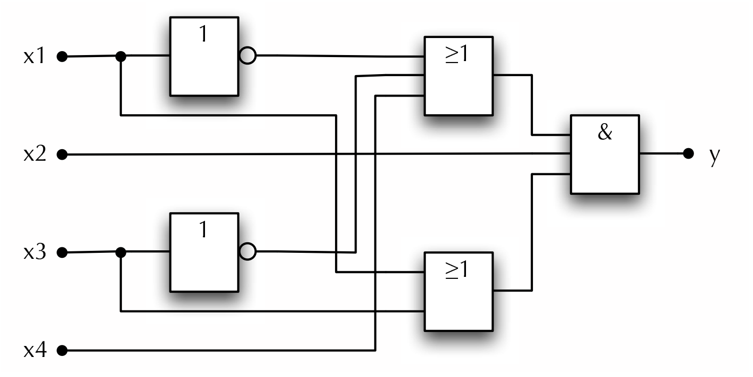
There exist of course other ways to formulate Boolean formulas but due to the fact that the conjunctive normal form is a de facto standard to formulate problem instances for SAT-solvers we will only concentrate ourselves to this form. The technical reason why most of the available SAT-solvers use CNF as their input format is the following: As already mentioned, CNF is a conjuncture of clauses. A formula ϕ evaluates to TRUE if and only if each clause contains at least one positive literal. Once, a clause evaluates to FALSE the complete formula ϕ is unsatisfiable. This is used to *propagate* literals in most SAT-solver implementation. For a detailed analysis of how state-of-the-art SAT-solvers work, we refer to [32]. `limboole` [3] as a front end to `limmat` [2] accepts arbitrary Boolean formulas as input and allows to check satisfiability and tautology. Inside, it then uses the so-called Tseitin-transformation [41] to transform the input into CNF. Let us have a look at the following example.

Example 1.

$$\varphi \equiv (x_1 \vee \neg x_3 \vee x_4) \wedge (x_2) \wedge (x_1 \vee \neg x_3) \tag{1.1}$$

Formula φ is in conjunctive normal form. It consists of three clauses, each having a different number of literals. On Figure 1.3 on the following page, we can see the corresponding circuit with four input pins $x_1, \dots x_4$, two NOT-gates, two OR-gates and one AND-gate. The output pin y has a high signal if and only if the input pins have an appropriate assignment with high or low signals. This problem is also referred to as CIRCUITSAT.

Probably, the most seminal theorem of complexity theory is Cook's theorem [9]. It states that SAT is **NP**-complete, i.e., all other problems of **NP** can be reduced in polynomial time to SAT. This subset is representative for the hardness of all problems in **NP** and is called **NP**-complete. A subsequent paper by Richard Karp [23] points



(a)

	x_1	x_2	x_3	x_4	x_1	x_2	x_3	x_4
solution 1	FALSE	TRUE	FALSE	FALSE	low	high	low	low
solution 2	TRUE	TRUE	FALSE	FALSE	high	high	low	low
solution 3	TRUE	TRUE	FALSE	TRUE	high	low	high	high
solution 4	TRUE	TRUE	TRUE	TRUE	high	high	high	high

(b)

Figure 1.3: (a) Formula (1.1) represented as a circuit, (b) φ's satisfying assignments on the left part of the table, on the right hand side we can see the assignment for the circuit's input pins.

out the true wealth of **NP**-complete problems, and therefore the significance of **NP**-completeness.

Theorem 1 (Cook, Levin). *(without proof)* SAT *is **NP**-complete.*

That was a milestone because due to the existence of a **NP**-complete problem the proof of **NP**-completeness for other problems become much easier. Other proofs of **NP**-completeness use a Karp reduction from an already known **NP**-complete problem to the relevant one.

That means that solving SAT allows to solve any other problem in **NP** by first reducing the problem to SAT, then solving the SAT instance, and finally by back-transforming the found assignment (if any) to the original problem domain. That means we cannot expect that there is a poly-time solver for SAT: if such a solver existed, $\mathbf{P} = \mathbf{NP}$ would hold. On the other hand, SAT-solvers work amazingly well in many practically relevant cases.

The efficiency of SAT-solvers has two main reasons, namely the use of highly optimized algorithms, in many cases a refinement of the *Davis-Putnam* [12] algorithm—the Davis-Putnam-Longmann-Loveland (DPLL) algorithm [11]—and the structure of real world problems. In this thesis we elaborate the question if we can use Sat-solvers to solve general **NP**-complete problems efficiently.

1.1 Motivation

As already mentioned in the introduction, Sat-solvers work pretty well even for large instances. A lot of work and research has been invested during the last decades to achieve that performance. Sat-solvers are so efficient for historical reasons, too—Sat was the first problem that has been shown to be **NP**-complete [9, 25]. Hence Sat plays a fundamental role in computational complexity and was studied intensively by many researchers with varying backgrounds.

The idea of reducing **NP**-complete problems to Sat and then using efficient Sat-solvers to search for solutions has a major advantage compared to other approaches: this is a totally generic way to solve problems in **NP**.

1.2 Outline

This work is organized as follows. We start with a motivating example to introduce the complexity class **NP**. Beside of other definitions we state the probably most seminal theorem of complexity theory: Cook's Theorem [9, 25]. It states, that Sat, i.e., the satisfiability problem for Boolean formulas, is **NP**-complete. Sat again is the problem at the center of this thesis.

Next is a brief description of the experimental environment, in Chapter 2. That is, a short overview of the used Sat-solver and used hardware. Finally, we introduce file formats used to encode the problems discussed in this thesis.

The aim of Chapter 3 is to present a Sat-based way to find chromatic numbers of hard graph instances. New results are presented.

In Chapter 4 we concentrate ours attention to bioinformatics questions. We consider different standpoints of RNA folding problems.

As another problem which allows the use of Sat-solvers, we focus on Ramsey Numbers, which is a number theoretical problem, in Chapter 5. We present an approach to determine bounds for even large Ramsey Numbers.

Finally, we conclude and present future directions in Chapter 6.

Chapter 2

Experimental Setup and Format Definitions

2.1 Sat-Solver

In this thesis, most computatios are made using the SAT-solver `minisat`. `minisat` is a minial SAT-solver developed by Niklas Eén and Niklas Sörensson. The term minimal refers to its source code size. Without comments it is unter 600 lines of C++ code and thus is one of the shortest implementations for a state-of-the-art SAT-solver. `minisat` is an advancement of `satzoo` and `satnik` which both are developed by the same authors as `minisat`.

`minisat` as most other SAT-solver is based in the DPLL [12, 11] algorithm and uses popular techniques like *conflict-driven backtracking* [27] and boolean constraint propagation (BCP) using *watched literals* [30]. Some of the fundamental techniques SAT-solvers use in general, are inspired by already existing solvers. Thus, `minisat` uses mostly the *propagation* procedure of `chaff` [30], the *learnig* procedure of `grasp` [28] and the *activity heuristic* of `satzoo` [15] which is a variant of the dynamic variable orderung based on activity first introduced in [30].

`limmat` [2, 4] and `compsat` [4] as two more SAT-solvers used in this thesis were developed at the formal methode group of Armin Biere at ETH Zürich. `limmat` follows in its implementation the ideas of `grasp` and `chaff`. Although `limmat` won some competitions during the SAT'02 Solver competition, its plain reasoning speed is much slower than the one of `chaff`, arguable based in its bulkyer data structures. On this account, `compsat` as the succesor of `limmat` has been designed to have a compact memory layout. Further improvements affect a new technique described in [6] to decompose SAT problems into connected components. We have to mention, that `compsat` only found one component for the problem instances used in this thesis.

In our experiments, `limmat` proved to be the slowest SAT-solver. `compsat` and `minisat` displayed an almost identical run-time behaviour. `zChaff` is in most cases fast and can be distinguished to be the robustest solver, i.e., did not crash during our experiments.

2.2 Format Definitions

2.2.1 DIMACS Format for CNF-Formula Encoding

A SATISFIABILITY problem in conjunctive normal form consists of a *conjunction* of clauses, where a clause is a *disjunction* of *literals*, i.e., a variable or its negation, see also Definition 10 on page 5. If we let x_i represent variables that can only assume the values TRUE or FALSE, then a sample formula in CNF would be

$$(x_1 \vee \neg x_3 \vee x_4) \wedge (x_2) \wedge (x_1 \vee \neg x_3) \tag{2.1}$$

where $\vee$ represents the Boolean *or* connective, $\wedge$ represents *and* and $\neg x_i$ is the negation of x_i. At this point, we refer to Example 1 given in the introduction. The SAT problem is to determine if such a formula is satisfiable, i.e., whether there exists an assignment such that the formula evaluates to TRUE under this assignment.

To represent an instance in the same way, we will create an ASCII file consisting of two major parts that cover all the information necessary to specify the problem.

Preamble The Preamble contains information about the input instance which is organized in lines. Each of these lines begins with a single character specifying the line type followed by a space. The following types can be distinguished:

Comments Comment lines are ignored by programs. They appear at the beginning of the preamble and are intended to give human-readable information about the file. Each comment line begins with a `c`.

```
c This is a comment line.
```

Problem line In each ASCII file, there is exactly one problem line. Problem descriptions have the following format.

```
p FORMAT VARIABLES CLAUSES
```

The lowercase `p` marks the line as a problem line. In the `FORMAT` field we can specify the type as which programs should interpret the file. In the CNF case we have to write `cnf`. The `VARIABLES` field contains the integer value specifying the number of variables in the instance. With `CLAUSES` we have to define the number of clauses of the input instance.

Clauses Clauses appear immediately after the problem line. Variables are numbered from 1 to `VARIABLES` whereas it is not necessary that each number appears in the instance. Numbers are separated by spaces, tabs or newlines. Positive variables i are represented just by writing `i` and negative variables $\neg i$ by writing `-i`. Each clause is terminated by `0`.

For the example given in Formula (2.1) a possible input file would be

```
c Example of an CNF format file
c
p cnf
1 -3 4 0
2 0
1 -3 0
```

2.2.2 PBO and LP Format for (Pseudo) Boolean Equations Encoding

In this section, we want to give a short review on the ASCII file format that is used for coding LP problem data[1]. LP is short for Linear Programming and intends to find an extreme point (i.e. minimum or maximum) of a linear optimization function subject to a set of linear inequalities. The format used in this thesis is the so-called *CPLEX LP* format, that was developed in the end of the 1980's by the CPLEX Optimization, Inc., which has been acquired by the ILOG, Inc. in 1997. It is intended to formulate Integer Programming (IP) and Linear Programming (LP) problems. This line-oriented file format is ordered in general like:

Objective function definition This defines the function which should either be minimized or maximized. The prescribed form is:

$$\left\{ \begin{array}{l} \texttt{minimize} \\ \texttt{maximize} \end{array} \right\} \quad f : s\ c\ x\ s\ c\ x\ \ldots\ s\ c\ x$$

where f is the name of the objective function, s is a sign (`+` or `-`), c is a numeric constant that denotes an objective coefficient and x is a variable name.

Constraints section The constraints section defines a system of equalities and/or inequalities. The required form is as follows:

[1]For detailed information on the CPLEX LP format, have a look at http://plato.asu.edu/cplex_lp.pdf (visited on 2006-06-21)

$$\begin{array}{r}\texttt{subject to}\\ constraint_1\\ constraint_2\\ \vdots\\ constraint_m\end{array}$$

where $constraint_i, 1 \leq i \leq m$ is a constraint definition of the form:

$$r : s\ c\ x\ s\ c\ x\ \ldots\ s\ c\ x \left\{\begin{array}{c}\texttt{<=}\\ \texttt{>=}\\ \texttt{=}\end{array}\right\} b$$

which is defined in the same way as before, but with the addition that r denotes the name of the specified constraint and b is the right-hand side.

Bounds section In this optional section, we can define bounds for variables. If no such section exists, all variables are assumed to be non-negative. The compulsory form is:

$$\begin{array}{r}\texttt{bounds}\\ bound_1\\ bound_2\\ \vdots\\ bound_k\end{array}$$

where $bound_j, 1 \leq j \leq k$ is a bound definition where the following intuitive six forms are allowed: x `>=` l`,` l `<=` x`,` x `<=` u`,` l `<=` x `<=` u`,` x `=` t`,` x `free`.

General Integer and Binary section This section is intended to specify some variables as general integer or binary. The form has to be like:

$$\begin{array}{r}\left\{\begin{array}{c}\texttt{general}\\ \texttt{binary}\end{array}\right\}\\ x_1\\ x_2\\ \vdots\\ x_q\end{array}$$

where $x_l, 1 \leq l \leq q$ are symbolic variable names. Variables in the general integer section are assumed to be general integer variables. If they appear in the binary section, they are assumed to be binary, i.e, they have 0 as lower bound and 1 as upper bound.

End keyword The `End` keyword terminates the LP file.

The following example shows a small LP format file:

```
Minimize F: -x1 +3 x2 -5 x3

Subject to:
    const1: x1 +2 x2 >= 5
    const2: x3 -2 x2 <= 1

Bounds
    1 <= x1 <= 3
    x2 free
    x3 free

Integer
    x2
    x3

End
```

2.3 Used Hardware Configuration

For our experiments and benchmarks, we use more or less standard desktop machines. Although some computations are made on computer clusters, only one processor can be used at the same time. Table 2.1 on the facing page summarizes the available machines and their configurations. Some machines have multiprocessor configuration, but all Sat-solvers are single-threaded and thus only one processor can be used for a single problem instance.

alias	processor	CPU speed	memory	cache	OS	kernel
		GHz	GB	kB		
iMac	Intel Core Duo	2	2	2048	OS X 10.4.7	Darwin 8.7.1
wall	AMD Athlon XP 2600+	1.9	1	512	GNU/Linux	2.6.16-2-k7
roof	AMD Athlon XP 2600+	1.9	1	512	GNU/Linux	2.6.16-2-k7
hand	Intel Pentium 4 CPU 3.20GHz	3.2	3	512	GNU/Linux	2.6.16-2-686-smp
head	Intel Pentium 4 CPU 3.20GHz	3.2	3	512	GNU/Linux	2.6.15-1-686-smp
mind	Intel Pentium 4 CPU 3.20GHz	2.8	3	1024	GNU/Linux	2.6.15-1-686-smp
king	AMD Athlon XP 2400+	1.99	1	256	GNU/Linux	2.6.16-2-k7
guru	AMD Athlon XP 2400+	1.99	1	256	GNU/Linux	2.6.16-2-k7
lord	AMD Athlon XP 2400+	1.99	1	256	GNU/Linux	2.6.16-2-k7
lady	AMD Athlon XP 2400+	1.99	1	256	GNU/Linux	2.6.16-2-k7
opt01	AMD Opteron 850	2.4	8	1024	GNU/Linux	2.6.5-7.147-smp
opt33	AMD Opteron 850	2.4	8	1024	GNU/Linux	2.6.5-7.147-smp
rayhalle1	Sun-Fire-880 UltraSPARC III (8x)	0.9	16	8192	SunOS 5.10	118833-20
condor	Sun-Fire-880 UltraSPARC III (8x)	0.9	32	8192	SunOS 5.9	117171-10

Table 2.1: Used hardware configuration

Chapter 3

Chromatic Numbers

3.1 Introduction

Solving **NP**-complete decision problems is the natural application of SAT-solvers: such problems require a yes/no answer on their respective query. In the yes-case, the solver also delivers a satisfying assignment. Therefore, by formulating optimization problems as a series of decision problems, it is possible to solve optimization problems by employing SAT-solvers. In the following, we are going to use the MINCOLORING problem as optimization problem.

Definition 11 (MINCOLORING problem).

Instance: *Given a graph G.*

Query: *Find the minimum number of colors with which G can be colored.*

This value is called the *chromatic number* χ. One way to compute χ, is it check all possible colorings exhaustively and to search for the one with the least number of colors. To check all possible bounds is of course the most naive way—we will use binary search to optimize the search procedure. So better approaches deserve study.

3.2 Reduction to Sat

In this section, we are going to present a reduction from the k-COLORING problem to SAT. By solving a corresponding SAT formula, we find out whether a given graph instance is k-colorable or not. As we will see in Section 3.3 on page 16, we are using a binary search approach to search for the chromatic number. In this way, we are able to find the chromatic number χ of a graph, i.e., the minimum number of colors needed to color the graph.

The following reduction transforms any given instance consisting of a graph $G = (V, E)$ and an integer k, i.e. $\langle G, k\rangle$, into a corresponding CNF formula, i.e. k-COLORING $\leq_p$ SAT. The resulting formula ψ has to ensure some properties:

First of all, each node of the graph has to be colored by one of the k colors. We introduce variables v_i^k with the following meaning: v_i^k is assigned TRUE if and only if vertex i is colored with color k. To enforce that each vertex is assigned a color, we introduce formula φ_1. This is a conjunction of the clauses $(v_i^1 \vee v_i^2 \vee \ldots \vee v_i^k)$ for each vertex $v_i \in V$, $1 \leq i \leq |V|$.

$$\varphi_1 \equiv \bigwedge_{v_i \in V} \left(v_i^1 \vee v_i^2 \vee \ldots \vee v_i^k\right) \tag{3.1}$$

The second step is to ensure, that no two adjacent vertices v_i, v_j have the same coloring. Let φ_2 be the conjunction of the formula $\neg(v_i^1 \wedge v_j^1) \wedge \neg(v_i^2 \wedge v_j^2) \wedge \ldots \wedge \neg(v_i^k \wedge v_j^k)$ for each edge (v_i, v_j) in the graph G. By applying DeMorgan's law we get the following formula in CNF:

$$\varphi_2 \equiv \bigwedge_{(v_i, v_j) \in E} (\neg v_i^1 \vee \neg v_j^1) \wedge (\neg v_i^2 \vee \neg v_j^2) \wedge \ldots \wedge (\neg v_i^k \vee \neg v_j^k) \tag{3.2}$$

The resulting formula ψ in CNF is:

$$\psi \quad \equiv \quad \varphi_1 \wedge \varphi_2 \tag{3.3}$$

Graph G is k-colorable if and only if the formula ψ is satisfiable.

3.2.1 Mapping from variables to integers and back

In the Formulae (3.1) and (3.2) we use variables v_i^l, $1 \leq i \leq |V|, l \in \{1, \ldots, k\}$ specifying the color of each vertex of G. The DIMACS format specifies, that variables have to be expressed as integer values. Therefore, it is necessary to encode variables as integer values, which is done in the following way. Let us consider Table 3.1 on the following page: columns represent the k possible colors and the rows encode the vertex index. Let

$$f: \{v_i^l \mid i \in \{1, \ldots |V|\} \text{ is the vertex index and } l \in \{1, \ldots, k\} \text{ is a color}\} \longrightarrow \mathbb{N}^+$$

be the mapping function:

$$f(v_i^l) = (i - 1) \cdot k + l \tag{3.4}$$

As soon as a DIMACS compatible SAT-solver returns a satisfying result, our next task is the interpretation of this result. For that, we use the inverse mapping to obtain a solution for the original problem. We need two such functions, since we encoded

	1	2	...	k
1	1	2	...	k
2	$k+1$	$k+2$	...	$2 \cdot k$
$\vdots$				
i	$(i-1) \cdot k + 1$	$(i-1) \cdot k + 2$	...	$i \cdot k$

Table 3.1: Mapping from colored vertices to integer values.

two distinct values into a single variable, namely the vertex index and the possible coloring: the first one, π, identifies the vertex index which is represented by variable v_i^l and the second one, β, determines the coloring for variable v_i^l.

$$\pi(i,k) = \left\lceil \frac{i}{k} \right\rceil \tag{3.5}$$

$$\beta(i,k) = \begin{cases} i \bmod k & \text{if } i \,(\bmod k) \neq 0 \\ k & \text{else.} \end{cases} \tag{3.6}$$

With the introduced method it is possible to use Sat-solvers as a decision routine to check whether a graph is k-colorable or not. This observation will be used in the following sections. There, we are going to enhance this idea to deal with solving optimization problems. As consequential continuation, we are using the MinColoring problem.

3.3 MinColoring as Optimization Problem

It is obvious, that if a graph G is not k-colorable then it is not l-colorable for $1 \leq l \leq k-1$. A k-colorable graph is naturally j-colorable for $k+1 \leq j \leq |V|$. We use these two observations to develop an algorithm to find a MinColoring which uses a binary-search approach.

3.3.1 Binary Search Algorithm

The idea of the binary search approach is the following: find a pivot element p, for example by taking the middle of the range of possible colorings: $p = \lfloor \frac{|V|}{2} \rfloor$. If G is p-colorable, then apply this procedure recursively to the range $[1; p-1]$, otherwise continue search in the interval $[p+1; |V|]$. In this vain, $\chi(G)$ will be more and more encircled until the procedure terminates and $\chi(G)$ has been found. On Figure 3.1 on page 18, we can see for the graph instance *queen8_ 8*[1] the steps of the binary search

[1]All graph instances are taken from [40, 31]

Algorithm 1 MinColorSearch

```
Require: $UpperBound = LastKnownColoring \leftarrow |V|$
 1: call Search(1, $|V|$, $|V|$);

 2: procedure Search($LowerBound$, $UpperBound$, $LastKnownColoring$)
 3:     if ($LowerBound = UpperBound$) then
 4:         if (Colorable($LowerBound$)) then
 5:             return $LowerBound$;
 6:         else
 7:             return $LastKnownColoring$;
 8:         end if
 9:     end if
10:     $m \leftarrow \left\lfloor \frac{LowerBound+UpperBound}{2} \right\rfloor$;
11:     if (Colorable($m$)) then
12:         Search($LowerBound$, $m-1$, $m$);
13:     else
14:         Search($m+1$, $UpperBound$, $LastKnownColoring$);
15:     end if
16: end procedure
```

algorithm. A solid arc going from one state to the next state, indicates that the graph can be colored by the color written in the source state. Dashed arcs, on the other hand, indicate that the appropriate graph cannot be colored by the source state label. E.g., the given graph is not 6-colorable, thus we have a dashed arc going from "6" to "9". We can see a red dashed arc going from state "8" to state "9". This tells us that $\chi(G)$ must be 9, as G is not 8-colorable but 9-colorable. Table 3.2 on page 19 shows for *queen8_8* the time needed to check whether a Sat-formula encoding an instance $\langle queen8\_8, k\rangle$ is satisfiable or not. The measurements are made using different Sat-solvers on *hand*. For a detailed description of the machine configurations, cf. Section 2.3 on page 12. The closer the algorithm encircles the chromatic number χ, the longer the run-time is to check whether the graph can be colored with the currently used number of colors. To illustrate the growth in a reasonable way, we use a logarithmic scale on the y-axis. The result can be seen in Figure 3.2 on the following page.

3.4 Experimental Results

In the following, we summarize our experiments on determining the chromatic number of hard graph instances. All computations are made with the above-named binary search algorithm. Most of them are taken from the Second DIMACS Implementation Challenge in 1992-1993. In addition to Michael Trick's graph benchmark instances

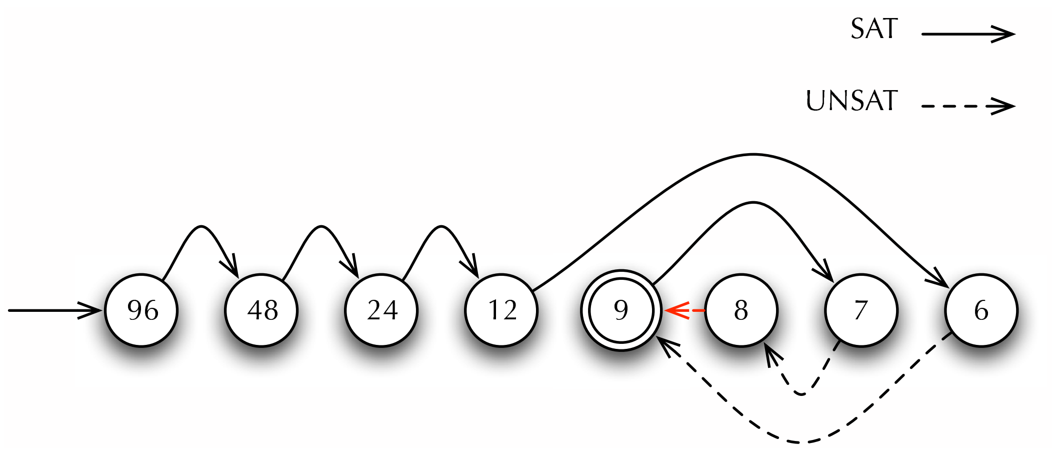


Figure 3.1: Example to illustrate in what way the chromatic number $\chi(G)$ can be found using a binary search approach. The used example instance is *queen8_ 8*.

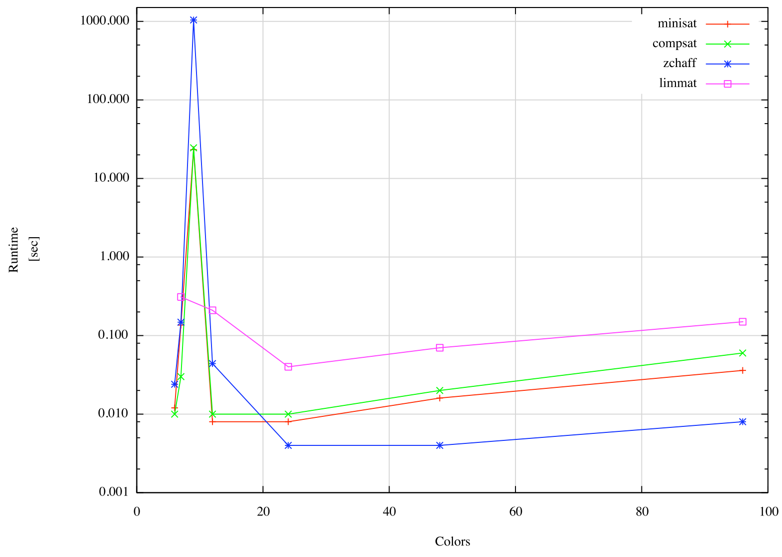


Figure 3.2: Run-time of different SAT-solver to find $\chi(queen8\_ 8)$

status	colors	`minisat` [sec]	`compsat` [sec]	`zChaff` [sec]	`limmat` [sec]
SAT	96	0.036	0.06	0.008	0.15
SAT	48	0.016	0.02	0.004	0.07
SAT	24	0.008	0.01	0.004	0.04
SAT	12	0.008	0.01	0.044	0.21
SAT	9	23.282	24.55	1042.660	∞
UNSAT	8	segfault	segfault	47385.2	∞
UNSAT	7	0.136	0.03	0.148	0.31
UNSAT	6	0.012	0.01	0.024	0.08

Table 3.2: Run-times needed to check whether *queen8_8* can be colored with the given number of colors. In this test, the following SAT-solvers were used: `minisat` [15], `compsat` [4, 6], `zChaff` [26] and `limmat` [2, 4]. ∞ indicates, that computations were manually terminated due to too long run-times.

[40], we also use instances from ThanhVu H. Nguyen [31] which coincide in large parts, whereas the latter compilation seems to be more up-to-date.

Table 3.3 summarizes the experimental results. Apart from graph details like number of vertices and edges, the density and the determined chromatic number χ, we also state the CPU-time needed on the specified machine always using `minisat` as SAT-solver.

Table 3.3: Chromatic numbers for hard graph instances

graph	V	E	density	χ	CPU [sec]	machine
le450_5a	450	5714	6 %	5	622.767341	king
le450_5c	450	9803	10 %	5	5.436327	king
le450_5d	450	9757	10 %	5	8.280522	king
anna	138	493	5 %	11	78.740903	guru
david	87	406	11 %	11	66.115174	guru
huck	74	301	11 %	11	125.672010	guru
jean	80	254	8 %	10	5.013015	guru
queen5_5	25	160	51 %	5	0.024000	guru
queen6_6	36	290	45 %	7	1.868116	guru
queen7_7	49	476	40 %	7	0.060001	guru
myciel3	11	20	33 %	4	0.006609	iMac
myciel4	23	71	27 %	5	0.059093	iMac
myciel5	47	236	21 %	6	256.655792	iMac
mugg88_1	88	146	4 %	4	0.052001	king
mugg88_25	88	146	4 %	4	0.044000	king

continued on the next page

graph	V	E	density	χ	CPU [sec]	machine
mugg100_1	100	166	3 %	4	0.056001	king
mugg100_25	100	166	3 %	4	0.052000	king
ash331GPIA	662	4185	2 %	4	4.504785	iMac
ash608GPIA	1216	7844	1 %	4	15.742970	iMac
abb313GPIA	1557	53356	4 %	9	52129.585358	iMac
will199GPIA	701	6772	3 %	7	6.408402	hand
(*) 1-Insertions_4	67	232	10 %	5	238.114507	iMac
2-Insertions_3	37	72	11 %	4	0.026668	iMac
3-Insertions_3	56	110	14 %	4	0.188895	iMac
(*) 4-Insertions_3	79	156	5 %	4	1.527256	iMac
1-FullIns_3	30	100	23 %	4	0.013447	iMac
1-FullIns_4	93	593	14 %	5	0.089279	iMac
1-FullIns_5	282	3247	8 %	6	12.347805	iMac
2-FullIns_3	52	201	15 %	5	0.024153	iMac
2-FullIns_4	212	1621	7 %	6	0.523934	iMac
3-FullIns_3	80	346	11 %	6	0.054105	iMac
3-FullIns_4	405	3524	4 %	7	4.841855	iMac
3-FullIns_5	2030	33751	2 %	8	35739.492569	iMac
4-FullIns_3	114	541	8 %	7	0.147497	iMac
4-FullIns_4	690	6650	3 %	8	90.528823	iMac
5-FullIns_3	154	792	7 %	8	0.772256	iMac
5-FullIns_4	1085	11395	2 %	9	1620.407509	iMac
DSJC125.1	125	1472	19 %	5	0.304403	king

Note, results for marked instances (*) differ from the results stated in [31, 40]. Further information on the run-times of the steps of the binary search procedure are provided in Appendix A on page 61.

Table 3.4 gives bounds for the chromatic number. For these graphs, the minimal coloring has not been found yet, but the computations are still running.

Table 3.4: Bounds for chromatic numbers for hard graph instances

graph	V	E	density	$\chi \in [\ldots]$	CPU [sec]	machine
1-Insertions_5	202	1227	6 %	[4;6]	N/A	iMac
1-Insertions_6	607	6337	3 %	[1;10]	N/A	lord
2-Insertions_4	149	541	5 %	[4;5]	N/A	head
2-Insertions_5	597	3936	2 %	[4;6]	N/A	iMac

continued on the next page

graph	V	E	density	$\chi \in [\ldots]$	CPU [sec]	machine
3-Insertions_4	281	1046	3 %	[4;5]	N/A	mind
3-Insertions_5	1406	9695	1 %	[4;6]	N/A	wall
4-Insertions_4	475	1795	2 %	[4;5]	N/A	roof
4-FullIns_5	4146	77305	1 %	[8;9]	N/A	duke
DSJC125.5	125	7782	99 %	[9;12]	N/A	bath
le450_25d	450	17425	17 %	[1;29]	N/A	fort
queen10_10	100	2940	59 %	[10;13]	N/A	king+opt33
latin_square	900	307350	76 %	[10;900]	N/A	condor

To conclude this Chapter, we give a last survey of our new results that—as far as we know—have not been published yet.

Table 3.5: New, not published results for chromatic numbers

graph	V	E	density	$\chi \in [\ldots]$	CPU [sec]	machine
DSJC125.5	125	7782	99 %	[9;12]	N/A	bath
abb313GPIA	1557	53356	4 %	[9;9]	52129.585358	iMac
will199GPIA	701	6772	3 %	[7;7]	6.408402	hand
4-FullIns_5	4146	77305	1 %	[8;9]	N/A	duke
5-FullIns_4	1085	11395	2 %	[9;9]	1620.407509	iMac
latin_square	900	307350	76 %	[10;900]	N/A	condor

Chapter 4

RNA Problems

Ribonucleic acid (RNA) plays a central role within living cells performing a variety of tasks. The tree-dimensional structure of polymers determines their functions. The computation of the secondary structure is used as a simplified model of the real nucleic acid structure. In the following, we regard a secondary structure as a set of base pairs.

Nucleic acid molecules may have alternative non-native conformations, i.e., alternative spatial structures with high energy barriers separating them. These meta-stable conformations can fulfill functions completely different to those of the native structure. This important feature is used in nature to implement *molecular switches* [1].

In the following we introduce a method to calculate RNA sequences that are compatible with any number of secondary structures.

4.1 Introduction

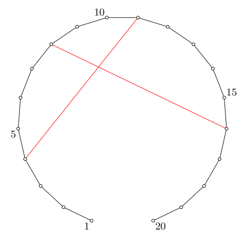
A secondary structure is defined as a set Ω of base pairs. We assume that the sequence positions are numbered consecutively from 1 to n. In the example (a) on Figure 4.2 on page 24 we have

$$\Omega = \{(1,20),(2,19),(3,18),(6,16),(7,15),(8,14),(9,13)\}$$

as secondary structure. Base pairs in secondary structure have to satisfy two constraints:

1. A base may participate in at most one base pair.

2. Base pairs may not cross, i.e., we cannot have two base pairs (i,j) and (k,l) with $i < k < j < l$. This condition excludes so called pseudo-knots.



In this case we have:
$i = 4$, $j = 11$, $k = 8$ and $l = 16$
which is not allowed.

Figure 4.1: Example for a pseudo-knot configuration.

The base pairing rule allows only 6 types of base pairs out of 16 possible combinations. These 6 pairs consist out of four Watson-Crick pairs [45] AU, UA, GC, CG and the two so-called "wobble" pairs GU, UG. In the following, we use the alphabet $\mathcal{A}$ of nucleic acids and pairing rules $\mathcal{B}$. The succeeding considerations can be adapted to arbitrary alphabets and general pairing rules. For the biophysical alphabet we have:

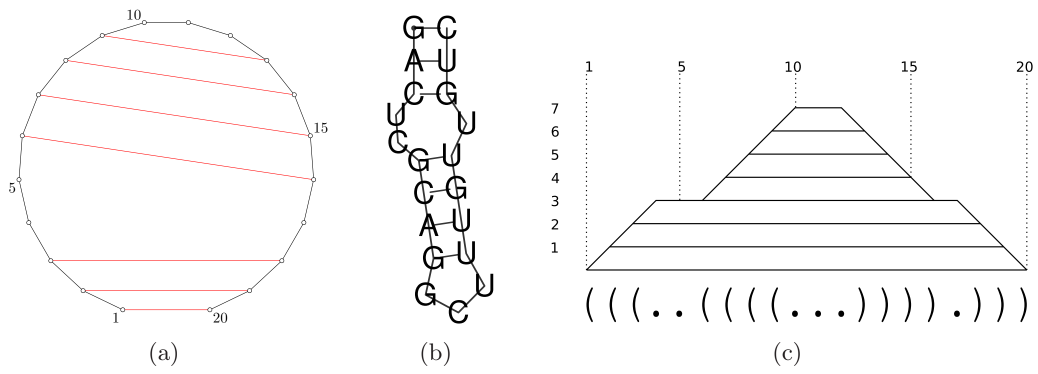
$$\mathcal{A} = \{\mathsf{A, G, C, U}\} \qquad \mathcal{B} = \{\mathsf{AU, UA, CG, GC, GU, UG}\} \tag{4.1}$$

One way to represent RNA secondary structures is the so-called "bracket notation" . Hereby, a secondary structure is composed of symbols "(", ")" and "." representing nucleotides that are paired with a partner towards the 3' end, towards the 5' end, and that are unpaired, respectively.

The basic module of ribonucleic acid are the nucleotides which consist of a ribose molecule (sugar), a hydroxyl and a phosphate group. Each of the five C-atoms (carbon) of the ribose is numbered from 1 to 5, referred to as 1' to 5'. If the hydroxyl group attached to the 3' carbon of one base attaches to the phosphate group attached to the 5' carbon of the next base, the polarity (orientation) of this strand is 5'-3'.

Pairs of matching parentheses therefore indicate base pairs, whereas a dot indicates that a base is unpaired. In Figure 4.2 on the next page we see among the bracket-dot representation of RNA secondary structure, three other forms of representation: at the circular representation (a), we organize the nucleotides in form of a circle and draw lines between them in the case of a base pairing. For large structures the so-called mountain representation (c) is very handy. A secondary structure is plotted in a two dimensional graph, in which the x-coordinate is the position k of a base in the sequence and the y-coordinate the number $m(k)$ of base pairs that enclose nucleotide k. In the following, we will use the circular representation which is an undirected graph $G = (V, E)$. V is the set of numbered base positions (vertices) and E is the set of edges indicated by red lines. Each edge $e = (i, j), e \in E$ symbolizes that we have a base pairing between the bases i and j.

Definition 12 (RNA folding problem). *The* RNA folding problem *is to find a secondary structure* Ω *of a given RNA sequence* σ*.*



(	(	(	.	.	(	(	(	(	.	.	.	)	)	)	)	.	)	)	)
G	A	C	U	C	G	C	A	G	G	C	U	U	U	G	U	U	G	U	C
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Figure 4.2: Different representations of secondary structure. (a) Circular plot, (b) conventional secondary structure graph and (c) mountain plot. Below, the structure is shown in the so-called "bracket notation".

It has been shown in [48, 47, 29] that this problem can be solved in an efficient way by using dynamic programming. On the other hand, the inverse RNA folding problem is defined as follows:

Definition 13 (Inverse RNA folding problem). *The* inverse RNA folding problem *is to find a base sequence σ which folds into a prescribed secondary structure Ω.*

In the following, we will turn our attention to the inverse RNA folding problem and develop a way to accomplish the problem with Sat- and LP techniques.

4.2 Reduction to SAT

In this section, we are going to present a reduction of the inverse RNA folding problem to Sat. This process will not use a single step, in fact we will first describe the problem from a graph theoretic point of view. Afterwards we give the actual translation into a Satisfiability problem.

The idea of solving the inverse RNA folding problem is as follows. As mentioned in Section 4.1 there are different ways to represent RNA secondary structures. In particular, we are interested in the *circular* representation which resembles the problem of coloring a graph.

Definition 14 (Graph). *An undirected* graph *G is the ordered pair (V, E), where V is a non-empty set of vertices and E is a set of 2-element subsets of V. A pair $\{u, v\} \in E$ is called an* edge.

Definition 15 (k-Coloring [18]). *Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$. The question is whether G is k-colorable, i.e., does there exist a function $f : V \to \{1, 2, \ldots, k\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?*

In our case, we are looking for a 4-Coloring due to the four bases A, G, C and U. So the question is, whether we can find a base-assignment that is compatible to a given secondary structure. In this context, compatible means that only the 6 types of base pairs (AU, UA, CG, GC, GU and UG) occur.

Our new approach uses at this point the capability of Sat-solvers instead of using coloring heuristics or other methods of bio-informatics. To be able to use these tools, we have to translate the Coloring problem into a formula in conjunctive normal form (CNF, for short) which serves as input.

Hence we can formulate the reduction (transformation) from the inverse RNA folding problem to Sat in two steps: in the first step, we transform an instance of the inverse RNA folding problem into an instance of the Coloring problem, then in the second step, we translate the constraint that restricts all possible pairings to the set of allowed pairings.

4.2.1 Interpretation as Coloring Problem

In the following, each consecutive position in the RNA strand corresponds to a vertex in the circular representation of the secondary structure. The first base position in the RNA strand corresponds to vertex v_1, the second with v_2, and so on. Each base-position $v_i \in V$ is either colored by A, G, C or U. We introduce the variables v_i^B, $B \in \{\mathsf{A}, \mathsf{G}, \mathsf{C}, \mathsf{U}\}$, where v_i^B means that on vertex i we have nucleotide B.

$$\bigwedge_{v_i \in V} \left(v_i^A \vee v_i^G \vee v_i^C \vee v_i^U\right) \tag{4.2}$$

Formula (4.2) guarantees, that each position is occupied by at least one of the bases. But it is possible that a single position is assigned multiple bases. To exclude this case, we have to add the following clauses which prohibit that a position is assigned to more than one base simultaneously.

$$\begin{aligned} \bigwedge_{v_i \in V} \neg\left(v_i^A \wedge v_i^G\right) \wedge \neg\left(v_i^A \wedge v_i^C\right) \wedge \neg\left(v_i^A \wedge v_i^U\right) \quad \wedge \\ \neg\left(v_i^G \wedge v_i^C\right) \wedge \neg\left(v_i^G \wedge v_i^U\right) \wedge \neg\left(v_i^C \wedge v_i^U\right) \end{aligned} \tag{4.3}$$

The following formula ensures that no two paired base-positions $(v_j, v_j) \in E$ are assigned the same base.

$$\bigwedge_{(v_i,v_j)\in E} \left(\neg v_i^A \vee \neg v_j^A\right) \wedge \left(\neg v_i^G \vee \neg v_j^G\right) \wedge \left(\neg v_i^C \vee \neg v_j^C\right) \wedge \left(\neg v_i^U \vee \neg v_j^U\right) \tag{4.4}$$

4.2.2 Biological Constraints

In addition, we have to ensure that only allowed base-pairings $\mathcal{B}$ occur:

$$\mathcal{A} = \{\mathsf{A},\ \mathsf{G},\ \mathsf{C},\ \mathsf{U}\} \qquad \mathcal{B} = \{\mathsf{AU},\ \mathsf{UA},\ \mathsf{CG},\ \mathsf{GC},\ \mathsf{GU},\ \mathsf{UG}\}$$

This formula restricts all possible pairings to only the allowed ones.

$$\bigwedge_{(v_i,v_j)\in E} \left(v_i^A \rightarrow v_j^U\right) \wedge \left(v_i^G \rightarrow \left(v_j^C \vee v_j^U\right)\right) \wedge \left(v_i^C \rightarrow v_j^G\right) \wedge \left(v_i^U \rightarrow \left(v_j^A \vee v_j^G\right)\right) \tag{4.5}$$

Note that Formula (4.5) is not in CNF, yet. But it can easily be rewritten in CNF using De Morgan's laws and rewriting rules from Boolean algebra. The conjunction of the four stated formulae is the result of the reduction from an inverse RNA instance to a SAT instance φ. Given a RNA secondary structure, the CNF formula φ is satisfiable if and only if there exists at least one base assignment that fulfills the base-pairing constraints. So far, we solve the inverse RNA folding problem which means that we have given exactly one RNA secondary structure and are looking for a base assignment.

4.2.3 Mapping from Variables to Integers and back

In the Formulae (4.2) to (4.5) we use variables specifying the base that a specific RNA position is assigned to. The problem at this point is, that the DIMACS CNF encoding format expects as input only integers. Thus, we use a transformation that maps possible base assignments represented by $c_i^b, 1 \leq i \leq |\Omega|, b \in \mathcal{A}$ to integers. Let us consider Table 4.1 on the facing page: columns represent the four bases and the rows encode the base position. Note that for arbitrary base alphabets this procedure has to be adopted. We used the standard alphabet $\mathcal{A}$ for RNA for the example shown in the table. Let

$$f:\ \{v_i^b \mid i \in \mathbb{N}^+ \text{ is the base position and } b \in \mathcal{A} \text{ is the base assignment}\} \longrightarrow \mathbb{N}^+$$

be the mapping function:

$$f(v_i^b) = (i-1)\cdot|\mathcal{A}| + b \tag{4.6}$$

where A corresponds to 1, G to 2, C to 3 and U to 4. As soon as a DIMACS compatible

	A	G	C	U
1	1	2	…	$\lvert\mathcal{A}\rvert = k = 4$
2	$k+1$	$k+2$	…	$2 \cdot k$
$\vdots$				
i	$(i-1)\cdot k+1$	$(i-1)\cdot k+2$	…	$i \cdot k$

Table 4.1: Mapping from coded based positions to integer values.

SAT-solver returns a satisfying result our next task is the interpretation of this result. Hence, the mapping which has just been introduced has to be inverted in order to obtain a solution for the original problem. We need two such functions, since we encoded two distinct values into a single Boolean variable, namely the position and a possible base assignment: the first one, π, identifies the position which is represented by variable v_i^b and the second one, β, determines the base assignment for a variable v_i^b.

$$\pi(i, |\mathcal{A}|) = \left\lceil \frac{i}{|\mathcal{A}|} \right\rceil \tag{4.7}$$

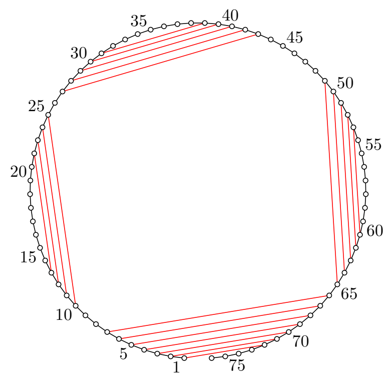
$$\beta(i, |\mathcal{A}|) = \begin{cases} i \bmod |\mathcal{A}| & \text{if } i \,(\text{mod} |\mathcal{A}|) \neq 0 \\ |\mathcal{A}| & \text{else.} \end{cases} \tag{4.8}$$

Once the SAT-solver's result has been interpreted, we have one of possibly many valid assignments. The solution at hand is chosen non-deterministically in the sense that different SAT-solvers may traverse the search space in different ways or even with some probabilistic methods. Thus, we cannot give any guarantee on the quality of the result with respect to metrics that may be interesting for biologists or chemists. In Section 4.4 on page 31 we get back to this consideration and introduce a simplified energy model which addresses the quality of a result namely for one RNA secondary structure. But let us come back to an example which illustrates the necessary steps to transfer a secondary structure to a valid base assignment.

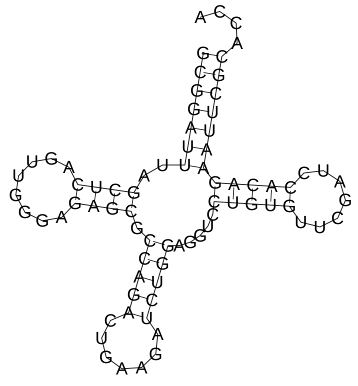
Example 2. *Let us assume that the secondary structure for the yeast tRNA^{Phe} molecule is given and we ask for a compatible base assignment, i.e., only base pairs from $\mathcal{B}$ appear. To simplify matters, we choose a relatively small secondary structure, but this works in the same manner for large secondary structures.*

```
(((((((..((((........)))).(((((.......))))).....(((((.......))))))))))))....
```

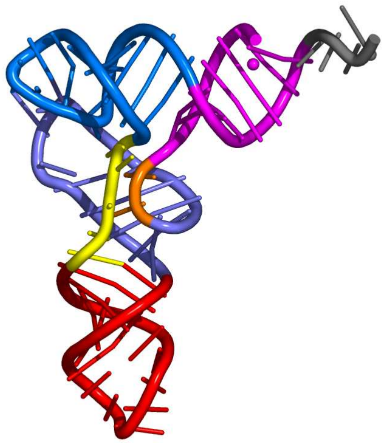
By parsing this input string conveniently we get a corresponding circular representation depicted on image (a) of Figure 4.3 on the following page. The idea of the parsing algorithm is outlined in Algorithm 2 on page 29. The different types of representation can be obtained from Figure 4.3 on the following page. In order to obtain a formula



(a)



(b)



(c)

Figure 4.3: Example for yeast tRNA$^{\text{Phe}}$: (a) shows the circular representation whereas (b) displays the common representation for the secondary structure. On image (c) we can see a 3D representation of the sample structure. This image is ray-traced by PyMOL [13] using the input file http://www.biosci.ki.se/groups/ljo/software/phe_trna.pse.gz (visited on July 1st)

which provides a basis for SAT*-solvers, we have to apply Formulae* (4.2) *through* (4.5) *to graph G. The resulting formula is listed in Appendix B.1 on page 76. After mapping the* SAT*-solver's result back to integers we get the base assignment shown in the second line of Figure 4.4 on the facing page. Note that* `*` *stands for arbitrary base assignment because no constraints were made for unpaired base positions. This result looks forced, but of course is a valid assignment using* `limmat` *[2]. The third line shows the base assignment for the yeast tRNA^{Phe} molecule as it occurs in nature. We can indeed find alternative base assignments as displayed in the last line. For example, the assignment shown in the second line is produced by* `limmat` *while the one of the third line is a biologically relevant one [46]. This is deeply rooted in the matter of fact, that* `limmat` *as the first used* SAT*-solver returns only one of possibly many results.*

```
(((((((..((((........)))).(((((.......))))).....(((((.......))))))))))))....
UUUUUUU**UUUU********GGGG*UUUUU*******GGGGG*****UUUUU*******GGGGGGGGGGGG****
GCGGAUUUAGCUCAGUUGGGAGAGCGCCAGACUGAAGAUCUGGAGGUCCUGUGUUCGAUCCACAGAAUUCGCACCA
```

Figure 4.4: Result for yeast tRNA$^{\mathrm{Phe}}$: first line: Bracket notation for the secondary structure, second line: interpretation of the solver's result, third line: alternative result.

But, as another example `satzoo` *[16] finds way above 240.000 satisfying models for the* SAT*-formula before interrupting further computations.*

Algorithm 2 SecondaryStructureToCircularRepresentation

```
1: Input: Secondary structure $\Omega$ in bracket notation where $n = |\Omega|$
2: Output: Graph $G = (V, E)$ where each $v \in V$ is a base position and each $e \in E$
   represents an opening and closing bracket pair
3: for $i \leftarrow 1$ to $n$ do
4:     $V \leftarrow V \cup \{i\}$
5:     if $\Omega[i] =$ '(' then
6:         push $i$
7:     end if
8:     if $\Omega[i] =$ ')' then
9:         $startVertex \leftarrow$ pop
10:        $endVertex \leftarrow i$
11:        $E \leftarrow E \cup \{(startVertex, endVertex)\}$
12:    end if
13: end for
14: return $G$
```

4.3 Molecular Switch

As mentioned above, RNA can fold into multiple non-native conformation, separated by an energy barrier and hence acts as a molecular switch: we can change conformation from one RNA secondary structure to another one just by increasing the temperature a little bit or other external events such as binding of another molecule. In the event of mutually exclusive alternatives where one corresponds to an active and the other one to an in-active conformation of the transcript, this can be used to switch between them and thus regulate gene expression. But note that each of the alternative structures has the same original base assignment. We can consider this problem for any number of secondary structures, however, currently no biological occurrence of an n-way switch for $n > 2$ is known. Interestingly, for any two secondary structures there exist sequences that are compatible with both structures, i.e. that

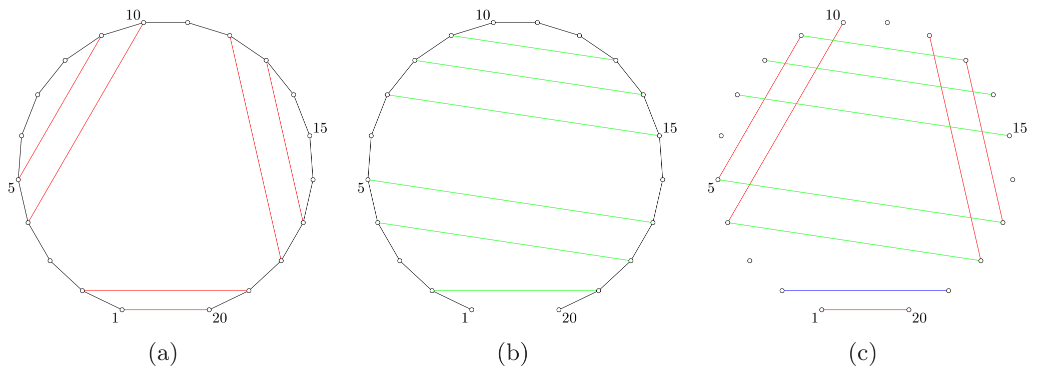


Figure 4.5: Dependency graph Ψ. (a) Circle representation of secondary structure 1. (b) Circle representation of secondary structure 2. (c) The dependency graph Ψ is constructed by super-imposing the circle representation of the two structures. Pairings that appear in both structures are colored blue. This example is according to [17].

can form both structures in principle [37]. To take account of this feature, we have to adopt and expand the graph coloring approach, conveniently.

We have an edge in a circular representation of a secondary structure if and only if the adjacent vertices form a base-pairing and thus must satisfy the base-pairing rules. Hence, this circular representation can be seen as a set of constraints for a single secondary structure's base assignment.

Definition 16 (RNA Switch Problem). *Given a set of RNA secondary structures $\Gamma = \{\Omega_1, \Omega_2, \ldots, \Omega_n\}$. The question is whether there is a base assignment which is compatible to each $\Omega_i \in \Gamma, 1 \leq i \leq n$.*

Now, a base assignment which should be compatible to each Ω_i, therefore has to comply all constraints of each single secondary structure. So it suggests itself to construct a so-called *dependency graph* Ψ which results from super-imposing every secondary structure graph.

We can solve the RNA switch problem for any number of secondary structures if and only if we can find a valid coloring for the dependency graph $\Psi = (V_\Psi, E_\Psi)$ with $V_\Psi = V_1 = V_2 = \ldots = V_n$ where n is the number of secondary structures, and $E_\Psi = \bigcup_{i=1}^{n} E_i$. If there exists such a coloring, then this assignment encodes a base-assignment which is compatible with each given secondary structure Ω_i in its circular representation $G_i = (V_i, E_i)$ for $1 \leq i \leq n$.

Example 3. *On the images (a) and (b) shown in Figure 4.5 we see two secondary structure graphs. To simplifie matters, we only use two structures, but this can be expanded to any number in this vein. For this new, super-imposed graph Ψ shown on image (c) we have to perform the reduction as specified above. A satisfiable resulting*

	CG	GC	GU	UG	AU	UA
CG	−2.4	−3.3	−2.1	−1.4	−2.1	−2.1
GC	−3.3	−3.4	−2.5	−1.5	−2.2	$\boxed{-2.4}$
GU	−2.1	−2.5	1.3	−0.5	−1.4	−1.3
UG	−1.4	−1.5	−0.5	0.3	−0.6	−1.0
AU	−2.1	−2.2	−1.4	−0.6	−1.1	−0.9
UA	−2.1	$\boxed{-2.4}$	−1.3	−1.0	−0.9	−1.3

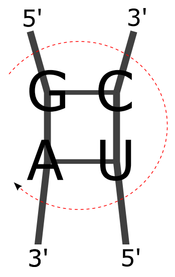


Figure 4.6: Free energies for stacked pairs in kcal/mol. Note that both base pairs have to be read in 5'-3' direction.

formula means, that we can color the dependency graph with the four bases A, G, C *and* U *and therefore get a base-assignment that is compatible for each of the given RNA secondary structures.*

4.4 Energy Parameters

Now, we consider again just a single secondary structure. A secondary structure corresponds not only to one conformation but also to a so-called *ensemble* of conformation compatible with a certain base pairing pattern. In the following, we use a simplified version of the “loop-based” energy model. In [21, 46], the authors state, that each secondary structure can be uniquely decomposed into *loops*, stacked base pairs are treated as loops of zero size. We can visualize a base stacking as a rectangle bordered by four bases. By twos opposing bases are paired, compare the right image of Figure 4.6. Different kinds of loops can be distinguished, whereas, in our approach we only use loops of size zero, i.e., stacked base pairs. The sum of all energy contributing loops (base stackings) is the secondary structure's energy due to the additivity of energy contributions. There are other approaches which use dynamic programming to calculate the minimum free energy recursively [44].

On the right image of Figure 4.6 wee see an example of a stacking pair that has an energy contribution of −2.4 kcal/mol [21]. The free energy of stacked pairs is listed in the table on the left hand side. Emphasized cells correspond to the example. Note that each base pair has to be read in 5'-3' direction which means for our example that we have to look up the corresponding energy value by reading a stacked base pair in clock-wise direction indicated by the dashed arrow: G-C and then U-A. The table is symmetric and thus can be used equivalently in the other direction.

In general, we find for a given secondary structure multiple compatible base assignments. But most desirable is an assignment which has the least free energy. Let us have a look at the following example.

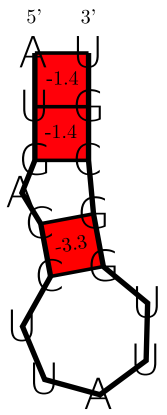


Figure 4.7: Example secondary structure with highlighted base stackings and contributing energies.

Example 4. *Assume secondary structure (((.((.....))))) is given . A compatible base assignment would be **AUGACCUUAUUGGCGU**. Figure 4.7 shows the structure.*

In the example above the red colored base stackings sum up to an energy value of -6.1 kcal/mol. The question is, whether this is the least free energy we can find for this example. In order to give an answer we have to solve in principle an optimization problem which would be in this case a minimization. This leads us directly to the next section.

4.5 RNA and Linear Programming

In a similar fashion as described in Section 4, we can give a reduction from the inverse RNA folding problem to INTEGER PROGRAMMING (IP). In this section, we evaluate the practical feasibility of this approach. IP is preferable to SAT, since optimization problems can be formulated in a more straightforward way. In our case, we want to minimize the free energy of the RNA molecule and thus have to solve an optimization problem. In the following, we first introduce INTEGER PROGRAMMING and then use IP to search for optimal solutions for the inverse RNA folding problem.

4.5.1 Introduction

Optimization in general deals with finding minima or maxima of a function f over a domain $\mathbb{X}$. In our case, we are looking for the minimum free energy during the simplified version of the loop decomposition. Thus, we are doing a minimization. Formally we can write a minimization problem as follows:

$$\begin{gathered}\min f(x)\\ \text{subject to } x \in \mathbb{X}\end{gathered}$$

where $f(x)$ is the objective function and $x \in \mathbb{X}$ is a constraint specifying the domain of x.

A special case where only linear optimization functions as well as linear constraints over the real numbers are allowed is referred to as a Linear optimization problem or better known as Linear Programming.

Definition 17 (Linear Programming Problem).

Instance:

$$\begin{array}{ll} \textit{minimize} & \sum_{j=1}^{n} c_j x_j \\ \textit{subject to} & \sum_{j=1}^{n} a_{ij} x_j \quad \le b_i, \quad 1 \le i \le m \end{array}$$

where the b_i's, c_j's, and a_{ij} are fixed real constants, and x_j's are variables over the real numbers.

Query: *The problem is to find a vector of x_j which satisfies $\sum_{j=1}^{n} a_{ij} x_j \le b_i, 1 \le i \le m$ and minimizes $\sum_{j=1}^{n} c_j x_j$.*

We used as example in the definition minimization, but we could have written equivalently maximization just as well, because both can be rewritten into each other. Often, the problem is expressed in matrix form as follows:

$$\text{minimize} \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\text{subject to} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \le \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

or shorten:

$$\begin{array}{rl} \text{minimize} & \mathbf{c}^T \mathbf{x} \\ \text{subject to} & A\mathbf{x} \le \mathbf{b} \quad , \quad \mathbf{x} \ge \mathbf{0} \end{array}$$

In 1947, Georg Dantzig introduced the so-called *simplex* algorithm [10] to solve LINEAR PROGRAMMING problems. Either the simplex algorithm finds a solution within a finite number of steps or it states the infeasibility of the problem. When requiring all variables to be integers, we call the problem an *Integer Programming* (IP) problem.

Definition 18 (INTEGER PROGRAMMING).

Instance:

$$\begin{aligned} &\textit{minimize} && \mathbf{c}^T\mathbf{x} \\ &\textit{subject to} && A\mathbf{x} \leq \mathbf{b} \quad , \quad \mathbf{x} \geq \mathbf{0} \end{aligned}$$

where the b_i's, c_j's, and a_{ij} are fixed real constants, and x_j's are integer variables.

Query: *The problem is to find the vector x_j which satisfies $A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}$ and minimizes $\mathbf{c}^T\mathbf{x}$.*

A special case of an IP problem is the 0-1 INTEGER PROGRAMMING problem which is also known as PSEUDO BOOLEAN OPTIMIZATION problem (PBO). Hereby, variables are required to be either 0 or 1. In the following we are using the notation PSEUDO BOOLEAN OPTIMIZATION.

Definition 19 (PSEUDO BOOLEAN OPTIMIZATION (PBO)).

Instance:

$$\begin{aligned} &\textit{minimize} && \mathbf{c}^T\mathbf{x} \\ &\textit{subject to} && A\mathbf{x} \leq \mathbf{b} \quad , \quad \mathbf{x} \geq \mathbf{0} \end{aligned}$$

where the b_i's, c_j's, and a_{ij} are fixed real constants, x_j's are integer variables, which can either be 0 or 1.

Query: *The problem is to find the vector x_j which satisfies $A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}$ and minimizes $\mathbf{c}^T\mathbf{x}$.*

The complexity of LP and IP is different, although the problems appear to be quite similar: for LP problems, a solution can be found in general within polynomial time and in fact, LP is **P**-complete. Khachiyan's ellipsoid method [24] was the first algorithm which solved LP in polynomial time. A more recent algorithm by Narendra Karmarkar seems to be much more promising in practice [22]. However, the simplex algorithm is the standard algorithm for LP although it has exponential run-time in the worst case.

However, IP as well as PBO are **NP**-complete [33]. As one of the most central **NP**-complete problems, 0-1 INTEGER PROGRAMMING was one of Richard Karp's

21 **NP**-complete problems which he published in his landmark paper "*Reducibility Among Combinatorial Problems*" [23].

In the following, we are going to confine ourselves to PBO problems which we are using to solve the inverse RNA optimization problem.

4.5.2 Pseudo-Boolean Representation

After having presented the required definitions, we transform Formulae (4.2) to (4.5) into a corresponding formulation in the CPLEX format for PBO problems. To do so, let us give a stepwise comparison between both representations. We have to mention, that the input instance remains the same as in the case of CNF—we have given a RNA secondary structure σ.

We write $\bigoplus_D C$ for a set of constraints C over a domain D which are written line-by-line in the CPLEX-format file. Let us give a short example to clarify this new notation.

Example 5. *In this example, an undirected graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_n\}$ is given. Edges are not of interest. Vertices can be colored either black or white. We ask for a vertex coloring, such that more than half of the vertices are colored black. In this case, domain D is V. We need constraints to ensure, that each vertex is assigned a single color and that most vertices are black. The color is denoted by $\bullet$ and $\circ$. For each vertex $v_i \in V$ we have the following constraint: $+1 * v_i^\bullet + 1 * v_i^\circ = 1;$.*

$$\begin{aligned} +1 * v_1^\bullet + 1 * v_1^\circ &= 1; \\ +1 * v_2^\bullet + 1 * v_2^\circ &= 1; \\ &\vdots \\ +1 * v_n^\bullet + 1 * v_n^\circ &= 1; \end{aligned}$$

With our notation, we write:

$$\bigoplus_{v_i \in V} +1 * v_i^\bullet + 1 * v_i^\circ = 1; \tag{4.9}$$

To satisfy the second part of the requirements we finally add as last constraint:

$$+1 * v_1^\bullet + 1 * v_2^\bullet + \ldots + 1 * v_n^\bullet > \frac{n}{2}; \tag{4.10}$$

Recall, that Formulae (4.2) and (4.3) ensure, that each RNA position is taken by exactly one base. Multiple assignments are not allowed. This property can be expressed

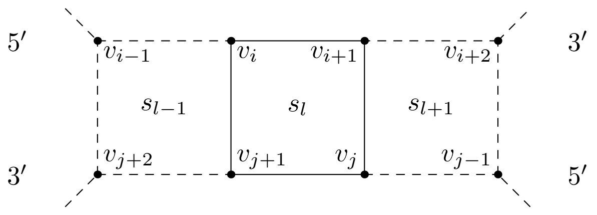


Figure 4.8: Each base stacking can hold one of 36 energy values.

by the following constraints.

$$\bigoplus_{v_i \in V} +1 * v_i^A + 1 * v_i^G + 1 * v_i^C + 1 * v_i^U = 1; \tag{4.11}$$

Now, the next step is to forbid that at a pairing participating base positions are assigned the same base.

$$\bigoplus_{(v_i,v_j)\in E} \left(\begin{array}{r} +1 * v_i^A + 1 * v_j^A \leq 1; \\ +1 * v_i^G + 1 * v_j^G \leq 1; \\ +1 * v_i^C + 1 * v_j^C \leq 1; \\ +1 * v_i^U + 1 * v_j^U \leq 1; \end{array} \right) \tag{4.12}$$

Next, the base pairing constraints ($\mathcal{B} = \{$AU, UA, CG, GC, GU, UG$\}$) which describe the set of possible pairings have to be encoded.

$$\bigoplus_{(v_i,v_j)\in E} \left(\begin{array}{r} -1 * v_i^A + 1 * v_j^U \geq 0; \\ -1 * v_i^G + 1 * v_j^C + 1 * v_j^U \geq 0; \\ -1 * v_i^C + 1 * v_j^G \geq 0; \\ -1 * v_i^U + 1 * v_j^A + 1 * v_j^G \geq 0; \end{array} \right) \tag{4.13}$$

By then, the transformation does exactly the same as in the CNF case. Now, to do the optimization of the energy level we have to consider the values given in Table 4.6 on page 31 (referred to as T). Figure 4.8 shows, that each base stacking is enclosed by four bases. Defined by these four bases, each of these rectangles contributes to the overall energy level one of the 36 constant values $C_1, C_2, \ldots, C_{36}$ listed in the table. To obtain integer coefficients, we multiply in the following each value C_i by 10. Let us traverse the table line-by-line starting with C_1 to obtain a consecutive numbering. For later considerations, it is necessary to distinguish between energy contributions, which come from different base-stackings of the instance at hand. Let $S = \{s_1, s_2 \ldots, s_k\}$ be the set of all such stacked rectangles with $1 \leq i < i+1 < j < j+1 \leq n$. Further considerations have to ensure two important points:

1. Each base-stacking rectangle is allowed to hold exactly one of the 36 energy values.

2. The overall energy value, i.e., the summation of energy values of all base-stacking rectangles, has to be minimized.

These two points are ensured by dint of the following deliberations.

Each rectangle can hold one of the C_i's as energy value. Let us introduce new indicator variables $A_{i+36(l-1)}$ with $1 \leq i \leq 36, 1 \leq l \leq k$. $A_{i+36(l-1)}$ becomes TRUE if and only if base-stacking l contributes with energy value C_i to the overall energy level. These indicator variables have to be chosen in a way that for each base-stacking s_l only one of them becomes TRUE. Thus, it is guaranteed that each stacking contributes precisely one energy value. So, it remains to choose the indicator variables adequately and therewith the base assignment, such that the overall energy is minimized.

In the following, we give a stepwise transformation from a Boolean formula into a formula in the PBO format. This is done using an example which can then be generalized. The following formula expresses that indicator variable $A_{32+36(l-1)}$ is TRUE if and only if positions v_i, v_{i+1}, v_j, and v_{j+1} of base-stacking s_l are assigned the bases G, A, U and C respectively:

$$\left(v_i^G \wedge v_{i+1}^A \wedge v_j^U \wedge v_{j+1}^C\right) \leftrightarrow A_{32+36(l-1)} \tag{4.14}$$

In order to transform this Boolean formula into the PBO format, we first decompose the equivalence into two implications, this yields:

$$\left(v_i^G \wedge v_{i+1}^A \wedge v_j^U \wedge v_{j+1}^C\right) \rightarrow A_{32+36(l-1)} \quad \wedge \quad A_{32+36(l-1)} \rightarrow \left(v_i^G \wedge v_{i+1}^A \wedge v_j^U \wedge v_{j+1}^C\right) \tag{4.15}$$

Rewriting the implication of both sub formulae leads to two constraints in the PBO problem definition.

$$\left[\neg\left(v_i^G \wedge v_{i+1}^A \wedge v_j^U \wedge v_{j+1}^C\right) \vee A_{32+36(l-1)}\right] \wedge \left[\neg A_{32+36(l-1)} \vee \left(v_i^G \wedge v_{i+1}^A \wedge v_j^U \wedge v_{j+1}^C\right)\right] \tag{4.16}$$

The formula on the left hand side of the logical *AND* as well as the formula on the right hand side lead to a constraints in the PBO problem definition. The first formula is for the left sub-formula, the second for the right one, respectively.

$$\begin{array}{c} -1 * v_i^G - 1 * v_{i+1}^A - 1 * v_j^U - 1 * v_{j+1}^C + 1 * A_{32+36(l-1)} > -4; \\ -4 * A_{32+36(l-1)} + 1 * v_i^G + 1 * v_{i+1}^A + 1 * v_j^U + 1 * v_{j+1}^C \geq 0; \end{array} \tag{4.17}$$

Now, we can give a formula for the general case. As aforesaid, each such stacking can hold one of the 36 energy values. Hence, we have to make the transformation step stated in Formula (4.17) for each stacking in S as well as for each combination of row base-pair and column base-pair. Let $T^r(p)$ and $T_c(p)$ $c, r \in \{1, 2, \ldots, 6\}, p \in \{1, 2\}$ be either the first or the second base-position p of a row r or a column c of table T. The following example clarifies the situation.

Example 6. *Let us have a look on the table already introduced above.*

		$T_2(1)$ $T_2(2)$				
	CG	*GC*	*GU*	*UG*	*AU*	*UA*
CG	-2.4	-3.3	-2.1	-1.4	-2.1	-2.1
GC	-3.3	-3.4	-2.5	-1.5	-2.2	-2.4
GU	-2.1	-2.5	1.3	-0.5	-1.4	-1.3
UG	-1.4	-1.5	-0.5	0.3	-0.6	-1.0
AU	-2.1	-2.2	-1.4	-0.6	-1.1	-0.9
UA	-2.1	$\boxed{-2.4}$	-1.3	-1.0	-0.9	-1.3
$T^6(1)$ $T^6(2)$						

Referring to example base stacking of Figure 4.6 on page 31, we box the value of interest - 2.4 kcal/mol. Variables are assigned as follows: $T_2(1) = G$, $T_2(2) = C$, $T^6(1) = U$ *and* $T^6(2) = A$.

$\pi(s_l, p)$ with $p = \{i, i+1, j, j+1\}$ returns the position in the RNA secondary structure sequence σ of the base which participates in base-stacking s_l with label p. Note, i, $i+1$, j, and $j+1$ are only labels to identify the four corners of a base-stacking rectangle. The actual position of these four bases in the RNA sequence is obtained by $\pi(s_l, p)$. Figure 4.9 on the facing page gives an example.

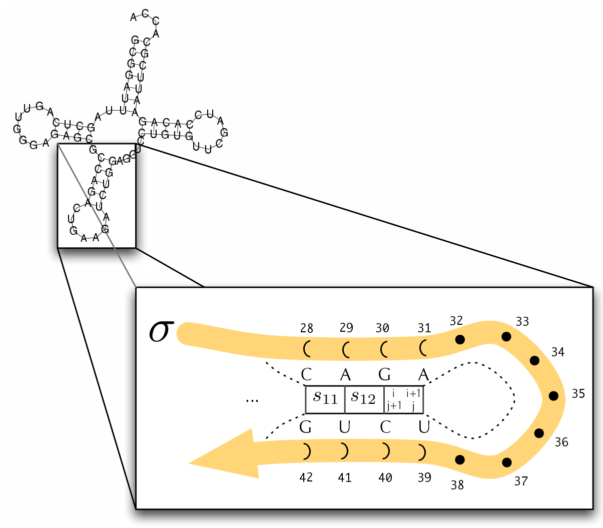
It follows the generalized version of Formula (4.17).

$$\bigoplus_{\substack{s_l \in S \\ 1 \le r,c \le 6}} \left(\begin{array}{l} -1 * v^{T_c(1)}_{\pi(s_l,i)} - 1 * v^{T^r(2)}_{\pi(s_l,i+1)} - 1 * v^{T^r(1)}_{\pi(s_l,j)} - 1 * v^{T_c(2)}_{\pi(s_l,j+1)} + 1 * A_{6(r-1)+c+36(l-1)} > -4; \\ -4 * A_{6(r-1)+c+36(l-1)} + 1 * v^{T_c(1)}_{\pi(s_l,i)} + 1 * v^{T^r(2)}_{\pi(s_l,i+1)} + 1 * v^{T^r(1)}_{\pi(s_l,j)} + 1 * v^{T_c(2)}_{\pi(s_l,j+1)} \ge 0; \end{array} \right) \tag{4.18}$$

As we said, each stacking can only hold one of the values of table T. We can formulate this like:

$$\bigoplus_{s_l \in S} \left(+1 * A_{1+36(l-1)} + 1 * A_{2+36(l-1)} + \ldots + 1 * A_{36+36(l-1)} = 1 \right) \tag{4.19}$$

The Formulae (4.11) (4.12), (4.13), (4.18) and (4.19) are all constraints we need to express all constraints to solve the inverse RNA folding problem by PBO. The next step is to declare all variables to be binary. This is done because we use these variables as indicator variables which are either TRUE or FALSE or 1 and 0 respectively. Variables declared in the `binary` section of a CPLEX file are restricted to 0 or 1. Formula (4.20)



$$\begin{aligned}\pi(s_{13}, i) &= 30\\ \pi(s_{13}, i+1) &= 31\\ \pi(s_{13}, j) &= 39\\ \pi(s_{13}, j+1) &= 40\end{aligned}$$

Figure 4.9: Example showing the usage of $\pi(s_l, p)$: use function π to determine the position of a base in secondary structure σ, which is defined by its label p and the base-stacking rectangle s_l to which it belongs.

restricts variables encoding vertices to 0 and 1 whereas Formula (4.21) is responsible for the restriction of indicator variables $A_{i+36(l-1)}$.

$$\bigoplus_{\substack{s_l \in S\\ b \in \mathcal{B}}} \begin{pmatrix} v^b_{\pi(s_l,i)} \\ v^b_{\pi(s_l,i+1)} \\ v^b_{\pi(s_l,j)} \\ v^b_{\pi(s_l,j+1)} \end{pmatrix} \tag{4.20}$$

$$\bigoplus_{s_l \in S} \begin{pmatrix} A_{1+36(l-1)} \\ A_{2+36(l-1)} \\ \vdots \\ A_{36+36(l-1)} \end{pmatrix} \tag{4.21}$$

Finally, we add the optimization function: in order to formulate a proper optimization function, it is necessary to refer to the energy values. We denote with $T(r, c)$ the energy values as described by the table, where r stands for the row and c for the column of the entry.

$$\sum_{\substack{s_l \in S\\ 1 \leq r,c \leq 6}} T(r, c) * A_{6(r-1)+c+36(l-1)} \tag{4.22}$$

In Appendix B.2 on page 79 we see an example CPLEX file which is the translation of the base stacking similar to those seen in Figure 4.6 on page 31. There are only two base-pairings, thus we are looking for the minimal energy value of just one rectangle.

When using the introduced transformation for the secondary structure of tRNA^{Phe}, which has been introduced in Figure 4.4 on page 29, we get a 107 kB large file and therefor it is not listed in this thesis. CPLEX was able to find a solution within 1.08 seconds on *rayhalle1*. The optimal found solution has energy level -56.5 kcal/mol after dividing by 10. Figure 4.10 completes Figure 4.4 by adding the found solution. The first line shows the RNA secondary structure, the second line a solution found by using a SAT-solver followed by the biologically relevant base assignment. Finally, in line four we see the solution found using PBO methods.

```
(((((((..((((........)))).(((((.......))))).....(((((.......))))))))))))....
UUUUUUU**UUUU********GGGG*UUUUU*******GGGGG*****UUUUU*******GGGGGGGGGGGG****
GCGGAUUUAGCUCAGUUGGGAGAGCGCCAGACUGAAGAUCUGGAGGUCCUGUGUUCGAUCCACAGAAUUCGCACCA
GGGGGGC**GCCC********GGGC*GGGGC*******GCCCC*****GCCCC*******GGGGCGCCCCCC****
```

Figure 4.10: Result for yeast tRNA$^{\mathrm{Phe}}$: First line: Bracket notation for the secondary structure, second line: interpretation of the SAT-solver's result, third line: alternative result and finally in the last line the optimal solution found by CPLEX.

Chapter 5

Ramsey Numbers

Imagine an alien force, vastly more powerful than us landing on Earth and demanding the value of $R(5,5)$ or they will destroy our planet. In that case, we should marshal all our computers and all our mathematicians and attempt to find the value. But suppose, instead, that they asked for $R(6,6)$, we should attempt to destroy the aliens. – Paul Erdős

5.1 Introduction

To describe intuitively what the Ramsey Numbers are, usually the so-called party problem is consulted. Thus, we want to maintain this proven method. Suppose, we are organizing a party of six people then there are at least three people who know each other, or three people who do not know each other. We can formulate the general case as follows: how many people do we need to invite to have at least m people to know each other, or n people who do not know each other. Frank P. Ramsey, after whom the *Ramsey Theory* is named, showed in 1930 that the size of the group is finite [36]. Since a group of constant size solving this (m,n)-constraint must exist, there must be an effective algorithm to find this value $R(m,n)$. Note, that we are not talking about an efficient way to calculate k with $R(m,n) = k$. However, we just said that there must be a procedure which finally gives us the value k.

5.2 Interpretation as Graph Theoretic Problem

The values $R(m,n)$ are called *Ramsey numbers* and describe in more abstract terms the minimal size of a graph, such that either at least m nodes are pairwise connected or at least n nodes are independent. Formally, we use the following definitions.

Definition 20 (Subgraph). *A subgraph $G' = (V', E')$ of a graph (V,E) is a graph for which $V' \subseteq V$ and $E' = E \cap V' \times V'$ holds.*

Definition 21 (Complete graph). *A* complete *graph is a graph where an edge connects every pair of vertices. A complete graph on n vertices is denoted by K_n.*

Definition 22 (Independent graph). *An* independent *graph is a graph without edges. An Independent graph on n vertices is denoted by I_n.*

Definition 23 (Clique). *A* clique *of a graph G of size k is a subgraph G' of G such that G' is isomorphic to K_k.*

Definition 24 (Independent Set). *An* independent set *of a graph G of size k is a subgraph G' of G such that G' is isomorphic to I_k.*

Let us give some more definitions in order to be able to define the central problem of this chapter in an easy formal way.

Definition 25 (m-CLIQUE).

Instance: *Given a graph G and an integer m: $\langle G, m\rangle$*

Query: *Does G has a* CLIQUE *of size m as subgraph?*

Definition 26 (n-INDEPENDENTSET).

Instance: *Given a graph G and an integer n: $\langle G, n\rangle$*

Query: *Does G has an* INDEPENDENTSET *of size n as subgraph?*

Definition 27 (RAMSEY).

Instance: *Given are three integers m, n and k: $\langle m, n, k\rangle$*

Query: *Does $R(m, n) > k$ hold? That is neither a m-*CLIQUE *nor an n-*INDEPENDENTSET *is contained in G.*

Now, the RAMSEY problem will be stated as follows. For all graphs of size k, is it possible, either to find a m-CLIQUE or an n-INDEPENDENTSET?

Definition 28 (Ramsey Number Problem). *$R(m, n) = k$ if k is the smallest number such that all graphs G of size k contain either a m-*CLIQUE *or an n-*INDEPENDENTSET.

If we want to check this property for a given k, we have to enumerate all possible graphs of size k and check whether a m-CLIQUE or an n-INDEPENDENTSET is included. Let us consider the complexity of this rather naive approach. First, we begin with the number of different undirected graphs of size k. We have $\frac{n(n-1)}{2}$ potential edges. Each of them can either exist or not. Hence, for the number of different undirected graphs of size k the following holds:

$$\text{number of different graphs of size } k = 2^{\left(\frac{n(n-1)}{2}\right)} \tag{5.1}$$

Next, one has to check whether these graphs have a m-Clique or an n-IndependentSet. Then, given a graph G, one has to check, whether K_m or I_n is isomorphic to a subgraph of G.

Definition 29 (Sub-Graph Isomorphism).

Instance: *Given are two graphs G_1 and G_2.*

Query: *Is G_1 isomorphic to an arbitrary subgraph of G_2?*

This problem is known to be **NP**-complete. Let us summarize the previous observations. The problem of finding $R(m, n)$ turns out—at best in such a direct approach—to require an exponential number of instances of the **NP**-complete problem. This fact will be addressed a little bit later in this chapter.

We can express the problem in another way, too: In lieu of talking about a property that has to hold for every subgraph, we can equivalently say that there is no subgraph that violates the property.

In the following, we use the abbreviations

- $C(\langle G, m\rangle) \equiv G$ has a m-Clique,
- $I(\langle G, n\rangle) \equiv G$ has an n-IndependentSet and
- $\mathcal{G}_k$ is the set of all graphs of size k.

So, our problem can be formulated like:

$$\forall G \in \mathcal{G}_k : \exists\, (C(\langle G, m\rangle) \vee I(\langle G, n\rangle)) \tag{5.2}$$

This is equivalent to

$$\nexists G \in \mathcal{G}_k : \neg\, (C(\langle G, m\rangle) \vee I(\langle G, n\rangle)) \tag{5.3}$$

An algorithm to find a minimal k can use this fact. It can ask, whether it is possible to find a graph of size k in which neither $C(\langle G, m\rangle)$ nor $I(\langle G, n\rangle)$ holds. If such a graph of size k exists, we know that $R(m, n) > k$. We repeat the test using each time a new $k' = k + 1$ until it is no longer possible to find such a graph. Then, we have found the minimal k that satisfies $R(m, n) = k$. From this idea we can state Algorithm 3 on the next page.

Algorithm 3 CoRamsey

```
Input: $\langle m, n \rangle$
Output: minimal $k$ such that $C(\langle G, m\rangle) \vee I(\langle G, n\rangle)$ holds for all graphs $G$ of size $k$.
$k \leftarrow 3$
$found \leftarrow$ TRUE
repeat
    for each graph $G$ of size $k$ do
        if not (SubGraphIso($K_m, G$) $\wedge$ SubGraphIso($I_n, G$)) then
            $found \leftarrow$ TRUE
            break
        else
            $found \leftarrow$ FALSE
        end if
    end for each
    if $found$ then
        $k \leftarrow k + 1$
    end if
until not $found$
return $k$
```

5.2.1 Complexity Considerations

Remarks on the algorithm's complexity

In a graph of size k, we have to consider all $\binom{k}{m}$ subgraphs of size m which can form a Clique, in the case of an IndependentSet $\binom{k}{n}$, respectively. When n is large enough, $n!$ can be estimated quite accurately using *Stirling*'s [39] approximation:

Sentence 1 (Stirling approximation). *(Without proof)*

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \cdot e^{\frac{1}{12n}} \tag{5.4}$$

When using Sentence 1, we can formulate the binomial coefficient in the following way:

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!} \leq \frac{\sqrt{n}\left(\frac{n}{e}\right)^n \left(\frac{n-k}{e}\right)^k e^{\frac{1}{12n}}}{\sqrt{2\pi k(n-k)}\left(\frac{n-k}{e}\right)^n \left(\frac{k}{e}\right)^k e^{\frac{1}{12(n-k)}} e^{\frac{1}{12k}}} \tag{5.5}$$

As an upper bound, we can say that $\binom{n}{k} \in \mathcal{O}(n^n)$. On Figure 5.1 on the next page, we recognize the rapid growth of the binomial coefficient. For our purpose, we have to consider both $\binom{k}{m}$ and $\binom{k}{n}$. Thus for the upper bound for the number of subgraphs

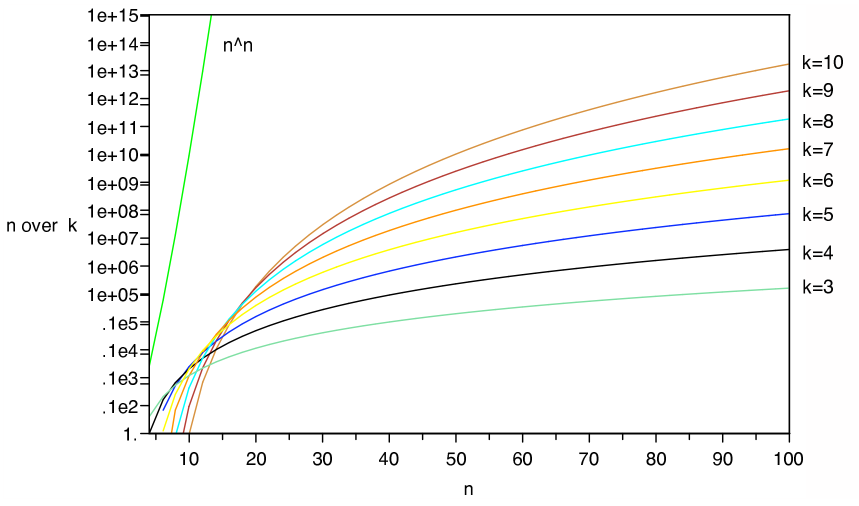


Figure 5.1: We can see $\binom{n}{k}$ for $1 \leq n \leq 100$ and $k = 3, 4, \ldots, 10$ on a logarithmic y-axis.

which form a CLIQUE or an INDEPENDENTSET, we obtain:

$$f(k,m,n) \leq \frac{1}{2\pi} \left(\frac{\sqrt{2\pi k}\left(\frac{k}{e}\right)^k e^{\frac{1}{12k}}}{\sqrt{m(k-m)}\left(\frac{m}{e}\right)^m e^{\frac{1}{12m}} \left(\frac{k-m}{e}\right)^{k-m} e^{\frac{1}{12(k-m)}}} + \frac{\sqrt{2\pi k}\left(\frac{k}{e}\right)^k e^{\frac{1}{12k}}}{\sqrt{n(k-n)}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n}} \left(\frac{k-n}{e}\right)^{k-n} e^{\frac{1}{12(k-n)}}} \right) \tag{5.6}$$

Remarks on the problem's complexity

The stated algorithm terminates in every case, because in [36] Ramsey proved that the number of people we have to invite to our party—or from the mathematical point of view, the minimal number of vertices k of a graph $G = (V, E)$ such that $C(\langle G, m\rangle)$ or $I(\langle G, n\rangle)$ holds—is finite. Thus, we *just* have to try consecutively each value for k.

Recall the formal definitions expressed in Formulae (5.2) and (5.3). For all $G \in \mathcal{G}_k$ either the relation $C(\langle G, m\rangle)$ or the relation $I(\langle G, n\rangle)$ holds. As we can see, we have a quantifier alternation $\forall - \exists$. Problems that are characterized in this way are located in Π_2^p which is on the second level of the *Polynomial Hierarchy* (**PH** for short). To define **PH**, we introduce briefly *oracle* computations: Up to now, the basis for all our considerations was that no computation whatsoever is for free. That is, we have to pay a certain amount of time and space in order to be able to run a computation. Let us assume for a moment, that it is possible to solve a number of problems without resource penalty. It might be possible to solve **NP** problems effortless with the help of some magic computational device, i.e., in just one computational step. With other words, we could use this powerful device as an *oracle* (or one can call it as well a very powerful sub-routine). But that is exactly what has already been mentioned above: The call of a sub-routine for a **NP** problem. This is, of course, a special case. Generally we can say that the complexity class of decision problems solvable by an algorithm in class **C** with an oracle for a problem in class **O** is written $\mathbf{C}^{\mathbf{O}}$, e.g., $\mathbf{P}^{\mathbf{NP}}$ is the set of problems which can be solved by a poly-time machine with access to an **NP** oracle. Oracles can be used to define the polynomial hierarchy.

Definition 30 (Polynomial Hierarchy). *The* polynomial hierarchy *is the following set of recursively defined classes: First,* $\Delta_0^P = \Sigma_0^P = \Pi_0^P = \boldsymbol{P}$. *For all* $i \geq 0$ *we have*

$$\begin{aligned} \Delta_{i+1}^P &:= \boldsymbol{P}^{\Sigma_i^P} \\ \Sigma_{i+1}^P &:= \boldsymbol{NP}^{\Sigma_i^P} \\ \Pi_{i+1}^P &:= \boldsymbol{coNP}^{\Sigma_i^P} \end{aligned}$$

In the case of Ramsey computation, we observe a quantifier alternation $\forall$–$\exists$. This translates directly into a $\mathbf{coNP}^{\mathbf{NP}}$ formulation. The $\forall$ quantification expresses that a property has to hold on any possible computation path, whereas the $\exists$ quantification says that there must exist at least one path with a special property. This is the same behavior of **coNP** and **NP**.

Hence, we know, that the Ramsey problem is in $\mathbf{coNP}^{\mathbf{NP}}$. For further details on the connection between the Polynomial Hierarchy and Ramsey numbers and other Ramsey types, refer to Marcus Schaefer [38].

5.2.2 Reduction to Sat

In the following, we are going to develop a straightforward translation from the decision problem Ramsey (m, n, k) to the satisfiability problem for Boolean formulae. Remark, that the corresponding Sat-formula is satisfiable if and only if it is possible to find a Graph G of size k which contains neither a m-Clique nor an n-IndependentSet. So, let us give for both properties an alternative characterization:

1. A graph G does not contain a m-Clique if and only if at least one edge is missing in each subgraph G' such that it is not isomorphic to K_m.
2. A graph G does not contain a n-IndependentSet if and only each subgraph G' contains at least one edge such that is is not isomorphic to I_n.

Both characterizations find their direct transformation into a CNF formula in the following way. We introduce variables $E(i, j)$ with $1 \leq i < j \leq |V|$. $E(i, j)$ is assigned TRUE if and only if in graph G there is an edge between vertices i to j. Let us begin with the translation of the first named property.

$$\begin{aligned}\neg E(1,2) \vee \neg E(1,3) \vee \ldots \vee \neg E(1,m) \vee \\ \neg E(2,3) \vee \ldots \vee \neg E(2,m) \vee \\ \vdots \\ \neg E(m-1,m)\end{aligned}$$

In short we can write:

$$\textsc{NonClique}\ (V', E) \equiv \bigvee_{i<j \in V'} \neg E(i,j) \qquad V' \subseteq V, |V'| = m \tag{5.7}$$

In the same manner, a CNF formula encoding a graph with n nodes including at least a single edge would be:

$$\begin{aligned}E(1,2) \vee E(1,3) \ldots \vee E(1,n) \vee \\ E(2,3) \vee \ldots \vee E(2,n) \vee \\ \vdots \\ E(n-1,n)\end{aligned}$$

And short:

$$\textsc{NonIndependentSet}\ (V', E) \equiv \bigvee_{i<j \in V'} E(i,j) \qquad V' \subseteq V, |V'| = n \tag{5.8}$$

Now, we have the characterization, that a graph of size m has no m-Clique and a graph of size n has no n-IndependentSet. In the next step, we have to put these pieces together.

$$\underset{|V|=k}{\exists} \langle V, E\rangle \bigwedge_{\substack{V' \subseteq V \\ |V'|=m}} \textsc{NonClique}\ (V', E) \wedge \bigwedge_{\substack{V' \subseteq V \\ |V'|=n}} \textsc{NonIndependentSet}\ (V', E) \tag{5.9}$$

This formula is satisfiable if and only if graph G neither contains a m-Clique nor an n-IndependentSet. At this point—again, a mapping between variables $E(i,j), 1 \leq i,j \leq |V|$ and integers has to be provided. We use $f(E(i,j)) = (j-1) \cdot |V| + i$ and correspondingly

$$\alpha(E(i,j)) = \begin{cases} E(i,j) \bmod |V| & \text{if } E(i,j) \pmod{|V|} \neq 0 \\ |V| & \text{else.} \end{cases} \tag{5.10}$$

$$\beta(E(i,j)) = \left\lceil \frac{E(i,j)}{|V|} \right\rceil \tag{5.11}$$

The process of encoding, back-transforming and analyzing of the SAT-solver's result works in the same way as already described in the previous two chapters.

The following example shows the resulting clause set for $R(3,3)$.

Example 7. *Let us start with $k = 4$, smaller k are trivial as we shall see later. k denotes the number of vertices the graph should have. Hence $G = (V, E)$ with $V = \{1, 2, 3, 4\}$. An instance for this problem consists of the two values for m and n and the just mentioned k. Our reduction provides the following CNF formula:*

`p cnf 16 8`	$\leftarrow$ *CNF formula with 16 variables and 8 clauses*
`-5 -9 -10 0`	$\leftarrow V' = \{1,2,3\}$ *is not a clique*
`-5 -13 -14 0`	$\leftarrow V' = \{1,2,4\}$ *is not a clique*
`-9 -13 -15 0`	$\leftarrow V' = \{1,3,4\}$ *is not a clique*
`-10 -14 -15 0`	$\leftarrow V' = \{2,3,4\}$ *is not a clique*
`5 9 10 0`	$\leftarrow V' = \{1,2,3\}$ *is not an independent set*
`5 13 14 0`	$\leftarrow V' = \{1,2,4\}$ *is not an independent set*
`9 13 15 0`	$\leftarrow V' = \{1,3,4\}$ *is not an independent set*
`10 14 15 0`	$\leftarrow V' = \{2,3,4\}$ *is not an independent set*

The first four lines after the problem definition represent the NONCLIQUE (V', E) *part, whereas the last four lines define the* NONINDEPENDENTSET (V', E) *part.*

*`limmat` for example, returns the following satisfying assignment for the given CNF formula: `-5 -9 10 13 -14 -15`. When transforming this result back, we obtain, that $E(2,3)$ and $E(1,4)$ are contained in G, other edges are not present. In this case, other results are possible. When repeating these two steps with $k = 5$, a satisfying assignment computed by `limmat` is `-6 -11 12 16 -17 18 21 22 -23 -24` with $E(1,4)$, $E(1,5)$, $E(2,3)$, $E(2,5)$ and $E(3,4)$ as available edges. As a consequent next step, we use $k = 6$. But now, it is not possible to find a satisfying assignment for the CNF formula. Thus we know that $k = 6$ is the number of vertices such that for each such graph of size 6 it is always possible to find either a m-*CLIQUE *or an n-*INDEPENDENTSET *and we are done. Figure 5.2 on the facing page shows the counterexamples generated by the* SAT*-solver. Image (a) depicts a graph with 4 vertices, image (b) on the right hand side a graph with 5 vertices. Indeed, both graphs show the existence of a k such that neither the* 3-CLIQUE *nor the* 3-INDEPENDENTSET *condition are satisfied.*

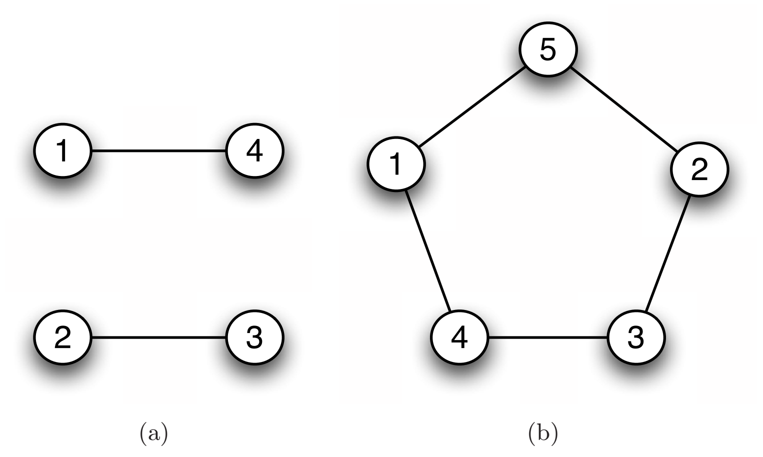


Figure 5.2: (a) Counterexamples for the assumptions $R(3,3)=4$ and (b) $R(3,3)=5$.

So the general procedure is as follows: We begin testing with a small k and check whether the corresponding Sat formula, as above mentioned, is satisfiable or not. If we get a positive assignment, we try the next greater $k'=k+1$ until we find a k such that Ramsey (m,n,k) does not hold and then $R(m,n)=k$. To minimize the number of necessary tests, one should start with good lower (and upper) bounds. Therefore, we are using some observations:

$$R(m,n) = R(n,m) \text{ is true by symmetry.} \tag{5.12}$$
$$R(m,1) = R(1,n) = 1 \tag{5.13}$$
$$R(m,2) = m \tag{5.14}$$
$$R(m,n) \leq R(m-1,n)+R(m,n-1) \text{ with strict inequality when both terms on the right hand side are even [20]} \tag{5.15}$$
$$R(m,m) \leq 4R(m,m-2)+2 \quad [43] \tag{5.16}$$
$$R(m,n) \geq R(m,n-1)+2m-3 \text{ for } m,n\geq 3 \ [7] \tag{5.17}$$

These observations alone are sufficient to solve $R(3,3)$ as the next example shows.

Example 8.

$$\begin{aligned} R(3,3) &\overset{(5.16)}{\leq} 4R(3,1)+2 \\ &\overset{(5.13)}{\leq} 4\cdot 1+2 \\ &\leq 6 \end{aligned}$$

On the other hand:

$$\begin{aligned} R(3,3) &\overset{(5.17)}{\geq} R(3,2) + 2 \cdot 3 - 3 \\ &\overset{(5.14)}{\geq} 3 + 2 \cdot 3 - 3 \\ &\geq 6 \end{aligned}$$

Combining both results yields:

$$6 \leq R(3,3) \leq 6 \tag{5.18}$$

In general, it is not sufficient to apply these (in)equalities to determine the desired Ramsey number $R(m,n)$. Nevertheless, we can use them to restrict the interval, in which $R(m,n)$ *has* to be.

5.3 Algorithm to determine Bounds for $R(m,n)$

Using Formulae (5.12) to (5.17), a recursive algorithm, which gives bounds for $R(m,n)$, can be stated. Obviously, for smaller Ramsey numbers, there are either exact values for $R(m,n)$, or there exist at least quite sharp bounds [35] on the possible values of $R(m,n)$. Algorithm 4 on the next page implements the above introduced (in)equalities in the form of a recursive procedure. Hereby, $R_{LB}(m,n)$ provides the lower bound and $R_{UB}(m,n)$ the upper bound of the interval, in which the correct result for $R(m,n)$ is. It is obvious, that this simple algorithm is not capable to provide as tight results as stated in [35] and reproduced in Table 5.1 on page 52. However this algorithm can be used to obtain bounds for even large values for $m,n \geq 15$ at ease. Table 5.2 on page 53 summarizes the bounds for the same combinations of m and n obtained by applying Algorithm 4. We can even improve these bounds by adding the already known optimal values as hard coded values into our algorithm. These better bounds are given in Table 5.3 on page 54.

For bounds with larger m,n, we refer to Appendix C on page 83.

5.4 Graphs with neither a Clique nor an IndependentSet

In the last section, we introduced both, an algorithm to determine the exact value of the Ramsey number $R(m,n)$ and an algorithm to obtain bounds for $R(m,n)$. In this section, we give examples for graphs, whose number of vertices is set to $R(m,n)-1$, i.e. these graphs have neither a m-Clique nor an n-IndependentSet. For small instances, i.e., $m,n \in \{3,4\}$ we can draw the graph in a clearly arranged way. In images (a) and (b) on Figure 5.3 on page 55 we can see example graphs for $G_{(a)} =$

Algorithm 4 RamseyBounds

```
Require: $m, n \geq 1$
 1: procedure $R_{UB}(m, n)$
 2:     if $((m = 1)||(n = 1))$ then
 3:         return $1$
 4:     end if
 5:     if $(m = 2)$ then
 6:         return $n$
 7:     end if
 8:     if $(n = 2)$ then
 9:         return $m$
10:     end if
11:     if $(m = n)$ then
12:         return $4 \cdot R_{UB}(m, m - 2) + 2$
13:     end if
14:     return $R_{UB}(m - 1, n) + R_{UB}(m, n - 1)$
15: end procedure

Require: $m, n \geq 1$
16: procedure $R_{LB}(m, n)$
17:     if $((m = 1)||(n = 1))$ then
18:         return $1$
19:     end if
20:     if $(m = 2)$ then
21:         return $n$
22:     end if
23:     if $(n = 2)$ then
24:         return $m$
25:     end if
26:     return $4 \cdot R_{LB}(m, m - 1) + 2 \cdot m - 3$
27: end procedure
```

m,n	3	4	5	6	7	8	9	10	11	12	13	14	15
3	6	9	14	18	23	28	36	40	46	52	59	66	73
								43	51	59	69	78	88
4		18	25	35	49	56	73	92	97	128	133	141	153
				41	61	84	115	149	191	238	291	349	417
5			43	58	80	101	125	143	159	185	209	235	265
			49	87	143	216	316	442		848		1461	
6				102	113	127	169	179	253	262	317		401
				165	298	495	780	1171		2566		5033	
7					205	216	233	289	405	416	511		
					540	1031	1713	2826	4553	6954	10581	15263	22116
8						282	317				817		861
						1870	3583	6090	10630	16944	27490	41525	63620
9							565	580					
							6588	12677	22325	39025	64871	89203	
10								798					1265
								23556		81200			

Table 5.1: Values and bounds for two color Ramsey numbers $R(m, n) = R(m, n; 2)$ [35].

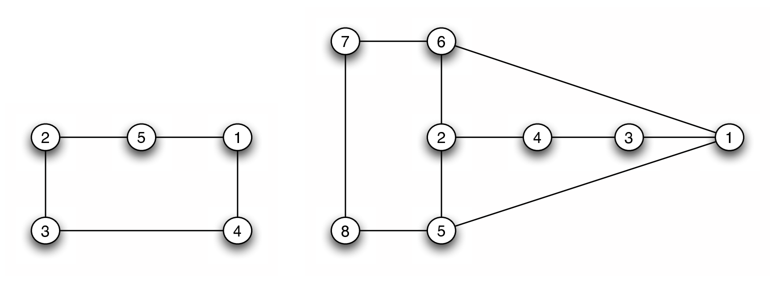
m,n	3	4	5	6	7	8	9	10	11	12	13	14	15
3	6	9	12	15	18	21	24	27	30	33	36	39	42
	6	10	15	21	28	36	45	55	66	78	91	105	120
4		14	19	24	29	34	39	44	49	54	59	64	69
		18	33	54	82	118	163	218	284	362	453	558	678
5			26	33	40	47	54	61	68	75	82	89	96
			62	116	198	316	479	697	981	1343	1796	2354	3032
6				42	51	60	69	78	87	96	105	114	123
				218	416	732	1211	1908	2889	4232	6028	8382	11414
7					62	73	84	95	106	117	128	139	150
					794	1526	2737	4645	7534	11766	17794	26176	37590
8						86	99	112	125	138	151	164	177
						2930	5667	10312	17846	29612	47406	73582	111172
9							114	129	144	159	174	189	204
							10950	21262	39108	68720	116126	189708	300880
10								146	163	180	197	214	231
								41250	80358	149078	265204	454912	755792

Table 5.2: Bounds for $R(m,n)$ calculated by Algorithm 4 on page 51

m,n	3	4	5	6	7	8	9	10	11	12	13	14	15
3	6	9	14	18	23	28	36	39	42	45	48	51	54
	6	9	14	18	23	28	36	46	57	69	82	96	111
4		18	25	30	35	40	45	50	55	60	65	70	75
		18	25	43	66	94	130	176	233	302	384	480	591
5			32	39	46	53	60	67	74	81	88	95	102
			58	101	167	261	391	567	800	1102	1486	1966	2557
6				45	54	63	72	81	90	99	108	117	126
				174	341	602	993	1560	2360	3462	4948	6914	9471
7					67	78	89	100	111	122	133	144	155
					670	1272	2265	3825	6185	9647	14595	21509	30980
8						93	106	119	132	145	158	171	184
						2410	4675	8500	14685	24332	38927	60436	91416
9							126	141	156	171	186	201	216
							9062	17562	32247	56579	95506	155942	247358
10								146	163	180	197	214	231
								34002	66249	122828	218334	374276	621634

Table 5.3: Better bounds for $R(m, n)$ calculated by Algorithm 4 on page 51 by adding knowledge about already known values.

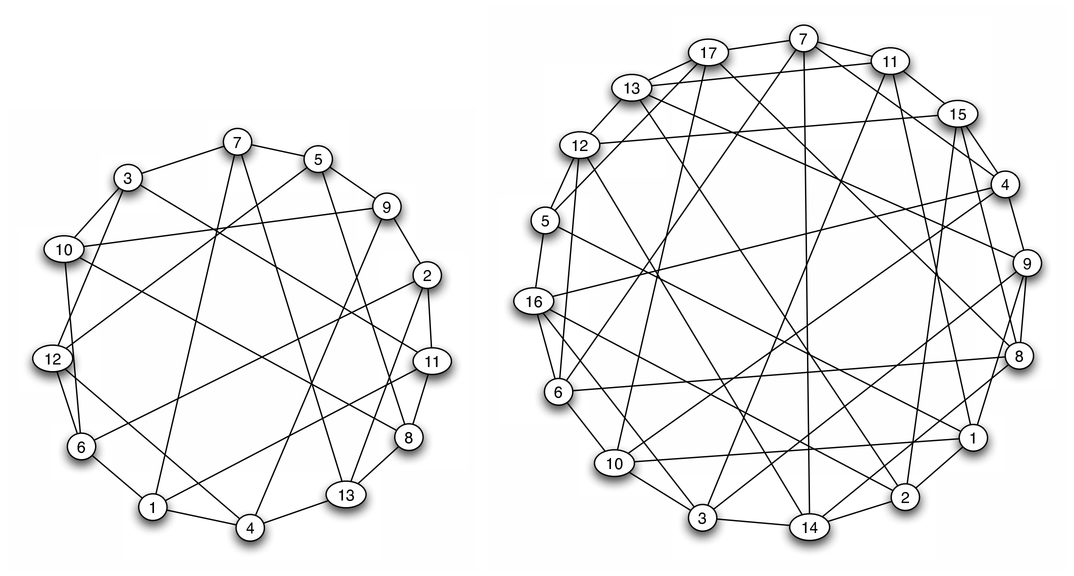
$(V_{(a)}, E_{(a)})$ with $|V_{(a)}| = 5 < R(3,3) = 6$ and $G_{(b)} = (V_{(b)}, E_{(b)})$ with $|V_{(b)}| = 8 < R(3,4) = 9$. For larger values for m and n, we give examples seen on Figure 5.4,



(a) $|V| = 5 < R(3,3) = 6$ (b) $|V| = 8 < R(3,4) = 9$

Figure 5.3: Graphs of small size without a CLIQUE and an INDEPENDENTSET of corresponding size.

too. However, these graphs are quite large and thus they cannot be visualized as concisely.



(a) $|V| = 13 < R(3,5) = 14$ (b) $|V| = 17 < R(3,6) = 18$

Figure 5.4: Graphs of larger size without a CLIQUE and an INDEPENDENTSET of corresponding size.

5.5 Estimation of the Instance Size

In the following, we give a short estimation of the size of an instance of Ramsey (m, n, k). We know that an undirected graph G of size k has $\binom{k}{m}$ subgraphs of size m. Each of them has a subset of up to $\frac{m(m-1)}{2}$ edges. In the above introduced reduction, we use for each edge a single variable $E(i, j)$. Each subgraph is encoded in the CNF file as a clause having $\frac{m(m-1)}{2}$ literals. We can give as a first estimation of the size of the CNF file the number of used literals occurrences:

$$\text{number of literals} = \binom{k}{m} \cdot \frac{m(m-1)}{2} + \binom{k}{n} \cdot \frac{n(n-1)}{2} \tag{5.19}$$

In CNF, literals are separated by a space symbol and clauses are separated by "`0`". Thus, we have for each clause $\frac{m(m-1)}{2} + 1$ additional bytes ($\frac{m(m-1)}{2}$ for the spaces and one for the separating "`0`"). As we said, edges are encoded by integer values. The largest occurring value can be k^2. As we present an upper bound for the file size, let us assume that each literal has the value k^2. The representation of integer value k^2 needs $\log_{10} k^2$ bytes. To encode negative literals, an additional "`-`" sign is needed. Together, we obtain:

$$\binom{k}{m}\left[\frac{m(m-1)}{2}\left(\log_{10} k^2 + 2\right) + 1\right] + \binom{k}{n}\left[\frac{n(n-1)}{2}\left(\log_{10} k^2 + 1\right) + 1\right] \tag{5.20}$$

The following Figure 5.5 on the facing page shows the dramatic growth of the instance file size. Note, the y-axis is in GiB-scale!

5.6 Experimental Results

In this section, we use our Sat-based approach to find the value for the Ramsey number $R(m, n)$. This approach is feasible at least for small values for m and n. On Figure 5.6 on page 58, we can see the run-times needed to find the exact value for $R(m, n)$. For these experiments, we use `minisat` on *iMac*. As we can see, the major part of the overall run-time is always the last step, i.e., to show that it is not possible to find a graph with $R(m, n)$ vertices with no m-Clique and no n-IndependentSet.

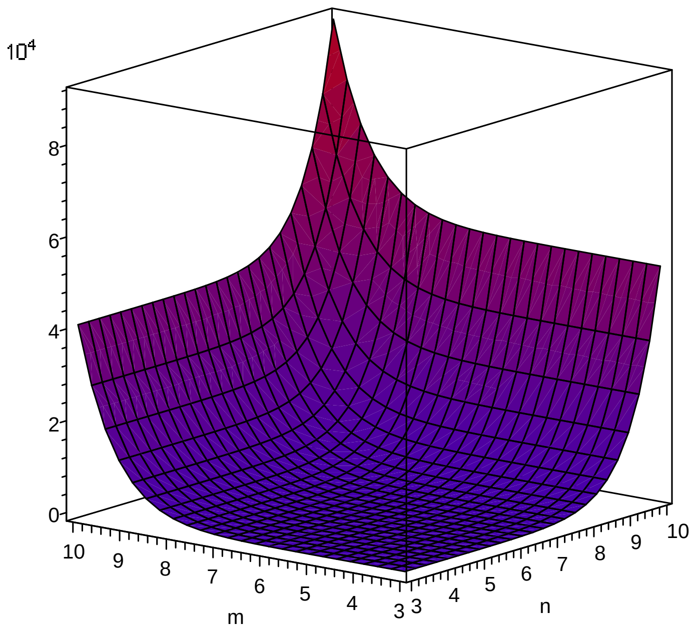
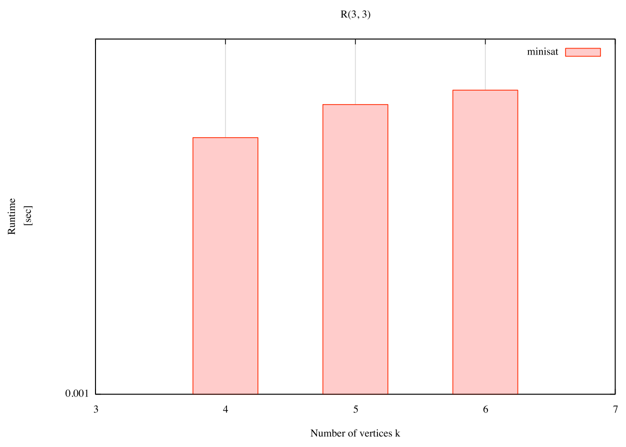
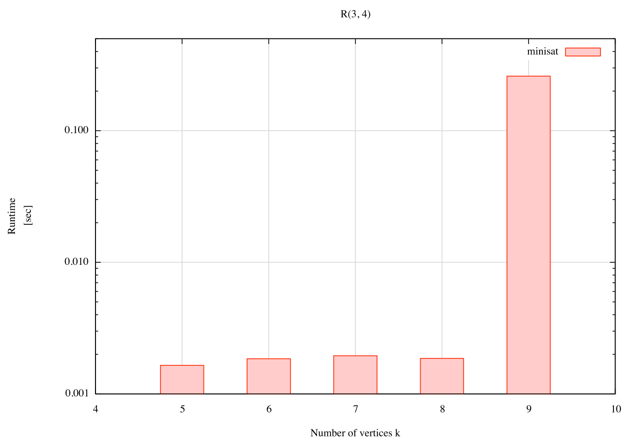


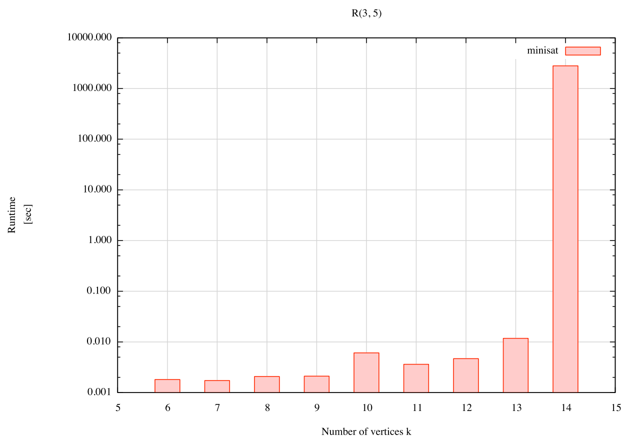
Figure 5.5: Ramsey instance files size for $k = 43$ and $3 \leq m, n \leq 10$.



(a) $R(3,3) = 6$



(b) $R(3,4) = 9$



(c) $R(3,5) = 14$

Figure 5.6: Run-times to find small Ramsey numbers $R(m, n)$.

Chapter 6

Conclusion and Future Work

The goal of the presented thesis was the evaluation of SAT as a general search technique for arbitrary **NP**-hard search and optimization problems. In the course of this thesis, we elaborated our methods on three example problems, namely graph coloring, inverse RNA folding, and the Ramsey numbers. Because such problems appear in thousands of real world applications, it is important to have tools at hand to solve them generically and efficiently—exactly this generality and efficiency is available in terms of today's highly optimized SAT-solvers. As these problems occur in totally different disciplines from biology, chemical synthesis, medical applications going to problems in physics, statistics, finance and many more, we chose our three problems each from a different discipline.

In the case of graph coloring and finding the chromatic number of a graph, our approach using binary search turned out to be very promising. We were able to find some new chromatic numbers of graph instances, introduced in the second DIMACS Implementation Challenge in 1992. We presented these results as well as new bounds for instances whose chromatic number is still unknown. Besides those new bounds and exact solutions for chromatic numbers, we were also able to verify a number of already known results. In particular, we were able to recompute many of these bounds within fractions of a second—confirming thereby the practical applicability of our generic approach.

Finding a solution for the inverse RNA folding problem with SAT-solvers proved to be practically successful. In general there is more than one feasible solution for a given RNA secondary structure. In this case, we are interested in a solution with a certain property, namely with a so-called minimal energy level. This energy level is obtained by a simplified energy model. This has been done using PSEUDO BOOLEAN OPTIMIZATION. A Solution for the yeast tRNA^{Phe} could be found within a second. This method shall be used in future experiments for larger RNA secondary structures to prove its capability.

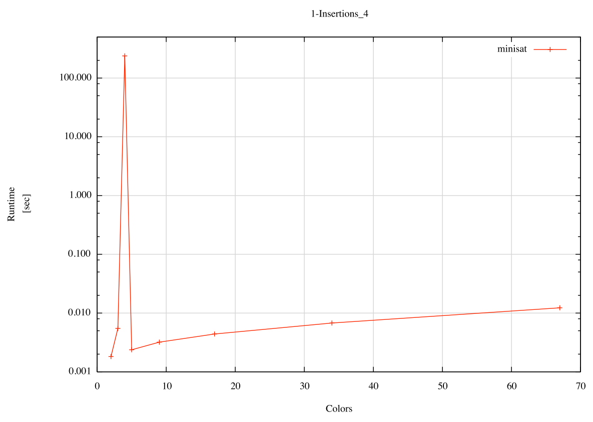
As the last application area for SAT-solvers analyzed in this thesis we chose the Ramsey numbers $R(m, n)$. It turned out, that this problem sets limits to the applicability of SAT-solvers. Only instances with very small values for m and n could be solved. To achieve further progress, one could try to optimize the encoding into a smaller SAT formula, or one could use random edge assignments to try to break symmetry. In this case, not only extremely long run-times, but also high memory consumption proved to be limiting the applicability of the approach.

We believe, that the resource efficient usage of clusters is a key to further proliferate the broad and generic application of SAT decision procedures as fundamental search technology. While highly promising, current parallel SAT-solvers are not capable to outperform solvers running on a single processor and therefore it is a challenge for further research and development to provide techniques for parallel SAT-solvers. The common availability of dual and quad core processors and prototypes with 80 cores and even more stresses the increasing importance of viable parallel and multi-threaded SAT-solvers.

Appendix A

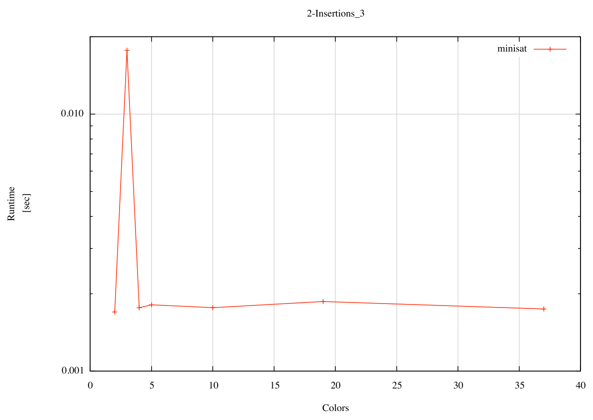
Chromatic Numbers

This chapter summarizes our experimental results on finding chromatic numbers for hard graph instances. We give for each benchmarking the found solution and the run-times during the binary-search steps in graphical as well as in tabular form.



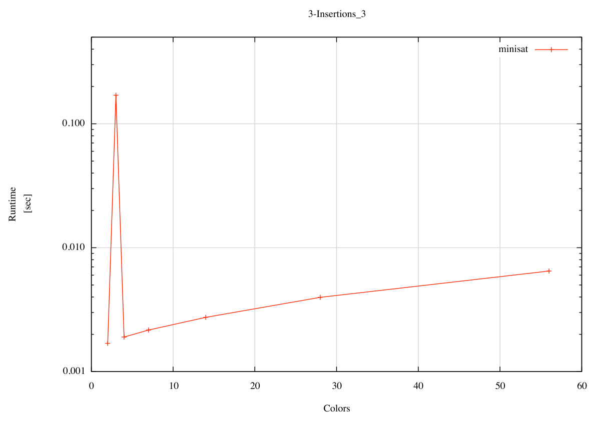
status	colors	`minisat` [sec]
SAT	67	0.012288
SAT	34	0.006770
SAT	17	0.004404
SAT	9	0.003195
SAT	5	0.002372
UNSAT	4	238.080000
UNSAT	3	0.005478

Figure A.1: *1-Insertions_4* is 5-colorable (computed on *iMac*)



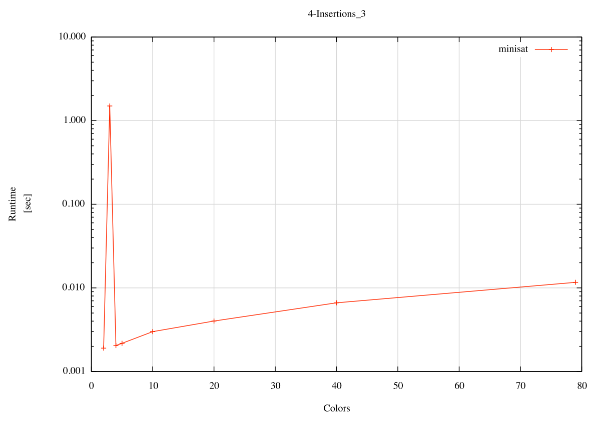
status	colors	`minisat` [sec]
SAT	37	0.001746
SAT	19	0.001866
SAT	10	0.001766
SAT	5	0.001812
SAT	4	0.001765
UNSAT	3	0.017713

Figure A.2: *2-Insertions_3* is 4-colorable (computed on *iMac*)



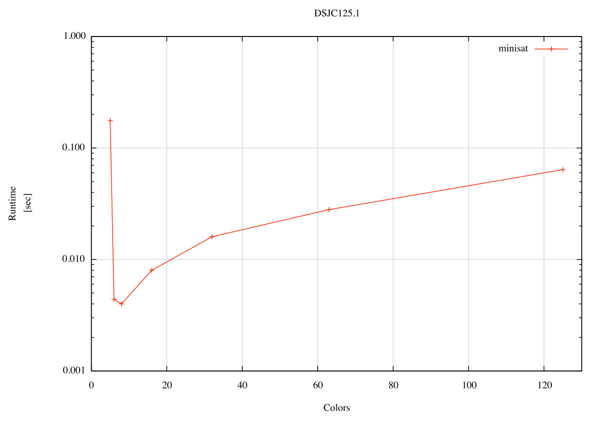
status	colors	minisat [sec]
SAT	56	0.006484
SAT	28	0.003976
SAT	14	0.002742
SAT	7	0.002166
SAT	4	0.001902
UNSAT	3	0.169933
UNSAT	2	0.001692

Figure A.3: *3-Insertions_3* is 4-colorable (computed on iMac)



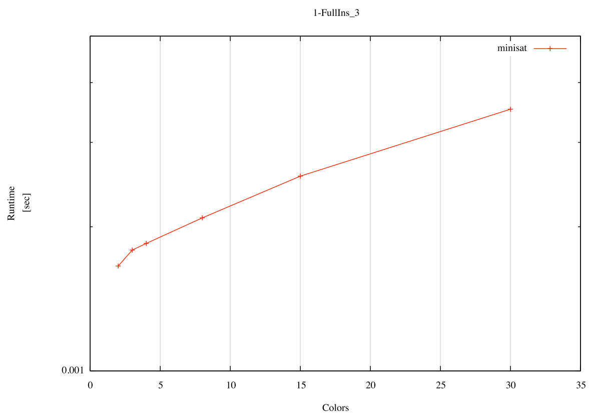
status	colors	minisat [sec]
SAT	79	0.011652
SAT	40	0.006631
SAT	20	0.004012
SAT	10	0.002996
SAT	5	0.002172
SAT	4	0.002043
UNSAT	3	1.497750

Figure A.4: *4-Insertions_3* is 4-colorable (computed on iMac)

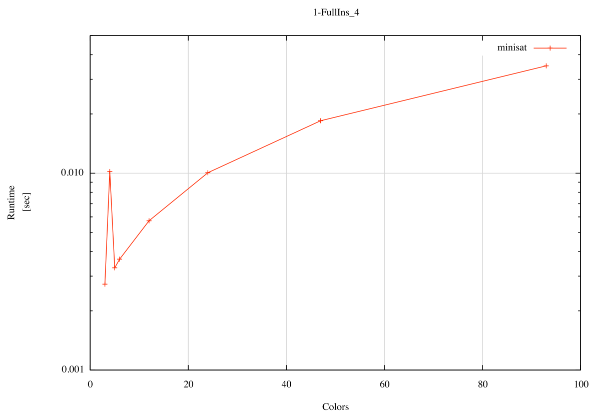


status	colors	minisat [sec]
SAT	125	0.064004
SAT	63	0.028001
SAT	32	0.016001
SAT	16	0.008000
SAT	8	0.004000
SAT	6	0.004396
SAT	5	0.176001
UNSAT	4	0.004000

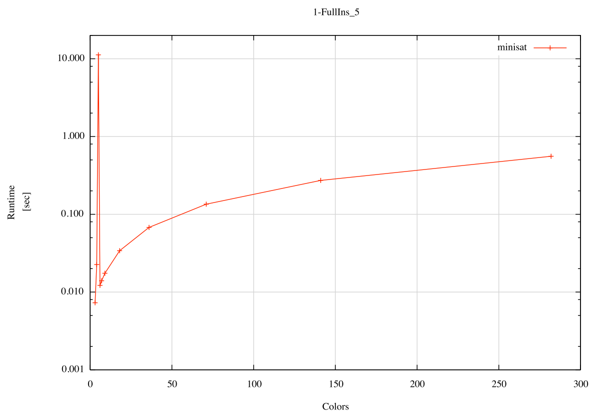
Figure A.5: *DSJC125.1* is 5-colorable (computed on *king*)



status	colors	`minisat` [sec]
SAT	30	0.003519
SAT	15	0.002551
SAT	8	0.002087
SAT	4	0.001846
UNSAT	3	0.001788
UNSAT	2	0.001656

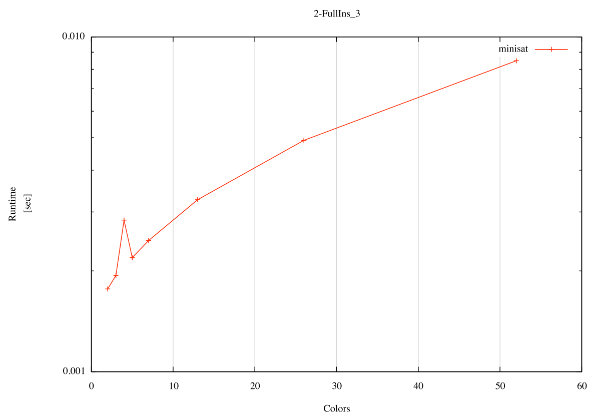


status	colors	`minisat` [sec]
SAT	93	0.035103
SAT	47	0.018479
SAT	24	0.010059
SAT	12	0.005733
SAT	6	0.003660
SAT	5	0.003308
UNSAT	4	0.010206
UNSAT	3	0.002731

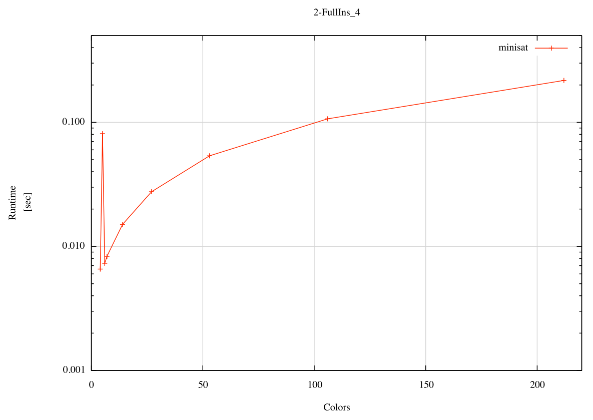


status	colors	`minisat` [sec]
SAT	282	0.557016
SAT	141	0.272599
SAT	71	0.134908
SAT	36	0.067766
SAT	18	0.034013
SAT	9	0.017436
SAT	7	0.013972
SAT	6	0.012195
UNSAT	5	11.237900

Figure A.6: From top to bottom: *1-FullIns_3* is 4-colorable, *1-FullIns_4* is 5-colorable and *1-FullIns_5* is 6-colorable (computed on *iMac*)

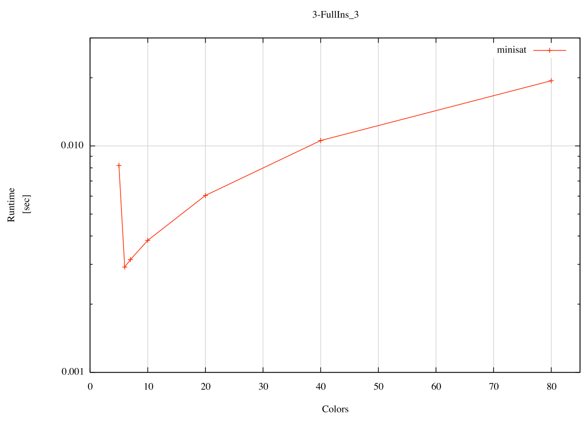


status	colors	`minisat` [sec]
SAT	52	0.008487
SAT	26	0.004909
SAT	13	0.003264
SAT	7	0.002465
SAT	5	0.002191
UNSAT	4	0.002837

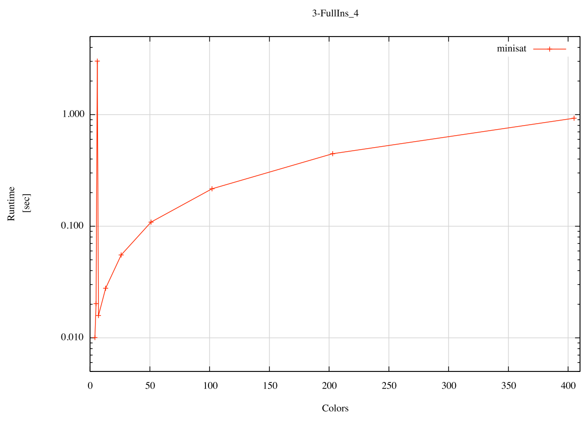


status	colors	`minisat` [sec]
SAT	212	0.217651
SAT	106	0.106713
SAT	53	0.053727
SAT	27	0.027580
SAT	14	0.015056
SAT	7	0.008321
SAT	6	0.007312
UNSAT	5	0.081009
UNSAT	4	0.006565

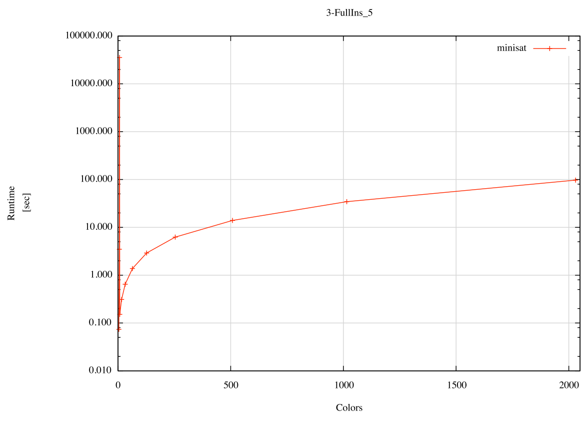
Figure A.7: From top to bottom: *2-FullIns_ 3* is 5-colorable and *2-FullIns_ 4* is 6-colorable (computed on *iMac*)



status	colors	`minisat` [sec]
SAT	80	0.019407
SAT	40	0.010563
SAT	20	0.006041
SAT	10	0.003827
SAT	7	0.003148
SAT	6	0.002916
UNSAT	5	0.008203

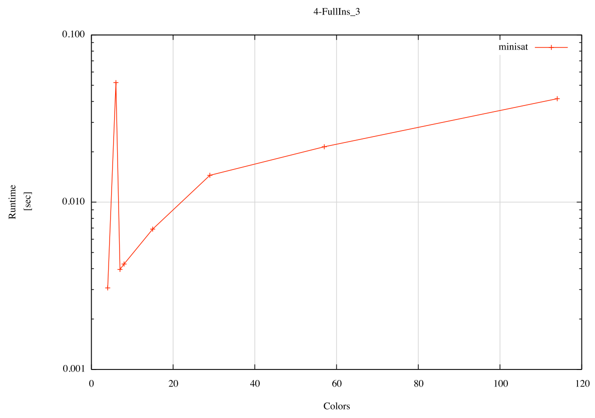


status	colors	`minisat` [sec]
SAT	405	0.928001
SAT	203	0.446317
SAT	102	0.216340
SAT	51	0.108868
SAT	26	0.055244
SAT	13	0.027829
SAT	7	0.015836
UNSAT	6	3.013120
UNSAT	5	0.020250
UNSAT	4	0.010050

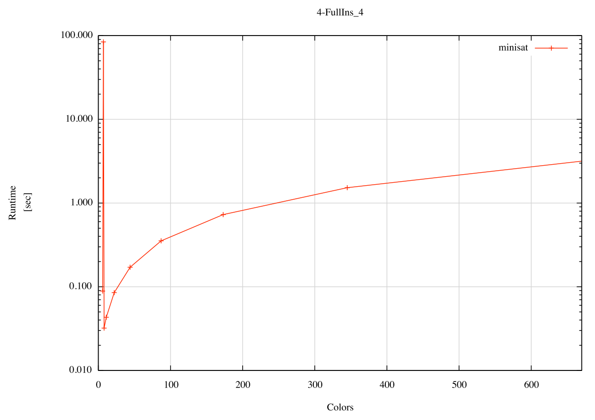


status	colors	`minisat` [sec]
SAT	2030	96.947400
SAT	1015	34.434200
SAT	508	13.929700
SAT	254	6.239500
SAT	127	2.899730
SAT	64	1.374400
SAT	32	0.647355
SAT	16	0.310409
SAT	8	0.151738
UNSAT	7	35579.000000
UNSAT	6	3.484290
UNSAT	4	0.073847

Figure A.8: From top to bottom: *3-FullIns_ 3* is 6-colorable, *3-FullIns_ 4* is 7-colorable and *3-FullIns_ 5* is 8-colorable (computed on *iMac*)

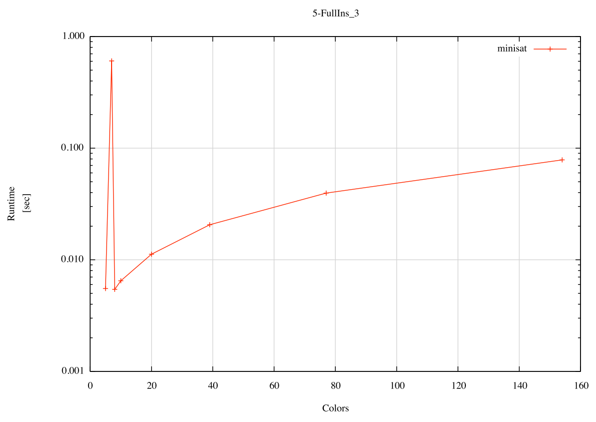


status	colors	`minisat` [sec]
SAT	114	0.041553
SAT	57	0.021437
SAT	29	0.014470
SAT	15	0.006897
SAT	8	0.004266
SAT	7	0.003960
UNSAT	6	0.051844
UNSAT	4	0.003070

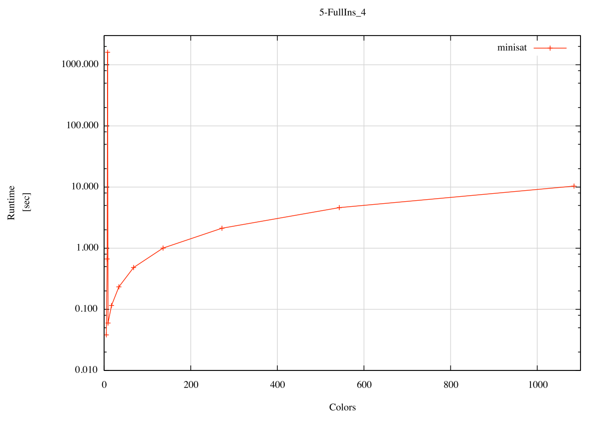


status	colors	`minisat` [sec]
SAT	690	3.312490
SAT	345	1.524620
SAT	173	0.727888
SAT	87	0.353407
SAT	44	0.171217
SAT	22	0.085097
SAT	11	0.043256
SAT	8	0.032056
UNSAT	7	84.190000
UNSAT	6	0.088792

Figure A.9: From top to bottom: *4-FullIns_3* is 7-colorable and *4-FullIns_4* is 8-colorable (computed on *iMac*)

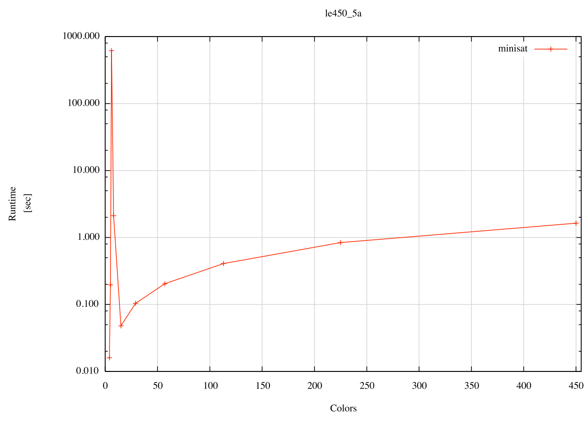


status	colors	`minisat` [sec]
SAT	154	0.078513
SAT	77	0.039564
SAT	39	0.020594
SAT	20	0.011220
SAT	10	0.006484
SAT	8	0.005430
UNSAT	7	0.604928
UNSAT	5	0.005523

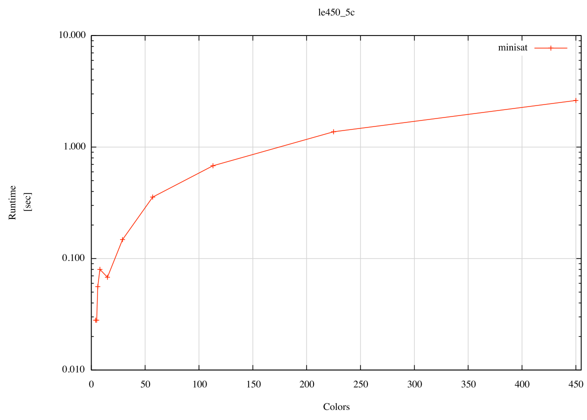


status	colors	`minisat` [sec]
SAT	1085	10.370000
SAT	543	4.608960
SAT	272	2.120130
SAT	136	1.003980
SAT	68	0.482396
SAT	34	0.233021
SAT	17	0.115439
SAT	9	0.059600
UNSAT	8	1600.710000
UNSAT	7	0.665806
UNSAT	5	0.038177

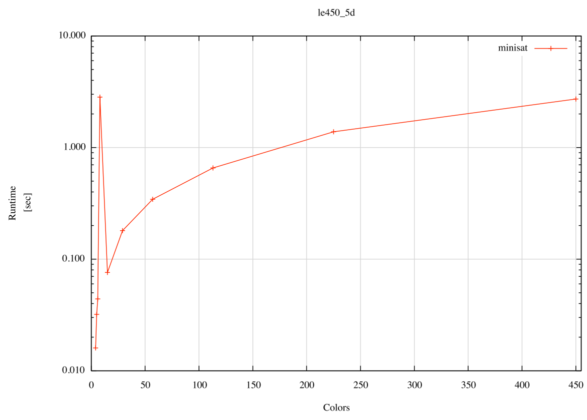
Figure A.10: From top to bottom: *5-FullIns_3* is 8-colorable and *5-FullIns_4* is 9-colorable (computed on *iMac*)



status	colors	`minisat` [sec]
SAT	450	1.636100
SAT	225	0.840052
SAT	113	0.408025
SAT	57	0.204012
SAT	29	0.104006
SAT	15	0.048003
SAT	8	2.116130
SAT	6	617.199000
SAT	5	0.196012
UNSAT	4	0.016001

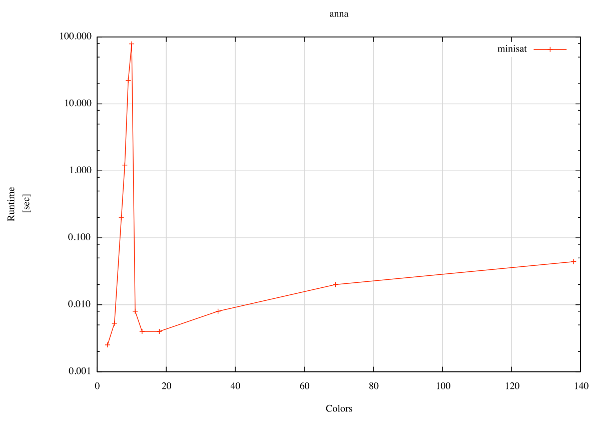


status	colors	`minisat` [sec]
SAT	450	2.620160
SAT	225	1.372080
SAT	113	0.680042
SAT	57	0.356022
SAT	29	0.148009
SAT	15	0.068004
SAT	8	0.080005
SAT	6	0.056003
SAT	5	0.028001
UNSAT	4	0.028001

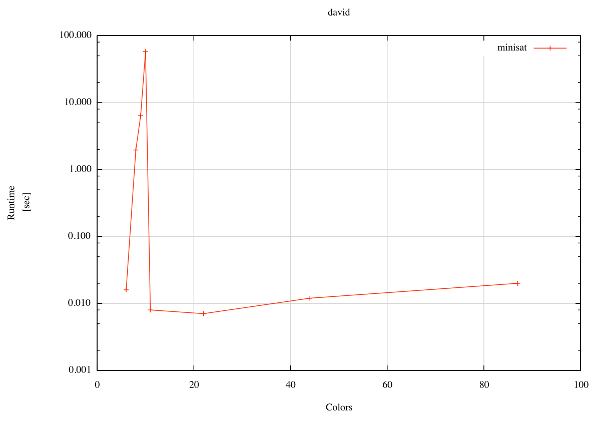


status	colors	`minisat` [sec]
SAT	450	2.720170
SAT	225	1.384090
SAT	113	0.656041
SAT	57	0.344021
SAT	29	0.180011
SAT	15	0.076004
SAT	8	2.828180
SAT	6	0.044002
SAT	5	0.032002
UNSAT	4	0.016001

Figure A.11: From top to bottom: *le450_5a* is 5-colorable, *le450_5c* is 5-colorable and *le450_5d* is 5-colorable (computed on *king*)

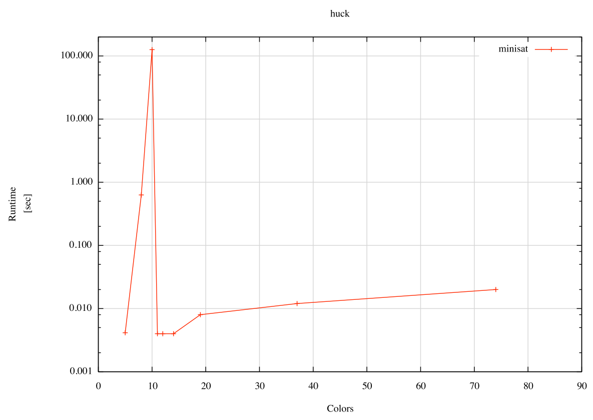


status	colors	`minisat` [sec]
SAT	138	0.044002
SAT	69	0.020001
SAT	35	0.008000
SAT	18	0.004000
SAT	13	0.004000
SAT	11	0.008000
UNSAT	10	78.652900
UNSAT	9	22.409400

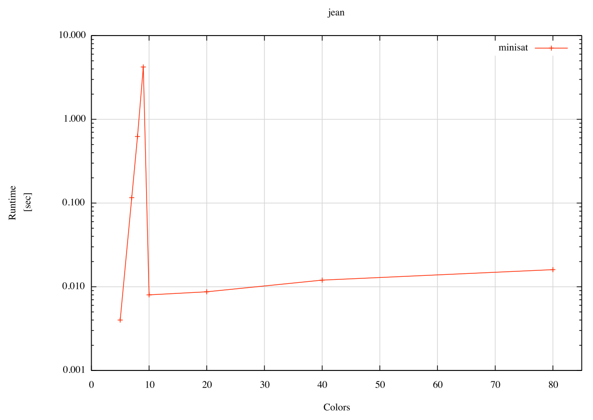


status	colors	`minisat` [sec]
SAT	87	0.020001
SAT	44	0.012000
SAT	22	0.007052
SAT	11	0.008000
UNSAT	10	57.715600
UNSAT	9	6.376400
UNSAT	8	1.960120
UNSAT	6	0.0160010

Figure A.12: From top to bottom: *anna* is 11-colorable and *david* is 11-colorable (computed on *guru*)

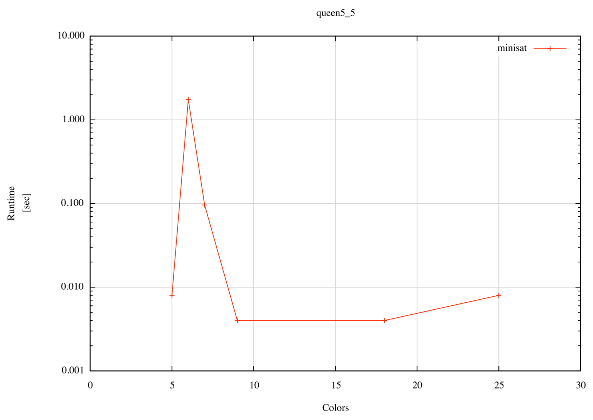


status	colors	`minisat` [sec]
SAT	74	0.020010
SAT	37	0.012000
SAT	19	0.008000
SAT	14	0.004000
SAT	12	0.004000
SAT	11	0.004000
UNSAT	10	125.620000

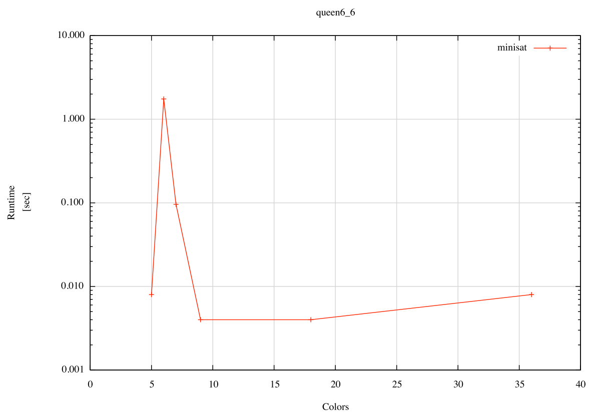


status	colors	`minisat` [sec]
SAT	80	0.016001
SAT	40	0.012000
SAT	20	0.008708
SAT	10	0.008000
UNSAT	9	4.224260
UNSAT	8	0.624039
UNSAT	7	0.116007
UNSAT	5	0.004000

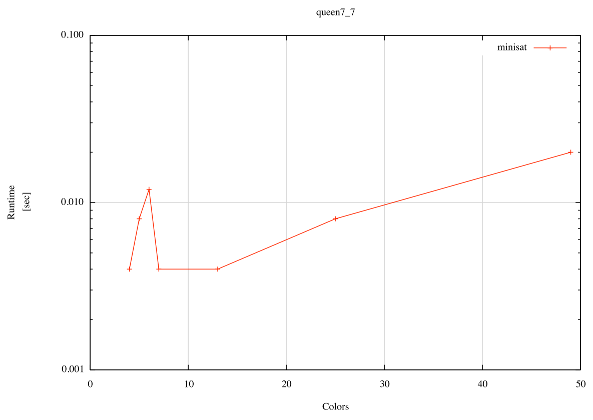
Figure A.13: From top to bottom: *huck* is 11-colorable and *jean* is 10-colorable (computed on *guru*)



status	colors	`minisat` [sec]
SAT	25	0.008000
SAT	13	0.004000
SAT	7	0.004000
SAT	5	0.004000
UNSAT	4	0.004000

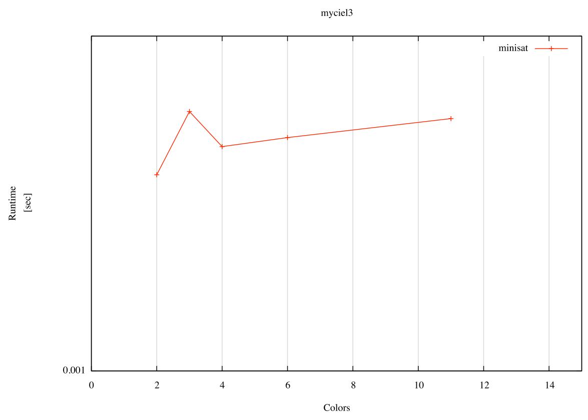


status	colors	`minisat` [sec]
SAT	36	0.008000
SAT	18	0.004000
SAT	9	0.004000
SAT	7	0.096006
UNSAT	6	1.748110
UNSAT	5	0.008000

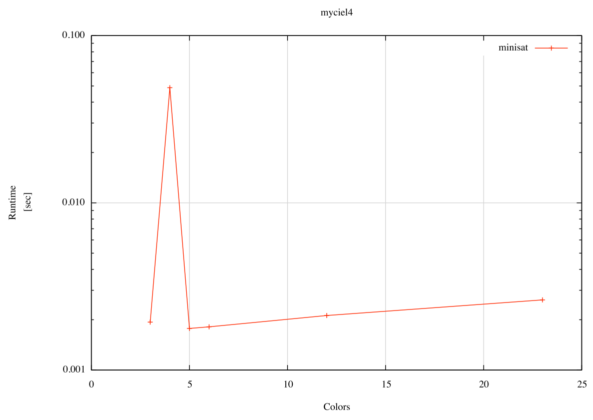


status	colors	`minisat` [sec]
SAT	49	0.020001
SAT	25	0.008000
SAT	13	0.004000
SAT	7	0.004000
UNSAT	6	0.012000
UNSAT	5	0.008000
UNSAT	4	0.004000

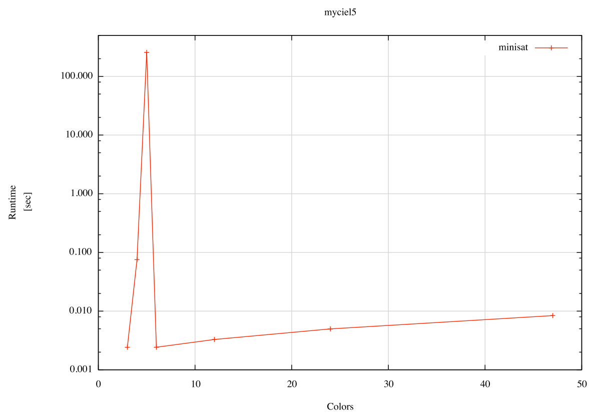
Figure A.14: From top to bottom: *queen5_ 5* is 5-colorable, *queen6_ 6* is 7-colorable and *queen7_ 7* is 7-colorable (computed on *guru*)



status	colors	`minisat` [sec]
SAT	11	0.001686
SAT	6	0.001621
SAT	4	0.001591
UNSAT	3	0.001711

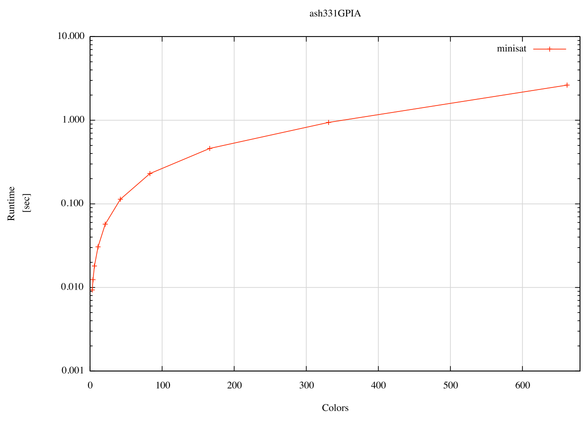


status	colors	`minisat` [sec]
SAT	23	0.002628
SAT	12	0.002120
SAT	6	0.001814
SAT	5	0.001774
UNSAT	4	0.048822
UNSAT	3	0.001935

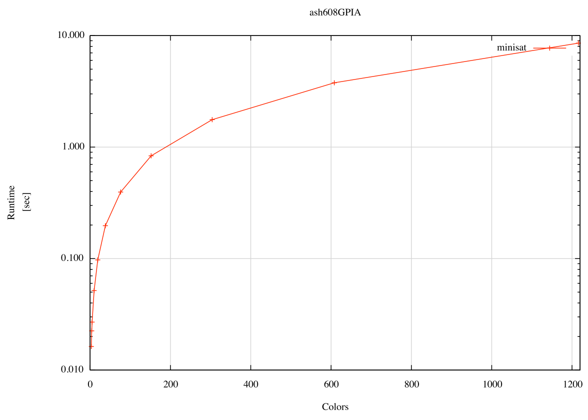


status	colors	`minisat` [sec]
SAT	47	0.008390
SAT	24	0.004976
SAT	12	0.003288
SAT	6	0.002424
UNSAT	5	256.559000
UNSAT	4	0.075293
UNSAT	3	0.002421

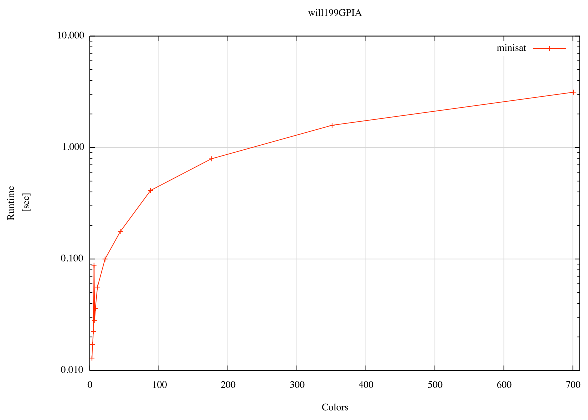
Figure A.15: From top to bottom: *myciel3* is 4-colorable, *myciel4* is 5-colorable and *myciel5* is 6-colorable (computed on *iMac*)



status	colors	`minisat` [sec]
SAT	662	2.631570
SAT	331	0.941934
SAT	166	0.460195
SAT	83	0.230059
SAT	42	0.113496
SAT	21	0.057146
SAT	11	0.030606
SAT	6	0.018060
SAT	4	0.012377
UNSAT	3	0.009342

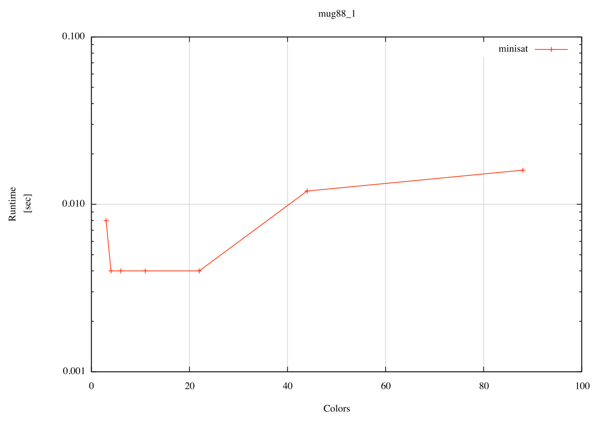


status	colors	`minisat` [sec]
SAT	1216	8.562010
SAT	608	3.775940
SAT	304	1.762980
SAT	152	0.835178
SAT	76	0.395088
SAT	38	0.197153
SAT	19	0.097316
SAT	10	0.051611
SAT	5	0.026939
SAT	4	0.022488
UNSAT	3	0.016267

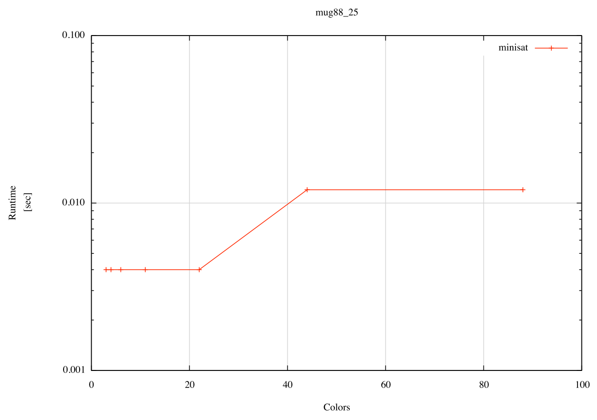


status	colors	`minisat` [sec]
SAT	701	3.136200
SAT	351	1.584100
SAT	176	0.792049
SAT	88	0.412025
SAT	44	0.176011
SAT	22	0.100006
SAT	11	0.056003
SAT	8	0.036002
SAT	7	0.028001
UNSAT	6	0.088005

Figure A.16: From top to bottom: *ash331GPIA* is 4-colorable, *ash608GPIA* is 4-colorable (both computed on *iMac*) and *will199GPIA* is 7-colorable (computed on *hand*)

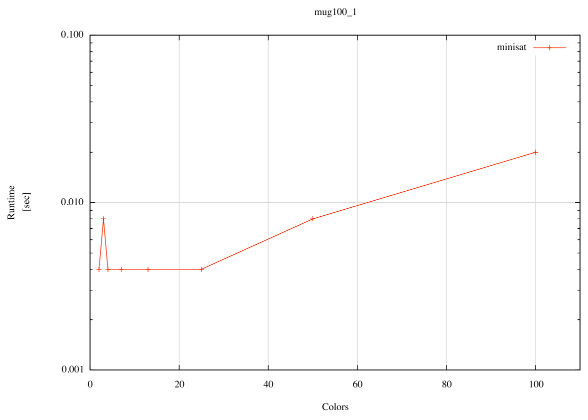


status	colors	`minisat` [sec]
SAT	88	0.016001
SAT	44	0.012000
SAT	22	0.004000
SAT	11	0.004000
SAT	6	0.004000
SAT	4	0.004000
UNSAT	3	0.008000

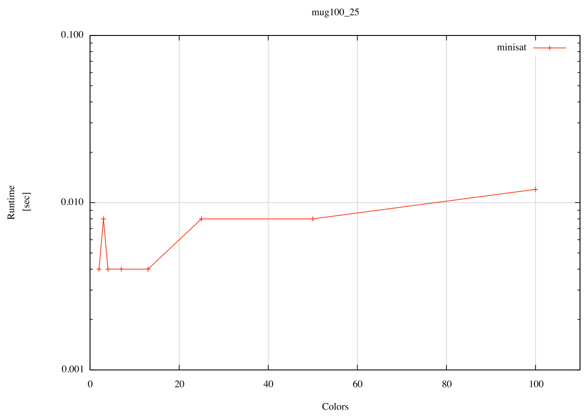


status	colors	`minisat` [sec]
SAT	88	0.012000
SAT	44	0.012000
SAT	22	0.004000
SAT	11	0.004000
SAT	6	0.004000
SAT	4	0.004000
UNSAT	3	0.004000

Figure A.17: From top to bottom: *mugg88_1* is 4-colorable and *mugg88_25* is 4-colorable (computed on *king*)

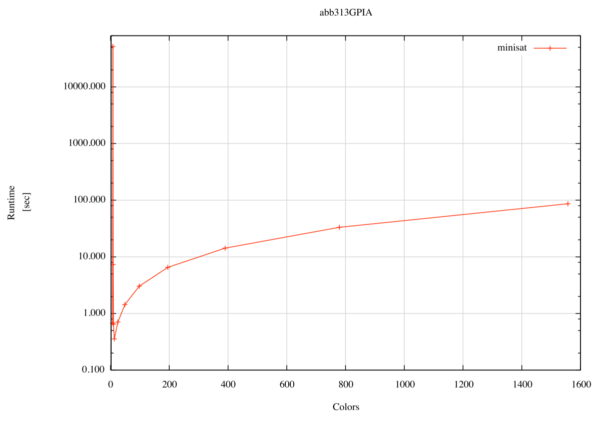


status	colors	`minisat` [sec]
SAT	100	0.020001
SAT	50	0.008000
SAT	25	0.004000
SAT	13	0.004000
SAT	7	0.004000
SAT	4	0.004000
UNSAT	3	0.008000
UNSAT	2	0.004000



status	colors	`minisat` [sec]
SAT	100	0.012000
SAT	50	0.008000
SAT	25	0.008000
SAT	13	0.004000
SAT	7	0.004000
SAT	4	0.004000
UNSAT	3	0.008000
UNSAT	2	0.004000

Figure A.18: From top to bottom: *mugg100_1* is 4-colorable and *mugg100_25* is 4-colorable (computed on *king*)



status	colors	`minisat` [sec]
SAT	1557	86.522000
SAT	779	33.209600
SAT	390	14.239800
SAT	195	6.527260
SAT	98	3.038340
SAT	49	1.437560
SAT	25	0.706871
SAT	13	0.356490
SAT	10	0.648647
SAT	9	7.321030
UNSAT	8	51974.900000
UNSAT	7	0.677760

Figure A.19: *abb313GPIA* is 9-colorable (computed on *iMac*)

Appendix B

RNA Encodings

In the following two sections, we see the transformation results after applying the respective reduction to the input instance. First we use tRNA$^{\mathrm{Phe}}$ as input to obtain a corresponding CNF representation.

```
(((((((..((((........)))).(((((.......))))).....(((((.......))))))))))))....
```

Since the description of tRNA$^{\mathrm{Phe}}$ in PBO format is too large to be imprinted in this thesis we use the multiple smaller example:

```
(())
```

B.1 Example CNF Encoding for the tRNA$^{\mathrm{Phe}}$ Molecule

```
p cnf 288 462       165 166 167 168 0  265 266 267 268 0  -51 -52 0
49 50 51 52 0       105 106 107 108 0  17 18 19 20 0      -85 -86 0
85 86 87 88 0       169 170 171 172 0  269 270 271 272 0  -85 -87 0
45 46 47 48 0       209 210 211 212 0  13 14 15 16 0      -85 -88 0
89 90 91 92 0       241 242 243 244 0  273 274 275 276 0  -86 -87 0
41 42 43 44 0       205 206 207 208 0  9 10 11 12 0       -86 -88 0
93 94 95 96 0       245 246 247 248 0  277 278 279 280 0  -87 -88 0
37 38 39 40 0       201 202 203 204 0  5 6 7 8 0          -45 -46 0
97 98 99 100 0      249 250 251 252 0  281 282 283 284 0  -45 -47 0
121 122 123 124 0   197 198 199 200 0  1 2 3 4 0          -45 -48 0
153 154 155 156 0   253 254 255 256 0  285 286 287 288 0  -46 -47 0
117 118 119 120 0   193 194 195 196 0  -49 -50 0          -46 -48 0
157 158 159 160 0   257 258 259 260 0  -49 -51 0          -47 -48 0
113 114 115 116 0   25 26 27 28 0      -49 -52 0          -89 -90 0
161 162 163 164 0   261 262 263 264 0  -50 -51 0          -89 -91 0
109 110 111 112 0   21 22 23 24 0      -50 -52 0          -89 -92 0
```

```
-90 -91 0
-90 -92 0
-91 -92 0
-41 -42 0
-41 -43 0
-41 -44 0
-42 -43 0
-42 -44 0
-43 -44 0
-93 -94 0
-93 -95 0
-93 -96 0
-94 -95 0
-94 -96 0
-95 -96 0
-37 -38 0
-37 -39 0
-37 -40 0
-38 -39 0
-38 -40 0
-39 -40 0
-97 -98 0
-97 -99 0
-97 -100 0
-98 -99 0
-98 -100 0
-99 -100 0
-121 -122 0
-121 -123 0
-121 -124 0
-122 -123 0
-122 -124 0
-123 -124 0
-153 -154 0
-153 -155 0
-153 -156 0
-154 -155 0
-154 -156 0
-155 -156 0
-117 -118 0
-117 -119 0
-117 -120 0
-118 -119 0
-118 -120 0
-119 -120 0
-157 -158 0
-157 -159 0
-157 -160 0
-158 -159 0
-158 -160 0
-159 -160 0
-113 -114 0
-113 -115 0
-113 -116 0
-114 -115 0
-114 -116 0
-115 -116 0
-161 -162 0
-161 -163 0
-161 -164 0
-162 -163 0
-162 -164 0
-163 -164 0
-109 -110 0
-109 -111 0
-109 -112 0
-110 -111 0
-110 -112 0
-111 -112 0
-165 -166 0
-165 -167 0
-165 -168 0
-166 -167 0
-166 -168 0
-167 -168 0
-105 -106 0
-105 -107 0
-105 -108 0
-106 -107 0
-106 -108 0
-107 -108 0
-169 -170 0
-169 -171 0
-169 -172 0
-170 -171 0
-170 -172 0
-171 -172 0
-209 -210 0
-209 -211 0
-209 -212 0
-210 -211 0
-210 -212 0
-211 -212 0
-241 -242 0
-241 -243 0
-241 -244 0
-242 -243 0
-242 -244 0
-243 -244 0
-205 -206 0
-205 -207 0
-205 -208 0
-206 -207 0
-206 -208 0
-207 -208 0
-245 -246 0
-245 -247 0
-245 -248 0
-246 -247 0
-246 -248 0
-247 -248 0
-201 -202 0
-201 -203 0
-201 -204 0
-202 -203 0
-202 -204 0
-203 -204 0
-249 -250 0
-249 -251 0
-249 -252 0
-250 -251 0
-250 -252 0
-251 -252 0
-197 -198 0
-197 -199 0
-197 -200 0
-198 -199 0
-198 -200 0
-199 -200 0
-253 -254 0
-253 -255 0
-253 -256 0
-254 -255 0
-254 -256 0
-255 -256 0
-193 -194 0
-193 -195 0
-193 -196 0
-194 -195 0
-194 -196 0
-195 -196 0
-257 -258 0
-257 -259 0
-257 -260 0
-258 -259 0
-258 -260 0
-259 -260 0
-25 -26 0
-25 -27 0
-25 -28 0
-26 -27 0
-26 -28 0
-27 -28 0
-261 -262 0
-261 -263 0
-261 -264 0
-262 -263 0
-262 -264 0
-263 -264 0
-21 -22 0
-21 -23 0
-21 -24 0
-22 -23 0
-22 -24 0
```

```
-23 -24 0
-265 -266 0
-265 -267 0
-265 -268 0
-266 -267 0
-266 -268 0
-267 -268 0
-17 -18 0
-17 -19 0
-17 -20 0
-18 -19 0
-18 -20 0
-19 -20 0
-269 -270 0
-269 -271 0
-269 -272 0
-270 -271 0
-270 -272 0
-271 -272 0
-13 -14 0
-13 -15 0
-13 -16 0
-14 -15 0
-14 -16 0
-15 -16 0
-273 -274 0
-273 -275 0
-273 -276 0
-274 -275 0
-274 -276 0
-275 -276 0
-9 -10 0
-9 -11 0
-9 -12 0
-10 -11 0
-10 -12 0
-11 -12 0
-277 -278 0
-277 -279 0
-277 -280 0
-278 -279 0
-278 -280 0
-279 -280 0
-5 -6 0
-5 -7 0
-5 -8 0
-6 -7 0
-6 -8 0
-7 -8 0
-281 -282 0
-281 -283 0
-281 -284 0
-282 -283 0
-282 -284 0
-283 -284 0
-1 -2 0
-1 -3 0
-1 -4 0
-2 -3 0
-2 -4 0
-3 -4 0
-285 -286 0
-285 -287 0
-285 -288 0
-286 -287 0
-286 -288 0
-287 -288 0
-49 -85 0
-50 -86 0
-51 -87 0
-52 -88 0
-45 -89 0
-46 -90 0
-47 -91 0
-48 -92 0
-41 -93 0
-42 -94 0
-43 -95 0
-44 -96 0
-37 -97 0
-38 -98 0
-39 -99 0
-40 -100 0
-121 -153 0
-122 -154 0
-123 -155 0
-124 -156 0
-117 -157 0
-118 -158 0
-119 -159 0
-120 -160 0
-113 -161 0
-114 -162 0
-115 -163 0
-116 -164 0
-109 -165 0
-110 -166 0
-111 -167 0
-112 -168 0
-105 -169 0
-106 -170 0
-107 -171 0
-108 -172 0
-209 -241 0
-210 -242 0
-211 -243 0
-212 -244 0
-205 -245 0
-206 -246 0
-207 -247 0
-208 -248 0
-201 -249 0
-202 -250 0
-203 -251 0
-204 -252 0
-197 -253 0
-198 -254 0
-199 -255 0
-200 -256 0
-193 -257 0
-194 -258 0
-195 -259 0
-196 -260 0
-25 -261 0
-26 -262 0
-27 -263 0
-28 -264 0
-21 -265 0
-22 -266 0
-23 -267 0
-24 -268 0
-17 -269 0
-18 -270 0
-19 -271 0
-20 -272 0
-13 -273 0
-14 -274 0
-15 -275 0
-16 -276 0
-9 -277 0
-10 -278 0
-11 -279 0
-12 -280 0
-5 -281 0
-6 -282 0
-7 -283 0
-8 -284 0
-1 -285 0
-2 -286 0
-3 -287 0
-4 -288 0
-49 88 0
-51 86 0
-50 87 88 0
-52 85 86 0
-45 92 0
-47 90 0
-46 91 92 0
-48 89 90 0
-41 96 0
-43 94 0
-42 95 96 0
-44 93 94 0
-37 100 0
```

```
-39 98 0            -112 165 166 0      -199 254 0          -20 269 270 0
-38 99 100 0        -105 172 0          -198 255 256 0      -13 276 0
-40 97 98 0         -107 170 0          -200 253 254 0      -15 274 0
-121 156 0          -106 171 172 0      -193 260 0          -14 275 276 0
-123 154 0          -108 169 170 0      -195 258 0          -16 273 274 0
-122 155 156 0      -209 244 0          -194 259 260 0      -9 280 0
-124 153 154 0      -211 242 0          -196 257 258 0      -11 278 0
-117 160 0          -210 243 244 0      -25 264 0           -10 279 280 0
-119 158 0          -212 241 242 0      -27 262 0           -12 277 278 0
-118 159 160 0      -205 248 0          -26 263 264 0       -5 284 0
-120 157 158 0      -207 246 0          -28 261 262 0       -7 282 0
-113 164 0          -206 247 248 0      -21 268 0           -6 283 284 0
-115 162 0          -208 245 246 0      -23 266 0           -8 281 282 0
-114 163 164 0      -201 252 0          -22 267 268 0       -1 288 0
-116 161 162 0      -203 250 0          -24 265 266 0       -3 286 0
-109 168 0          -202 251 252 0      -17 272 0           -2 287 288 0
-111 166 0          -204 249 250 0      -19 270 0           -4 285 286 0
-110 167 168 0      -197 256 0          -18 271 272 0
```

B.2 Example IP Encoding for only one Base Stacking

It follows the transformation of `(())` into the CPLEX format. It consists of the following parts:

1. Function to be minimize.
2. A set of constraints.
3. Definition of variables which are used as indicators and thus have to restricted to be either 0 or 1.

```
Minimize F:
        - 24 A1 - 33 A2 - 21 A3 - 14 A4 - 21 A5 -21 A 6
        - 33 A7 - 34 A8 - 25 A9 - 15 A10 - 22 A11 - 24 A12
        - 21 A13 - 25 A14 + 13 A15 - 5 A16 - 14 A17 - 13 A18
        - 14 A19 - 15 A20 - 5 A21 + 3 A22 - 6 A23 - 10 A24
        - 21 A25 - 22 A26 - 14 A27 - 6 A28 - 11 A29 - 9 A30
        - 21 A31 - 24 A32 - 33 A33 - 10 A34 - 9 A35 - 13 A36

Subject To
        constraint1:   + 1 x13 + 1 x14 + 1 x15 + 1 x16 = 1
        constraint2:   + 1 x29 + 1 x30 + 1 x31 + 1 x32 = 1
```

```
constraint3:   + 1 x9 + 1 x10 + 1 x11 + 1 x12 = 1
constraint4:   + 1 x33 + 1 x34 + 1 x35 + 1 x36 = 1
constraint5:   + 1 x13 + 1 x29 <= 1
constraint6:   + 1 x9 + 1 x33 <= 1
constraint7:   + 1 x14 + 1 x30 <= 1
constraint8:   + 1 x10 + 1 x34 <= 1
constraint9:   + 1 x15 + 1 x31 <= 1
constraint10:  + 1 x11 + 1 x35 <= 1
constraint11:  + 1 x16 + 1 x32 <= 1
constraint12:  + 1 x12 + 1 x36 <= 1
constraint13:  - 1 x13 + 1 x32 >= 0
constraint14:  - 1 x14 + 1 x31 + 1 x32 >= 0
constraint15:  - 1 x15 + 1 x30 >= 0
constraint16:  - 1 x16 + 1 x29 + 1 x30 >= 0
constraint17:  - 1 x9 + 1 x36 >= 0
constraint18:  - 1 x10 + 1 x35 + 1 x36 >= 0
constraint19:  - 1 x11 + 1 x34 >= 0
constraint20:  - 1 x12 + 1 x33 + 1 x34 >= 0
constraint21:  - 1 x11 - 1 x34 - 1 x31 - 1 x14 + 1 A1 > -4
constraint22:  - 4 A1 + 1 x11 + 1 x34 + 1 x31 + 1 x14 >= 0
constraint22:  - 1 x10 - 1 x35 - 1 x31 - 1 x14 + 1 A2 > -4
constraint23:  - 4 A2 + 1 x10 + 1 x35 + 1 x31 + 1 x14  >= 0
constraint23:  - 1 x10 - 1 x36 - 1 x31 - 1 x14 + 1 A3 > -4
constraint24:  - 4 A3 + 1 x11 + 1 x34 + 1 x31 + 1 x14  >= 0
constraint25:  - 1 x12 - 1 x34 - 1 x31 - 1 x14 + 1 A4 > -4
constraint26:  - 4 A4 + 1 x12 + 1 x34 + 1 x31 + 1 x14  >= 0
constraint27:  - 1 x9 - 1 x36 - 1 x31 - 1 x14 + 1 A5 > -4
constraint28:  - 4 A5 + 1 x9 + 1 x36 + 1 x31 + 1 x14  >= 0
constraint29:  - 1 x12 - 1 x33 - 1 x31 - 1 x14 + 1 A6 > -4
constraint30:  - 4 A6 + 1 x12 + 1 x33 + 1 x31 + 1 x14  >= 0
constraint31:  - 1 x11 - 1 x34 - 1 x30 - 1 x15 + 1 A7 > -4
constraint32:  - 4 A7 + 1 x11 + 1 x34 + 1 x30 + 1 x15  >= 0
constraint33:  - 1 x10 - 1 x35 - 1 x30 - 1 x15 + 1 A8 > -4
constraint34:  - 4 A8 + 1 x10 + 1 x35 + 1 x30 + 1 x15  >= 0
constraint35:  - 1 x10 - 1 x36 - 1 x30 - 1 x15 + 1 A9 > -4
constraint36:  - 4 A9 + 1 x10 + 1 x36 + 1 x30 + 1 x15  >= 0
constraint37:  - 1 x12 - 1 x34 - 1 x30 - 1 x15 + 1 A10 > -4
constraint38:  - 4 A10 + 1 x12 + 1 x34 + 1 x30 + 1 x15  >= 0
constraint39:  - 1 x9 - 1 x36 - 1 x30 - 1 x15 + 1 A11 > -4
constraint40:  - 4 A11 + 1 x9 + 1 x36 + 1 x30 + 1 x15  >= 0
constraint41:  - 1 x12 - 1 x33 - 1 x30 - 1 x15 + 1 A12 > -4
```

```
constraint42:  - 4 A12 + 1 x12 + 1 x33 + 1 x30 + 1 x15  >= 0
constraint43:  - 1 x11 - 1 x34 - 1 x30 - 1 x16 + 1 A13 > -4
constraint44:  - 4 A13 + 1 x11 + 1 x34 + 1 x30 + 1 x16  >= 0
constraint45:  - 1 x10 - 1 x35 - 1 x30 - 1 x16 + 1 A14 > -4
constraint46:  - 4 A14 + 1 x10 + 1 x35 + 1 x30 + 1 x16  >= 0
constraint47:  - 1 x10 - 1 x36 - 1 x30 - 1 x16 + 1 A15 > -4
constraint48:  - 4 A15 + 1 x10 + 1 x36 + 1 x30 + 1 x16  >= 0
constraint49:  - 1 x12 - 1 x34 - 1 x30 - 1 x16 + 1 A16 > -4
constraint50:  - 4 A16 + 1 x12 + 1 x34 + 1 x30 + 1 x16  >= 0
constraint51:  - 1 x9 - 1 x36 - 1 x30 - 1 x16 + 1 A17 > -4
constraint52:  - 4 A17 + 1 x9 + 1 x36 + 1 x30 + 1 x16  >= 0
constraint53:  - 1 x12 - 1 x33 - 1 x30 - 1 x16 + 1 A18 > -4
constraint54:  - 4 A18 + 1 x12 + 1 x33 + 1 x30 + 1 x16  >= 0
constraint55:  - 1 x11 - 1 x34 - 1 x32 - 1 x14 + 1 A19 > -4
constraint56:  - 4 A19 + 1 x11 + 1 x34 + 1 x32 + 1 x14  >= 0
constraint57:  - 1 x10 - 1 x35 - 1 x32 - 1 x14 + 1 A20 > -4
constraint58:  - 4 A20 + 1 x10 + 1 x35 + 1 x32 + 1 x14  >= 0
constraint59:  - 1 x10 - 1 x36 - 1 x32 - 1 x14 + 1 A21 > -4
constraint60:  - 4 A21 + 1 x10 + 1 x36 + 1 x32 + 1 x14  >= 0
constraint61:  - 1 x12 - 1 x34 - 1 x32 - 1 x14 + 1 A22 > -4
constraint62:  - 4 A22 + 1 x12 + 1 x34 + 1 x32 + 1 x14  >= 0
constraint63:  - 1 x9 - 1 x36 - 1 x32 - 1 x14 + 1 A23 > -4
constraint64:  - 4 A23 + 1 x9 + 1 x36 + 1 x32 + 1 x14  >= 0
constraint65:  - 1 x12 - 1 x33 - 1 x32 - 1 x14 + 1 A24 > -4
constraint66:  - 4 A24 + 1 x12 + 1 x33 + 1 x32 + 1 x14  >= 0
constraint67:  - 1 x11 - 1 x34 - 1 x29 - 1 x16 + 1 A25 > -4
constraint68:  - 4 A25 + 1 x11 + 1 x34 + 1 x29 + 1 x16  >= 0
constraint69:  - 1 x10 - 1 x35 - 1 x29 - 1 x16 + 1 A26 > -4
constraint70:  - 4 A26 + 1 x10 + 1 x35 + 1 x29 + 1 x16  >= 0
constraint71:  - 1 x10 - 1 x36 - 1 x29 - 1 x16 + 1 A27 > -4
constraint72:  - 4 A27 + 1 x10 + 1 x36 + 1 x29 + 1 x16  >= 0
constraint73:  - 1 x12 - 1 x34 - 1 x29 - 1 x16 + 1 A28 > -4
constraint74:  - 4 A28 + 1 x12 + 1 x34 + 1 x29 + 1 x16  >= 0
constraint75:  - 1 x9 - 1 x36 - 1 x29 - 1 x16 + 1 A29 > -4
constraint76:  - 4 A29 + 1 x9 + 1 x36 + 1 x29 + 1 x16  >= 0
constraint77:  - 1 x12 - 1 x33 - 1 x29 - 1 x16 + 1 A30 > -4
constraint78:  - 4 A30 + 1 x12 + 1 x33 + 1 x29 + 1 x16  >= 0
constraint79:  - 1 x11 - 1 x34 - 1 x32 - 1 x13 + 1 A31 > -4
constraint80:  - 4 A31 + 1 x11 + 1 x34 + 1 x32 + 1 x13  >= 0
constraint81:  - 1 x10 - 1 x35 - 1 x32 - 1 x13 + 1 A32 > -4
constraint82:  - 4 A32 + 1 x10 + 1 x35 + 1 x32 + 1 x13  >= 0
```

```
        constraint83:  - 1 x10 - 1 x36 - 1 x32 - 1 x13 + 1 A33 > -4
        constraint84:  - 4 A33 + 1 x10 + 1 x36 + 1 x32 + 1 x13  >= 0
        constraint85:  - 1 x12 - 1 x34 - 1 x32 - 1 x13 + 1 A34 > -4
        constraint86:  - 4 A34 + 1 x12 + 1 x34 + 1 x32 + 1 x13  >= 0
        constraint87:  - 1 x9 - 1 x36 - 1 x32 - 1 x13 + 1 A35 > -4
        constraint88:  - 4 A35 + 1 x9 + 1 x36 + 1 x32 + 1 x13  >= 0
        constraint89:  - 1 x12 - 1 x33 - 1 x32 - 1 x13 + 1 A36 > -4
        constraint90:  - 4 A36 + 1 x12 + 1 x33 + 1 x32 + 1 x13  >= 0
        constraint91:  + 1 A1 + 1 A2 + 1 A3 + 1 A4 + 1 A5 + 1 A6 + 1 A7
                        + 1 A8 + 1 A9 + 1 A10 + 1 A11 + 1 A12 + 1 A13
                        + 1 A14 + 1 A15 + 1 A16 + 1 A17 + 1 A18 + 1 A19
                        + 1 A20 + 1 A21 + 1 A22 + 1 A23 + 1 A24 + 1 A25
                        + 1 A26 + 1 A27 + 1 A28 + 1 A29 + 1 A30 + 1 A31
                        + 1 A32 + 1 A33 + 1 A34 + 1 A35 + 1 A36 = 1

Binary                     x32                A12                A26
        x13                x12                A13                A27
        x29                x36                A14                A28
        x9                 A1                 A15                A29
        x33                A2                 A16                A30
        x14                A3                 A17                A31
        x30                A4                 A18                A32
        x10                A5                 A19                A33
        x34                A6                 A20                A34
        x15                A7                 A21                A35
        x31                A8                 A22                A36
        x11                A9                 A23
        x35                A10                A24
        x16                A11                A25
```

Appendix C

Ramsey Numbers

Our recursive procedure to find upper and lower bounds for Ramsey numbers $R(m, n)$ is not only capable to give bounds for small values for m and n but also provides bounds for values $m, n \geq 15$. In Table C.1 bounds for $3 \leq m \leq 10$ and $16 \leq n \leq 20$ are presented.

m,n	16	17	18	19	20
3	45 136	48 153	51 171	54 190	57 210
4	74 814	79 967	84 1138	89 1328	94 1538
5	103 3846	110 4813	117 5951	124 7279	131 8817
6	132 15260	141 20073	150 26024	159 33303	168 42120
7	161 52850	172 72923	183 98947	194 132250	205 174370
8	190 164022	203 236945	216 335892	229 468142	242 642512
9	219 464902	234 701847	249 1037739	264 1505881	279 2148393
10	248 1220694	265 1922541	282 2960280	299 4466161	316 6614554

Table C.1: Bounds for $R(m, n)$ with $3 \leq m \leq 10$ and $16 \leq n \leq 20$ calculated by Algorithm 4 on page 51.

The following compilation of bounds for $R(m, n)$ seen in Table C.2 on the next page shows results for even larger values for m and n.

m,n	20	21	22	23	24	25
15	501 606415993	528 1040259091	555 1748018179	582 2881883262	609 4668162874	636 7438754716
16	538 1355953974	567 2396213065	596 4144231244	625 7026114506	654 11694277380	683 19133032096
17	575 2925749175	606 5321962240	637 9466193484	668 16492307990	699 28186585370	730 47319617466
18	612 6111061458	645 11433023698	678 20899217182	711 37391525172	744 65578110542	777 112897728008
19	649 12390242264	684 23823265962	719 44722483144	754 82114008316	789 147692118858	824 260589846866
20	686 24444245834	723 48267511796	760 92989994940	797 175104003256	834 322796122114	871 583385968980
21	723 48267511796	762 95293063850	801 188283058790	840 363387062046	879 686183184160	918 1269569153140
22	760 92989994940	801 188283058790	842 371959979762	883 735347041808	924 1421530225968	965 2691099379108
23	797 175104003256	840 363387062046	883 735347041808	926 1453548248186	969 2875078474154	1012 5566177853262
24	834 322796122114	879 686183184160	924 1421530225968	969 2875078474154	1014 5686120903874	1059 11252298757136
25	871 583385968980	918 1269569153140	965 2691099379108	1012 5566177853262	1059 11252298757136	1106 22264711413050

Table C.2: Bounds for larger $R(m,n)$ calculated by Algorithm 4 on page 51 by adding knowledge about already known values.

List of Figures

1.1 Reduction from B to A: $B \leq_p A$ 3

1.2 Schema how to use SAT-solvers to solve arbitrary **NP** problems 4

1.3 Interpretation of a Boolean formula as a Boolean circuit 6

3.1 Example: Binary search to find $\chi(G)$ 18

3.2 Run-time of different SAT-solver to find $\chi(\textit{queen8\_8})$. 18

4.1 Example for a pseudo-knot configuration 23

4.2 Different representations of secondary structure 24

4.3 Secondary structure representations of yeast tRNA$^{\text{Phe}}$ 28

4.4 Yeast tRNA$^{\text{Phe}}$'s inverse folding result 29

4.5 Dependency graph Ψ for multiple secondary structures 30

4.6 Free energies for stacked pairs 31

4.7 Example secondary structure with contributing energies 32

4.8 Each base stacking can hold one of 36 energy values. 36

4.9 Example showing the usage of $\pi(s_l, p)$ 39

4.10 Yeast tRNA$^{\text{Phe}}$'s inverse folding result using CPLEX 40

5.1 Plot of $\binom{n}{k}$ for $1 \leq n \leq 100$ and $k = 3, 4, \ldots, 10$ 45

5.2 $R(3,3) \neq 4$ and $R(3,3) \neq 5$ 49

5.3 Small graphs without a CLIQUE and an INDEPENDENTSET 55

5.4 Larger graphs without a CLIQUE and an INDEPENDENTSET 55

5.5 Ramsey instance files size for $k = 43$ and $3 \leq m, n \leq 10$. 57

5.6 Run-times to find small Ramsey numbers $R(m,n)$ 58

A.1 *1-Insertions_4* is 5-colorable 61

A.2 *2-Insertions_3* is 4-colorable 61

A.3 *3-Insertions_3* is 4-colorable 62

A.4 *4-Insertions_3* is 4-colorable 62

A.5 *DSJC125.1* is 5-colorable 62

A.6 *1-FullIns_3* is 4-colorable, *1-FullIns_4* is 5-colorable and *1-FullIns_5* is 6-colorable 63

A.7 *2-FullIns_3* is 5-colorable and *2-FullIns_4* is 6-colorable 64

A.8 *3-FullIns_3* is 6-colorable, *3-FullIns_4* is 7-colorable and *3-FullIns_5* is 8-colorable 65

A.9 *4-FullIns_3* is 7-colorable and *4-FullIns_4* is 8-colorable 66
A.10 *5-FullIns_3* is 8-colorable and *5-FullIns_4* 67
A.11 *le450_5a* is 5-colorable, *le450_5c* is 5-colorable and *le450_5d* is 5-colorable 68
A.12 *anna* is 11-colorable and *david* is 11-colorable 69
A.13 *huck* is 11-colorable and *jean* is 10-colorable 70
A.14 *queen5_5* is 5-colorable, *queen6_6* is 7-colorable and *queen7_7* is 7-colorable 71
A.15 *myciel3* is 4-colorable, *myciel4* is 5-colorable and *myciel5* is 6-colorable 72
A.16 *ash331GPIA* is 4-colorable, *ash608GPIA* is 4-colorable and *will199GPIA* is 7-colorable 73
A.17 *mugg88_1* is 4-colorable and *mugg88_25* is 4-colorable 74
A.18 *mugg100_1* is 4-colorable and *mugg100_25* is 4-colorable 75
A.19 *abb313GPIA* is 9-colorable 75

List of Tables

2.1 Used hardware configuration 13

3.1 Mapping from colored vertices to integer values. 16
3.2 Run-times to find χ for *queen8_8* 19
3.3 Chromatic numbers for hard graph instances 19
3.4 Bounds for chromatic numbers for hard graph instances 20
3.5 New, not published results for chromatic numbers 21

4.1 Mapping from coded based positions to integer values. 27

5.1 Values and bounds for Ramsey numbers $R(m, n)$ 52
5.2 Bounds for $R(m, n)$ calculated by Algorithm 4 53
5.3 Improved bounds for $R(m, n)$ using already known values 54

C.1 Bounds for $R(m, n)$ with $3 \leq m \leq 10$ and $16 \leq n \leq 20$ 83
C.2 Bounds for larger $R(m, n)$ 84

List of Algorithms

1 MinColorSearch 17

2 SecondaryStructureToCircularRepresentation 29

3 CoRamsey 44

4 RamseyBounds 51

Bibliography

[1] I. G. Abfalter, C. Flamm, and P. F. Stadler, *Design of multi-stable nucleid acid sequences.*, in German Conference on Bioinformatics, 2003, pp. 1–7.

[2] A. Biere, *Limmat, Computer Systems Institute ETH Zürich.*

[3] ——, *Limboole, Computer Systems Institute ETH Zürich*, 2003.

[4] ——, *The Evolution from Limmat to Nanosat*, Tech. Rep. 444, Dept. of Computer Science, ETH Zürich, 2004.

[5] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, *Symbolic Model Checking without BDDs.*, in TACAS, 1999, pp. 193–207.

[6] A. Biere and C. Sinz, *Decomposing SAT Problems into Connected Components*, Journal on Satisfiability, Boolean Modeling and Computation, (2006). Accepted for publication.

[7] S. A. Burr, P. Erdös, R. J. Faudree, and R. H. Schelp, *On the difference between consecutive Ramsey numbers.*, Utilitas Mathematica, 35 (1989), pp. 115–118.

[8] A. Cobham, *The intrinsic computational difficulty of functions*, in Proc. 1964 International Congress for Logic, Methodology, and Philosophy of Science, Y. Bar-Hillel, ed., North-Holland, Amsterdam, 1964, pp. 24–30.

[9] S. A. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the Third IEEE Symposium on the Foundations of Computer Science, 1971, pp. 151–158.

[10] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, (1963).

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

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

[13] W. DeLano, *The PyMOL Molecular Graphics System (2002) DeLano Scientific, San Carlos, CA, USA.*, 2002.

[14] J. Edmonds, *Paths, trees, and flowers*, Can. J. Math., 17 (1965), pp. 449–467.

[15] N. Eén and N. Sörensson, *An Extensible SAT-solver.*, in SAT, E. Giunchiglia and A. Tacchella, eds., vol. 2919 of Lecture Notes in Computer Science, Springer, 2003, pp. 502–518.

[16] ——, *An Extensible SAT-solver*, Notes in Computer Science, 2919 (2004), pp. 502–518.

[17] C. Flamm, I. L. Hofacker, S. Maurer-Stroh, P. F. Stadler, and M. Zehl, *Design of multistable RNA molecules.*, RNA, 7 (2001), pp. 254–265.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability (A Guide to Theory of NP-Completeness)*, Freeman, San Francisco, 1979.

[19] K. Gödel, *letters to von Neuman*, 1956.

[20] R. E. Greenwood and A. M. Gleason, *Combinatorial relations and chromatic graphs*, Canadian Journal of Mathematics, 7 (1955), pp. 1–7.

[21] I. L. Hofacker and P. F. Stadler, *RNA Secondary Structure.*

[22] N. Karmarkar, *A New Polynomial-Time Algorithm for Linear Programming*, in STOC, 1984, pp. 302–311.

[23] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, 1972, pp. 85–103.

[24] L. G. Khachiyan, *A Polynomial Algorithm in Linear Programming*, Soviet Mathematics Doklady, 20 (1979), pp. 191–194.

[25] L. A. Levin, *Universal search problems*, Problemi Peredachi Informatsii, 9 (1973), pp. 265–266.

[26] Y. S. Mahajan, Z. Fu, and S. Malik, *zchaff2004: An Efficient SAT Solver.*, in SAT (Selected Papers, 2004, pp. 360–375.

[27] J. P. Marques-Silva and K. A. Sakallah, *GRASP - A New Search Algorithm for Satisfiability*, in Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 1996, pp. 220–227.

[28] J. P. Marques-Silva and K. A. Sakallah, *GRASP: A Search Algorithm for Propositional Satisfiability*, in IEEE Transactions on Computers, vol. 48, 1999, pp. 506–521.

[29] J. S. McCaskill, *The equilibrium partition function and base pair binding probabilities for RNA secondary structure.*, Biopolymers, 29 (1990), pp. 1105–1119.

[30] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, *Chaff: Engineering an Efficient SAT Solver*, in Proceedings of the 38th Design Automation Conference (DAC'01), June 2001.

[31] T. H. Nguyen, *Graph Coloring Benchmark Instances.* On-line at http://cs.hbg.psu.edu/txn131/INSTANCES/graphcoloring.html.

[32] J. Oberhofer, *The architecture of SAT solvers and their applicability to NP-complete problems*, Master's thesis, Technische Universität Wien, 2003.

[33] C. H. Papadimitriou, *On the complexity of integer programming.*, J. ACM, 28 (1981), pp. 765–768.

[34] C. M. Papadimitriou, *Computational complexity*, Addison-Wesley Publishing Company, Inc., 1994.

[35] S. P. Radziszowski, *Small Ramsey Numbers, Dynamic Survey DS1 Revision 11, August 1, 2006*, Electronic Journal of Combinatorics, (1994).

[36] F. P. Ramsey, *On a problem in formal logic*, Proceedings of the London Mathematical Society (3), 30 (1930), pp. 264–286.

[37] C. Reidys, P. F. Stadler, and P. Schuster, *Genric Properties of Combinatory Maps: Neutral Networks of RNA Secondary Structures.*, Bull Math Biol, 59 (1997), pp. 339–397.

[38] M. Schaefer, *Graph Ramsey Theory and the Polynomial Hierarchy.*, in STOC, 1999, pp. 592–601.

[39] J. Stirling, *Methodus Differentialis: sive Tractatus de Summatione et Interpolatione Serierum Infinitarum*, 1730.

[40] M. Trick, *Network Ressources for Coloring a Graph.* On-line at http://mat.gsia.cmu.edu/COLOR/color.html.

[41] G. Tseitin, *On the complexity of derivation in propositional calculus*, Studies in Constructive Mathematics and Mathematical Logic, (1968), pp. 115–125.

[42] A. M. Turing, *On Computable Numbers, with an application to the Entscheidungsproblem*, Proc. London Math. Soc., 2 (1936), pp. 230–265.

[43] K. Walker, *Dichromatic Graphs and Ramsey Numbers*, Journal of Combinatorial Theory, 5 (1968), pp. 238–243.

[44] M. S. Waterman and T. F. Smith, *RNA secondary structure: A complete mathematical analysis.*, Mathematical Biosciences, 42 (1978), pp. 257–266.

[45] J. D. Watson and F. H. C. Crick, *Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid*, Nature, 171 (1953), pp. 737–738.

[46] G. Weberndorfer, *Computational Models of the Genetic Code Evolution Based on Empirical Potentials*, PhD thesis, Fakultät für Naturwissenschaften und Mathematik der Universität Wien, 2002.

[47] M. Zuker, *On finding all suboptimal foldings of an RNA molecule.*, Science, 244 (1989), pp. 48–52.

[48] M. Zuker and P. Stiegler, *Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information.*, Nucleic Acids Res, 9 (1981), pp. 133–148.

Index

coNP$^{\mathbf{NP}}$, 46
m-CLIQUE, 42
n-INDEPENDENTSET, 42
0-1 INTEGER PROGRAMMING, 34
AU, UA, CG, GC, GU, UG, 23
BOOLEAN SATISFIABILITY, 5
A, G, C, U, 23
CIRCUITSAT, 5
COLORING problem, 25
IP, *see* INTEGER PROGRAMMING
INTEGER PROGRAMMING, 10, 32, 34
LP, *see* LINEAR PROGRAMMING, *see* LINEAR PROGRAMMING
LP format, 10
LINEAR PROGRAMMING, 10, 33
MINCOLORING, 14, 16
NP, 1, 2
P, 1, 2
PBO, 10, *see* PSEUDO BOOLEAN OPTIMIZATION
PH, *see* Polynomial Hierarchy
PSEUDO BOOLEAN OPTIMIZATION, 34
RAMSEY, 42
SAT, *see* BOOLEAN SATISFIABILITY
SAT-solver, 4
SATISFIABILITY problem, 5
`compsat`, 19
k-COLORING, 25
`limmat`, 19, 28
`minisat`, 19
`zChaff`, 19

binary search, 16
BMC, *see* Bounded Model Checking
Bounded Model Checking, 4
bracket notation, 23

chromatic number χ, 14
circular representation, 23
CNF, *see* conjunctive normal form
complete graph, 42
completeness, 3
  NP-completeness, 6, 7
conformation, 22
  meta-stable, 22
  non-native, 22, 29
conjunctive normal form, 5, 9
Cook's theorem, 6
CPLEX, 10

decision problem, 1
dependency graph, 30
DIMACS, 9

energy barrier, 22
energy parameter, 31
ensemble, 31

graph, 24
  complete graph, 42
  independent graph, 42
  subgraph, 41
  undirected, 24
graph COLORING, 24

hardness, 3
hydroxyl group, 23

independent graph, 42
inverse RNA folding problem, 24

Karp's 21 problems, 34

literal, 9
loop, 31
loop-based energy model, 31

minimum free energy, 31
molecular switch, 22, 29
mountain representation (plot), 23

nucleotide, 23

objective function, 10
optimization problem, 32
oracle, 45

phosphate group, 23
polymer, 22
Polynomial Hierarchy, 45, 46
pseudo-knots, 22

quantifier alternation, 45, 46

Reduction
  Karp Reduction, 2
ribonucleic acid, 22
ribose, 23
RNA, *see* ribonucleic acid
RNA folding problem, 23
RNA switch problem, 30

satisfiability, 4
satisfiable expression, 4
satzoo, 29
secondary structure, 22
  circular representation, 23
  mountain representation (plot), 23
simplex algorithm, 34
stacked base pairs, 31
subgraph, 41
subgraph isomorphism, 43

tRNA$^{\mathrm{Phe}}$, 27
Turing machine
  deterministic, 2
  non-deterministic, 2

Watson-Crick, 23
wobble pairs, 23