Exploiting Structure in Integer Programs

Jakub Mareček

Thesis submitted to the University of Nottingham
for the degree of Doctor of Philosophy

June 2011

The Abstract

The thesis argues the case for exploiting certain structures in integer linear programs.

Integer linear programs are optimisation problems, where one minimises or maximises a linear function of variables, whose values are required to be integral as well as satisfying certain linear equalities and inequalities. For such an abstract problem, there are very good general-purpose solvers. The state of the art in such solvers is an approach known as "branch and bound". The performance of such solvers depends crucially on four types of in-built heuristics: primal, improvement, branching, and cut-separation or, more generally, bounding heuristics. However, such heuristics have, until recently, not exploited structure in integer linear programs beyond the recognition of certain types of single-row constraints.

Many alternative approaches to integer linear programming can be cast in the following, novel framework. "Structure" in any integer linear program is a class of equivalence among triples of algorithms: deriving combinatorial objects from the input, adapting them, and transforming the adapted object to solutions of the original integer linear program. Many such alternative approaches are, however, inherently incompatible with branch and bound solvers. We, hence, define a structure to be "useful", only when it extracts submatrices, which allow for the implementation of more than one of the four types of heuristics required in the branch and bound approach. Although the extraction of the best possible submatrices is non-trivial, the lack of a considerable submatrix with a given property can often be recognised quickly, and storing useful submatrices in a "pool" makes it possible to use them repeatedly. The goal is to explore whether the state-of-the-art solvers could make use of the structures studied in the academia.

Three examples of useful structures in integer linear programs are presented. A particularly widely applicable useful structure relies on the aggregation of variables. Its application can be seen as a decomposition into three stages: Firstly, we partition variables in the original instance into as small number as possible of support sets of constraints forcing convex combinations of binary variables to be less than or equal to one in the original instance, and one-element sets. Secondly, we solve the "aggregated" instance corresponding to the partition of variables. Under certain conditions, we obtain a valid lower bound. Finally, we fix the solution of the aggregated instance in primal and improvement heuristics for the original instance, and use the partition in hyper-plane branching heuristics. Under certain conditions, the

primal heuristics are guaranteed to find a feasible solution to the original instance.

We also present structures exploiting mutual-exclusion and precedence constraints, prevalent in scheduling and timetabling applications. Mutual exclusion constraints correspond to instances of graph colouring. For numerous extensions of graph colouring, there are natural primal and branching heuristics. We present lower bounding heuristics for extensions of graph colouring, based on augmented Lagrangian methods for novel semidefinite programming relaxations, and reformulations based on a novel transformation of graph colouring to graph multicolouring. Precedence constraints correspond to an instance of precedence-constrained multi-dimensional packing. For such packing problems, we present heuristics based on an adaptive discretisation and strong discretised linear programming relaxations. On instances of packing unit-cubes into a box, the reformulation makes it possible to solve instances that are by five orders of magnitude larger than previously. On instances from complex timetabling problems, which combine mutual-exclusion and packing constraints, the combination of heuristics above can often result in the gap between primal and dual bounds being reduced to under five percent, orders of magnitude faster than using state of the art solvers, without any information being used that is outside of the instance.

Acknowledgements

[C]ertain kinds of hard work are in fact [...]
more fun than just about anything else.

Donald E. Knuth

First of all, I would like to thank Professor Edmund K. Burke and Dr. Andrew J. Parkes, who have kindly supervised my work, and Professor Adam N. Letchford, who has kindly agreed to serve as the external examiner of my dissertation. Thanks are also due to the University of Nottingham, the Engineering and Physical Sciences Research Council, ARM Ltd., IBM Corp., Fields Institute, and London Mathematical Society for their financial support.

I would also like to thank many others for their informal, but no less valuable support and encouragement. Petr Hliněný introduced me to Integer Linear Programming in the fall of 2005, and has been a source of inspiration since. The organisers and attendees of Mixed Integer Programming workshops have helped shape my thinking about integer programming. In particular, I would like to acknowledge a discussion with George Nemhauser at my first MIP Workshop in the summer of 2007 and discussions with Tobias Achterberg and Roland Wunderling in subsequent years. Jacek Gondzio, Florian Jarre, and Adam Letchford have tried to help me understand convex programming. Danny Kershaw has tried to help me understand the wonderful world of engineering. Timo Berthold, Tomáš Brázdil, Vašek Brožek, Honza Obdržálek, Peter Richtárik, Hana Rudová, and Standa Živný have taught me – knowingly or not – much about the academe. Sam Allen, Janka Chlubná, Petr Dvořák, Enrico Glaab, Sven Grönemeyer, Rupa Jagannathan, Klára Marečková, David Sim, Anna Siyanova, Filip Sokol, Pavel Šabatka, Paweł Widera, and Radka Zounková have been a great company, all throughout. Thank you.

Last but not least, I would like to thank my parents.

The List of Papers

This dissertation is loosely based on the following papers:

1. Decomposition, Reformulation, and Diving in University Course Timetabling. *Computers and Operations Research* (2010) 37(3), 582–597.

2. A Supernodal Formulation of Vertex Colouring with Applications in Course Timetabling. *Annals of Operations Research* (2010) 179(1), 105–130.

3. Semidefinite Programming Relaxations in Timetabling. *Int. Conference on the Practice and Theory of Automated Timetabling, 2010.*

4. Programming Language for Optimisation with Semi-Automated Reformulations. In preparation.

5. A Space-indexed Formulation of Packing Boxes into a Larger Box. *Operations Research Letters* (2011), to appear.

6. Some Probabilistic Results on Width Measures of Graphs. *Graph Classes, Optimization, and Width Parameters, 2009.*

7. A Primal Heuristic for Integer Linear Programming with Automated Aggregations. *Int. Symposium on Mathematical Programming, 2009.*

The first five papers are co-authored with my supervisors. The fifth paper is a very pleasant joint work with Sam D. Allen. The author is indebted to Noga Alon, whose elegant proofs [23, 20] have inspired those in the sixth paper, and Chris Wuthrich, Bjarki Holm, and Sang-il Oum, who have kindly provided invaluable comments on early drafts of the paper.

During my time at Nottingham and Cambridge, I had the pleasure to work on a number of projects unrelated to my dissertation. This has resulted in the following papers and drafts:

1. Penalising Patterns in Timetables: Novel Integer Programming Formulations, in *Operations Research Proceedings 2007*, Stefan Nickel and Jörg Kalcsics, eds., Springer, 2008, pp. 409–414.

2. A Branch-and-Cut Procedure for the Udine Course Timetabling Problem. *Annals of Operations Research*, to appear.

3. A Cutting Surface Method for Integer Least Squares with Application in MIMO Signal Decoding. Submitted.

4. Dynamic Data Structures for Taskgraph Scheduling Policies with Applications in OpenCL Accelerators. *Multidisciplinary Int. Scheduling Theory and Applications Conference 2011*, to appear.

5. The Complexity of Scheduling in OpenCL Accelerators. In preparation.

6. Semidefinite Programming Relaxations of Integer Least Squares with Applications in MIMO Signal Decoding. Submitted.

7. Discretisation in Stochastic Control of Bandits with Uncountably Many Arms with Applications in Tuning Parameter of Scanning Probe Microscopes. In preparation.

The third and fourth papers are based on an internship with ARM Ltd. The sixth paper grew out of an idea of Florian Jarre and subsequent joint work. The last-listed paper is based on an fellowship with the Nottingham Nanotechnology and Nanoscience Centre.

Contents

I Preliminaries **1**

1 Introduction **3**

2 The State of the Art in Integer Linear Programming **13**

2.1 Integer Linear Programming 13

2.2 Primal Heuristics 17

2.3 Improvement Heuristics 22

2.4 Branching Heuristics 26

2.5 Bounding and Cut Generation Heuristics 30

II Known Structures in Integer Programming **37**

3 Tree Decompositions **47**

3.1 The Many Definitions of Tree-width 47

3.2 Branch-width and Related Parameters 48

3.3 Finding the Decomposition 49

3.4 Exploiting the Decomposition 51

3.5 An Analysis 53

4 Unimodularity and Generalisations **57**

4.1 Good Matrices 57

4.2 Finding Good Submatrices 60

4.3 Exploiting Good Submatrices 63

4.4 An Analysis 63

5 The Bordered Block-Diagonal Form 65

5.1 Dantzig-Wolfe Decomposition and Multi-Commodity Flows . . 65

5.2 Reordering in a Bordered Block-Diagonal Form 66

5.3 Exploiting the Bordered Block-Diagonal Form 69

5.4 An Analysis 69

III Useful Structures in Integer Programming 73

6 Variable Aggregations 79

6.1 Aggregating Variables in Integer Programs: An Introduction . 79

6.2 Finding Aggregations 81

6.3 Exploiting Aggregations 85

6.4 An Analysis 87

6.5 Extensions 89

6.6 Computational Experience 90

7 Colouring Components 93

7.1 Graph Colouring and Integer Programming: An Overview . . 93

7.2 Finding the Colouring Component 95

7.3 Exploiting the Colouring Component 100

8 Precedence-Constrained Components 119

8.1 Scheduling, Packing, and Integer Programming: An Overview 119

8.2 Finding the Precedence-Constrained Component 123

8.3 Exploiting the Precedence-Constrained Component 125

9 Conclusions 131

9.1 Open Questions 131

A The Preservation of Structure **135**

B The Exploitation of Unknown Structures **137**

C The Computational Results **141**

Bibliography **155**

Part I

Preliminaries

Chapter 1

Introduction

[Optimisation is a] fascinating blend of theory and computation, heuristics and rigor.

Roger Fletcher [163]

Integer Programming

There are a number of problems where one needs to find the best out of certain distinct configurations, and the number of configurations is finite, but grows exponentially with some natural measure of the input. Many problems in air traffic management, production planning, supply chain management, vehicle routing, or staff rostering can be cast in this way. For an important class of such problems (NPO-Complete, [26]), all known methods require amounts of time that are exponential in some measure of the input in the worst case. In practice, such problems are often formulated as instances of integer linear programming, which can be formulated as follows.

Problem 1.1. Integer Linear Programming (ILP, [324, 415]): Given positive integers n, m, an n-vector c, an m-vector b, and an $m \times n$ matrix $A = (a_{i,j})$, find the assignment of integral values to n-vector variable x such that $Ax \leq b$ and $c^T x$ is minimised:

$$\begin{array}{lll} \text{minimise } c_1x_1 + c_2x_2 + \cdots + c_nx_n & & \text{(Objective)} \\ \text{s. t. } a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n & \leq b_1 & \\ \cdots & & \\ a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n & \leq b_m & \text{(Constraints)} \\ x_1, \ldots, x_n & \in \mathbb{Z}_0^+ & \text{(Integrality)} \end{array}$$

Any integral n-vector x' satisfying $Ax' \leq b$ is called a "feasible solution" or just solution, whereas x' minimising $c^T x'$ among x satisfying $Ax \leq b$ is called

an "optimal solution".

There are a number of general-purpose integer linear programming solvers, which can obtain the optimal solution to any instance of integer linear programming. Surprisingly often, their run-time is not only finite, but also usable in practice.

When the solver is stopped prior to convergence, but after some solver- and instance-dependent time t_0, an assignment x' of feasible integer values together with a lower bound g on $c^T x$ for any x is produced. The quantity $1 - (g/c^T x')$ is denoted "the gap". Unlike the "approximation ratio" of approximation algorithms, which is an *a priori* guarantee, the gap of convergent general-purpose integer linear programming solvers is, until convergence, only a *post-hoc* guarantee.

As a running example through-out the thesis, we use instances of a timetabling problem, where events need to be assigned to periods and rooms, subject to mutual-exclusion constraints on the assignment to periods, and capacity constraints on the assignment to rooms:

Problem 1.2. TIMETABLING PROBLEM WITH OBJECTIVE f: Given an undirected graph $G = (V, E)$, called the "conflict graph" whose vertices V are called "events", integers $n, m \leq |V|$, a $|V|$-vector of capacity-requirements q, an m-vector of capacities c, return an assignment of events to periods and rooms encoded using a binary variable $x_{v,p,r}$, where 1 denotes that event v is assigned to period p and room r and 0 otherwise, such that

- for each event v, $\sum_{p,r} x_{v,p,r} = 1$
- for each period-room pair p, r, $\sum_{v} x_{v,p,r} \leq 1$
- for each event v, $\sum_{p,r} x_{v,p,r} = q_v$
- for each event-room pair v, r, $\sum_{p} q_v x_{v,p,r} \leq c_r$
- for each edge $(u, v) \in E$ and period p, $\sum_{r}(x_{u,p,r} + x_{v,p,r}) \leq 1$
- an objective function f is minimised.

Notice that the statement of the problem is, in effect, an integer linear program. Notice also that the assignment of events to periods corresponds to the well-known problem of graph colouring, whose most common compact ILP formulation uses binary variables $x_{v,c}$, where 1 denotes vertex v is

assigned colour c and 0 otherwise. The assignment of events to rooms corresponds to another well-known problem, bipartite matching. It seems only natural for solvers to use such insight. However, no current general-purpose integer linear programming solvers do so.

Standard Integer Linear Programming solvers traverse a "branch and bound" search tree. At each node of the tree, the solver obtains the optimum of a linear programming relaxation (Objective, Constraints, $x_i \geq 0$) [134, 110] of a minimisation problem, which gives a lower-bound valid for the corresponding sub-tree. A variable x_i with fractional value v_i in the relaxation is picked, and two new nodes are formed, where $x_i \leq \lfloor v_i \rfloor$ in one, and $x_i \geq \lceil v_i \rceil$ in the other. If an upper bound is available, whose value is less than or equal to the lower bound at a node, the node can be disregarded ("pruned"), as it is guaranteed not to contain a solution better than what we have. In theory, one needs to visit a number of nodes that are exponential in the number of variables in order to find the optimal solution.

In practice, the performance of such methods depends on the quality of the lower bounds and upper bounds, and the method for choosing variables to branch on. As will be explained in Chapter 2, the problems of arriving at solutions to instances of integer linear programming, of finding an additional constraint, which is valid for all integer solutions but violated by the current fractional solution, and the problem of choosing the best variable to branch on, are in some sense as hard as the problem of integer linear programming. One, hence, resorts to heuristics, which are often called primal heuristics, cut separation heuristics, and branching heuristics, respectively. See Figure 2.1 (p. 35) for an illustration. Such heuristics have undergone considerable development recently [61, 63, 253, 60, 64], so much so that it is often possible for general-purpose solvers, such as IBM ILOG CPLEX [249], FICO Xpress Optimization Suite [215], Gurobi Optimizer, SCIP [8], and CBC [165], to visit only a number of nodes of the branch and bound tree sublinear in the number of variables (!) before an optimal solution is found. This constitutes the state of the art.

The state-of-the-art solvers (listed above) use a very similar design, and hence share its limitations. They treat an instance of integer linear programming as a collection of constraints, where each constraint is a collection of variable coefficients. In particular, they do not employ:

- alternative forms of input, allowing the user to describe the structure in the instance, except for special ordered sets [39]
- the storage of combinatorial objects allowing for the exploitation of the

structure in the instance

- aggregation of variables, other than the removal of any variables that are "obviously" superfluous
- lower-bounding procedures other than linear programming relaxations on the original variables
- branching on general disjunctions, rather than just the "variable disjunction" $x_i \leq \lfloor v_i \rfloor \vee x_i \geq \lceil v_i \rceil$, except for special ordered sets [39].

In Example 1.2, it is easy to illustrate the gravity of the shortcomings: The state-of-the-art solvers have no means of branching on the decision whether an event v is assigned to period p, i.e. $(\sum_r x_{v,p,r} = 1) \vee (\sum_r x_{v,p,r} = 0)$, but rather branch only on the decision whether event v is assigned to period p and room r, i.e. $(x_{v,p,r} = 1) \vee (x_{v,p,r} = 0)$. We propose to relieve these short-comings.

THE PROPOSED APPROACH

This thesis focuses on solving instances, which are:

- compact and can be passed to the solver in full
- binary, that is all variables are either 0 or 1 in any solution
- structured, where structure is defined only in Chapter II, but one can imagine that a structured instance may have multiple different components in the solution, each with a number of related constraints, linked together by a number of further constraints.

Although most of the methods apply to mixed integer programming in general, when there is no "useful" structure present, there will not be any improvement. In the main body of the thesis, we assume only the constraint matrix and right-hand-side and objective vectors are known. Only in the appendices, we present methods for passing the input to the solver in a structured form.

We present algorithms for both finding and exploiting certain "useful" structures, which can be seen as useful combinatorial objects stored in a "pool". The structures under consideration are aggregations of variables, mutual-exclusion- and precedence-constrained components. We propose to

exploit the structures in primal, improvement, branching, and bounding heuristics, of modern general-purpose solvers.

The approach can be seen as a decomposition into three stages: Firstly, we find the structure. In the case of aggregations of variables, one partitions variables in the original instance into as small a number as possible of one-element sets and support sets of constraints forcing convex combinations of binary variables to be less than or equal to one in the original instance. Secondly, we obtain the bounds. In the case of aggregations of variables, one solves the aggregated instance. For other structures, we propose automated and semi-automated reformulations for obtaining strong relaxations of pseudopolynomial size and certain strong convex programming relaxations of polynomial size. Thirdly, we try to obtain a solution to the original problem, which is, under certain conditions, guaranteed to be available. In the case of aggregations of variables, one restricts the original problem with the solution to the aggregated problem. Furthermore, one can use the structure to branch on. Fractional sums of variables bounded from above by one correspond, indeed, to natural hyper-planes to branch on. We stress that this approach is compatible with the architecture of modern general-purpose solvers.

The aggregations of variables, mutual-exclusion and precedence constraints are prevalent in scheduling, packing, and timetabling applications. Mutual exclusion constraints correspond to an instance of graph colouring. For numerous extensions of graph colouring, there are natural primal and branching heuristics, and the present best lower bounding heuristics are based on matrix-free implementations of augmented Lagrangian methods for semidefinite programming relaxations, which we present. Precedence constraints correspond to an instance of precedence-constrained multi-dimensional packing. For such packing problems, we present natural primal heuristics and lower bounding heuristics based on an adaptive discretisation and strong discretised linear programming relaxations. In other applications, other structures may need to be found.

We focus on a reduction of the time it takes to obtain a solution with non-trivial post-hoc guarantees of its quality. On a set of instances of integer linear programming from certain complex timetabling, packing, and scheduling problems, for example, the combination of heuristics above can often close to a gap of under five percent by orders of magnitude faster than state of the art solvers. This consequently speeds up the run-time to the convergence proper.

The Structure of the Thesis

The thesis consists of three parts:

- Chapter 2 provides an overview of the state of the art in integer linear programming and the literature describing it. A few, albeit straightforward, original results are included, obtained by drawing certain connections between integer linear programming, on one hand, and Logic, Computational Complexity, and Number Theory on the other.
- Part II provides a critical overview of three alternative approaches to integer linear programming, which have been presented in three papers [16, 12, 45] at the Integer Programming and Combinatorial Optimization (IPCO) conference in recent years. Original probabilistic results suggest why such approaches may not work particularly well in general.
- Part III presentes a novel alternative approach to integer linear programming, based on "useful structures", and gives three examples of such useful structures: aggregations of variables, colouring components, and precedence-constrained components. Computational complexity of the problems is studied, alongside probabilistic analyses of the structures and heuristics for finding them.

Throughout the text, original results are given as lemmata and theorems, together with proofs, whereas results of other authors are given as propositions, with a reference, but without a proof.

There are two appendices:

- Appendix A describes an extension of a well-known optimisation programming language, so as to support semi-automated reformulation, and the tools for picking the best reformulation, based on a benchmark set of instances.
- Appendix B describes a tool for tuning parameters of integer programming solvers, which had not been finished, before the publication of some very similar work by Fatma Kilinç-Karzan [264] and Hutter et al. [247, 245, 246], and hence will not be finished.

Unlike in the main body of text, where only the constraint matrix and right-hand-side and objective vectors are required as input, the appendices focus on families of instances:

- formulated in an algebraic modelling language allowing for separation of model and data

- (optionally) with multiple data files to accompany a single model.

Although this is currently not the standard input to general-purpose integer linear programming solvers, it seems very natural.

The Contribution of the Thesis

To a developer of integer programming solvers, the following prototypes could be of interest, outside of the broad concepts of a "useful" structure and "structure pool":

- a primal heuristic for integer programming with algebraic reformulations and subsequent diving

- a method for discovering compact formulations of graph colouring in matrix (A, b) of an integer linear program, together with an augmented Lagrangian method for a novel semidefinite programming relaxation of the corresponding extension of the graph colouring problem

- a programming language for optimisation with semi-automated reformulations, complete with an open-source parser with Boost Spirit, which makes it easier to manage compact integer programming formulations and pick the appropriate one

To theoretical computer scientists, integer programming may seem too general a problem to tackle. Indeed, the present-best worst-case run-time lower bound for integer programming in n variables of arity a in realistic models of computation is $\Omega(n^2 \log(a+1))$ [312], as it seems. The following original results could, however, be of interest:

- tight lower bounds of the order of $\Theta(n)$ on tree-width, branch-width, clique-width, NLC-width, and rank-width of graphs drawn from a simple random model $G(n, \frac{1}{2})$ and the corresponding matrices, asymptotically almost surely, together with related straightforward probabilistic results and conjectures on unimodular submatrices and the size of a border in block-diagonal forms

- the definition of a structure in integer programming (2.35), related to kernelisation [220], and its restriction to a useful structure (5.9), formal statements of a number of problems (6.1, 6.2, 6.3, 7.4, 7.5, 8.4) related to the notion of a structure in integer programs, or equivalently, extracting certain submatrices from a matrix, together with proofs of $\mathcal{NP}$-Hardness or polynomial-time algorithms

- a probabilistic analysis of variable-aggregating heuristics for integer linear programming

- an optimal transformation of graph colouring on graph G to multi-colouring of a possibly much smaller graph G', which is computable in polynomial time

- semidefinite programming relaxations of a number of extensions of graph colouring, which correspond to one of the useful structures, together with a conjecture concerning asymptotic optimality of convergence rates of a certain augmented Lagrangian method for solving them.

A number of connections between Logic, Computational Complexity, Number Theory, and Probabilistic Combinatorics on one hand and practical challenges encountered in the development of solvers for integer programming could also be of interest.

To operations research practitioners and open-source enthusiasts, the following open-source programs could be of interest:

- a free tool for deciding which one, out of a number of succinctly described integer linear programming formulations, performs best on a given dataset

- a free translator of a large subset of OPL [400], the algebraic modelling language, to the sparse-matrix format known as ".lp"

- a preliminary version of free tools, which take a mathematical model written in OPL, the optimisation programming language, and either set it in type using LaTeX or translate it to code in C++ using Concert libraries

- a free toolkit for the development of solvers for a certain kind of timetabling problems based on integer programming decompositions.

This software is available on-line at:

- http://plosar.com
- http://philipaa.sf.net
- http://memos-solvers.sf.net
- http://discretisation.sf.net

This thesis tries to be many things to many people, which may be a bad omen. It is hoped, nevertheless, that it could be of some use to at least one person. If it was useful to you, please cite it or let me know otherwise.

Chapter 2

The State of the Art in Integer Linear Programming

The possibility of formulating many important applicational problems as diophantine programming problems has been recognized for years by many other workers in the field of linear programming. However, such a formulation has hitherto been almost exclusively a dead-end.

Abraham Charnes, *Review of Gomory's* Outline of an algorithm for integer solutions to linear programs *(MR102437)*

This chapter provides an overview of the state of the art in general-purpose integer linear programming solvers. Firstly, it describes the problem and its complexity. Subsequently, it describes the branch-and-cut approach to solving the problem by decomposition into a number of subproblems, which is employed by all modern general-purpose solvers, together with known complexity results and heuristics for the subproblems. A few, if straightforward, original results are included.

2.1 Integer Linear Programming

In many optimisation problems, the number of distinct configurations to choose a solution from is finite, but grows exponentially with some natural measure of the input. When one needs to find the configuration maximising or minimising a linear objective function, one may be able to formulate the problem as an integer linear program, and solve it using a general-purpose integer linear programming solver.

2.1.1 Problem Definitions

Let us start with a simple decision problem:

Problem 2.1. INEQUALITY-CONSTRAINED INTEGER FEASIBILITY (FEAS): Given an $m \times n$ integer matrix A and an m-vector b of integers, decide if there exists $x \in \mathbb{N}^n$ such that $Ax \leq b$.

where $\mathbb{N}$ denotes the set of natural numbers including zero. There are two important special cases:

Problem 2.2 ([259])**.** KNAPSACK: Given an n-vectors a, c of integers, and integers b, k, decide if there exists $x \in \{0,1\}^n$ such that $ax \leq b$ and $c^T x \geq k$.

Problem 2.3 ([274])**.** SUPPORT-2 INTEGER PROGRAMMING FEASIBILITY: Given an $m \times n$ integer matrix A with at most 2 non-zero entries per row, an m-vector b of integers, and an n-vector c of integers, decide if there exists $x \in \mathbb{N}^n$ such that $Ax \leq b$.

The optimisation counterpart of INEQUALITY-CONSTRAINED INTEGER FEASIBILITY is:

Problem 2.4. INEQUALITY-CONSTRAINED INTEGER PROGRAM (IP): Given an $m \times n$ integer matrix A, an m-vector b of integers, and an n-vector c of integers, find $x \in \mathbb{N}^n$ such that $Ax \leq b$ and $c^T x$ is minimised or report no such x exists.

Its equality-constrained variant is:

Problem 2.5. EQUALITY-CONSTRAINED INTEGER PROGRAM: Given an $m \times n$ integer matrix A, m-vector b of integers, and n-vector c of integers, find $x \in \mathbb{N}^n$ such that $Ax = b$ and $c^T x$ is minimised or report that no such x exists.

An important special case is:

Problem 2.6. SINGLE EQUALITY CONSTRAINT INTEGER PROGRAMMING FEASIBILITY (SCIPF, [295]). Given a non-negative vector $a \in \mathbb{Z}^n$ and an integer b, does there exist $x \in \mathbb{Z}^n$ satisfying $a^t x = b, x \geq 0$?

Notice it is analogous to KNAPSACK above.

2.1.2 Complexity

Considering that the knapsack problem is a special case of Inequality-constrained Integer Feasibility with a single constraint, and that one can introduce slack variables in order to solve inequality-constrained problems using Equality-constrained Integer Feasibility, it is easy to see that both problems are $\mathcal{NP}$-Hard[1]:

Proposition 2.7 (Karp [259])**.** Knapsack is $\mathcal{NP}$-Complete.

Proposition 2.8 (Leuker [295])**.** Single Equality Constraint Integer Programming Feasibility is $\mathcal{NP}$-Complete.

Proposition 2.9 (Lagarias [274])**.** Support-2 Integer Programming Feasibility is $\mathcal{NP}$-Complete.[2]

Considering that there is a pseudo-polynomial time algorithm for knapsack, it should not be too surprising that:

Proposition 2.10 (Lenstra, [282])**.** There is a polynomial algorithm for Inequality-constrained Integer Feasibility, when the number of variables n is fixed.

Proposition 2.11 (Papadimitriou, [337])**.** There is a pseudopolynomial algorithm for Equality-constrained Integer Program, when the number of constraints m is fixed.

This is due to the intuitive, but non-trivial:

Proposition 2.12 (Papadimitriou, [337])**.** Let A be an $m \times n$ integer matrix and b an m-vector, both with entries from $\{0, +1, \ldots, +a\}$. Then if $Ax = b$ has a solution $x \in \mathbb{N}^n$, it also has one in $\{0, 1, \ldots, n(ma)^{2m+l}\}^n$.

Proposition 2.12 is also the basis of proofs that integer linear programming is in $\mathcal{NP}$. The present-best unconditional lower bound seems to be:

Proposition 2.13 (Meyer auf der Heide, [312])**.** In random access machines, there is $\Omega(n^2 \log(a+1))$ lower bound on the run-time of a decision problem corresponding to Equality-constrained Integer Program in n variables of arity a.

This may well illustrate the limits of our understanding of computational complexity in theory.

[1] See the seminal paper of Cook [118] or any textbook [26] for the definitions. For the proofs, see [257, 374].

[2] The proof is non-trivial. It uses a reduction from Good Simultaneous Approximation, given as Problem 2.14 in the following section.

2.1.3 The State of the Art

In practice, state-of-the-art integer programming solvers decompose one $\mathcal{NP}$-Hard problem into a number of $\mathcal{NP}$-Hard problems, some of which may again be formulated as integer programs, some of which are actually often solved using a recursive call to the integer programming solver.

The search proceeds by visiting nodes of an implicit binary search tree, where each node of the tree corresponds to a distinct linear programming relaxation, and each edge corresponds to the addition of a single variable bound. Initially, the solver tries to find an integer feasible solution, which is $\mathcal{NP}$-Hard in general. Most solvers also try to simplify the instance in a process known as "domain propagation" or "constraint propagation" or "bound strengthening" and "(node) pre-solving" or similar. Subsequently, the solver obtains the optimum of a linear programming relaxation at the current node of the tree, which bounds the objective function values encountered in the corresponding sub-tree. Subsequently, the solver tries to find inequalities which are valid for all integer solutions, but violated by the current linear programming relaxation, which is $\mathcal{NP}$-Hard, and resolve the relaxation. Subsequently, a variable with fractional value v_i in the relaxation is picked, and two new nodes of the search tree are formed, where $x_i \leq \lfloor v_i \rfloor$ in one, and $x_i \geq \lceil v_i \rceil$ in the other. The choice of the variable to branch on, so that a certain criterion is maximised, is again $\mathcal{NP}$-Hard for several natural criteria. If a solution is available, whose value is better than the optimum of a linear programming relaxation at a certain node, the sub-tree corresponding to that node can be disregarded, as it is guaranteed not to contain the optimum. Out of the nodes of the search tree that were formed previously, but not yet visited or pruned, we need to pick the node to visit next. Tests on standard benchmarks [132] suggest, however, that this final decision is not particularly important. Once the node to visit next is chosen, the process is repeated. See Figure 2.1 for a schematic illustration.

The idea of branch-and-bound is often attributed to Land and Doig [277]. The addition of cuts at the root node has been suggested by Crowder et al. [125] and extended to the addition of cuts at other nodes by Grötschel et al. [304]. The name of "branch-and-cut" together with the introduction of the "cut pool" is dut to Padberg and Rinaldi [335]. The use of domain propagation has been studied by a number of authors [371, 9], independently. In the following sections, we describe the sub-problems and routines for solving them in more detail.

2.2 Primal Heuristics

Some of the most important heuristics within integer programming solvers try to find the first integral feasible solution, preferably at the root node of the search tree.

2.2.1 Background

The problem of finding the (first) integral feasible solution is closely related to the following problem in Number Theory:

Problem 2.14. Minimum Good Diophantine Approximation in l_∞-norm (GDA, [365]): Given a rational vector $\alpha = (\alpha_1, \ldots, \alpha_d) \in \mathbb{Q}^d$ and an integer n, find an integer Q, $1 \leq Q \leq n$, such that $||Q\alpha \mod \mathbb{Z}||_\infty := \max_{i=1}^{d} \min_{n \in \mathbb{Z}} |Q\alpha_i - n|$ is minimised.

It is known that:

Proposition 2.15 (Rössner and Seifert [365])**.** There exists no polynomial-time algorithm which for $\alpha \in \mathbb{Q}^d$ and integer n computes a number Q, $1 \leq Q \leq 2^{\log^{\frac{1}{2}-\gamma} d} n$ where for an arbitrarily small positive γ

$$||Q\alpha \mod \mathbb{Z}||_\infty \leq 2^{\log^{\frac{1}{2}-\gamma} d} \min_{q=1}^{n} ||q\alpha \mod \mathbb{Z}||_\infty,$$

unless $\mathcal{NP} \subseteq \text{DTIME}(n \,\text{poly}(\log n))$.

We make the connection precise when we use the result in the proof of Lemma 2.16 below.

2.2.2 Complexity

See Section 2.1.2, for results related to deciding if an integer program is feasible and what is the optimum. Notice, in particular, Problem 2.3 and Proposition 2.9, which illustrate that deciding the feasibility of an integer program is hard even in very restricted cases. An interesting question remains, however, whether solutions within certain bounds of optimum can be obtained in polynomial time, when the feasibility is guaranteed. This is often be formalised as a-approximation: Let us have a minimisation problem with a positive real-valued objective function $f(S)$ for solution S and

optimal solution $OPT(I)$ for instance I. An algorithm A is said to be a-approximate for the problem, if it produces solution $A(I)$ with objective function $f(A(I)) \leq af(OPT(I)), a \geq 1$ for any instance I.

Inapproximability within Almost Polynomial Factor

One can show:

Lemma 2.16. Unless $\mathcal{NP} \subseteq \text{DTIME}(n \operatorname{poly}(\log n))$, there exists no algorithm for SUPPORT-2 INTEGER PROGRAMMING that is $2^{\log^{\frac{1}{2}-\gamma} d}$-approximate and runs in polynomial time, even when the existence of a feasible solution is guaranteed.

Proof. There is a polynomial-time reduction between MINIMUM GOOD DIOPHANTINE APPROXIMATION IN l_∞-NORM and SUPPORT-2 INTEGER PROGRAMMING. Let us have an integer program in $d+1$ variables and $2d+2$ constraints:

$$-s_1 b \leq s_2 \left(\frac{a_i b}{b_i} y - b x_i \right) \leq s_1 b, 1 \leq i \leq d \tag{2.1}$$

$$1 \leq y \leq N \tag{2.2}$$

This reduction has been used by Lagarias [274] elsewhere previously. □

2.2.3 The State of the Art

Heuristics in combinatorial optimisation, in general [213], and primal heuristics in integer programming, in particular, are typically based on:

- Relaxation: enlarge the feasible region, for instance by ignoring certain constraints or variables, and try to make the "relaxed" solution feasible for the original problem afterwards. For example, many heuristics use rounding in linear programming relaxation, which ignores the integrality constraints. Many other "diving" heuristics solve a subproblem limited to "core" variables [351, 244], with the hope it will be possible to extend it to a solution of the complete problem
- Restrictions: shrink the feasible region, so that it resembles a well-known problem, such as network flows. If there is a solution in the "restricted" feasible region, it is also a solution to the original problem

- Randomisation: attempt to generate a solution for a sub-problem repeatedly, randomly, until a usable candidate is found
- Replacement objective: Change the objective function so that it is no longer equivalent, but loosely speaking "pointing in the same direction".

There is now a wide variety of published heuristics.

We first provide an extensive overview of published work, and subsequently an overview of heuristics implemented in current solvers. A brief overview is also provided in [287, 9, 27, 48, 8]. The only recent account of computational experience is, as it seems, in [9, 48, 8].

Search without Relaxations

A number of algorithms have been proposed, which do not use linear (or convex programming) relaxations. Those proposed prior to 1990 [30, 279, 182, 142, 349, 238, 271, 291, 152, 180] can perhaps be considered pioneering work. Zanakis [419] presents an empirical comparison of three of the early approaches. There are also a number of more recent proposals [38, 5, 343, 344, 326, 345, 346, 327, 347, 348, 317, 329]. None of these does, however, seem to be implemented in modern solvers, perhaps due to them not being competitive.

Rounding within a Single LP Relaxation

The most widely used class of primal heuristics are based on rounding within a single linear programming relaxation:

- Randomized rounding [360, 50, 51, 385]: Round all variables randomly up or down and check if the solution is feasible
- Simple rounding [106]: For each variable, perhaps randomly chosen, check if rounding preserves feasibility, and round
- Shifting and rounding with look-ahead [48]: for each variable, check if rounding it preserves feasibility, and if not, whether there is a sequence of k further roundings (or bound-changes) that would restore feasibility
- Iterated rounding [87]: Round and fix variables close enough to integer values, resolve for the remaining variables, repeat while possible

- Ceiling point algorithms [370]: move towards various "ceiling points" in turn and try rounding, where a "ceiling point" for constraint i is x such that $a_i^T x \leq b_i$ and for each $j = 1, \ldots, n$, $a_i^T x + a_{ij} \geq b_i$
- Vector-length diving [48]: Try to round variables in the increasing order of the product of the fractional part of a value in the relaxation, the objective function coefficient, and the largest absolute value of a coefficient associated with the variable
- Coefficient diving [48] and active-constraint methods [342]: Try to round variables in the order of the numbers of constraints that the variables occur in
- Pseudo-cost or reduced-cost diving [377, 393, 303, 48, 405]: Try to round variables in an order of some estimate of the impact on the variable on the objective function value
- Further variants of rounding (HEURIS4, BEST21, RSS1, [106, 107])
- Local-search in a linear programming relaxation [406]: search FLIP neighbourhoods defined by rounding each variable up or down. The mechanism for navigating the search space could be a tabu-list [193, 130, 262, 4, 288, 199, 200, 289, 124, 402, 194, 195, 197, 237], where one cannot round one variable twice within a fixed number of iterations
- Fractional diving and relaxation-enforced neighbourhood search (RENS, [48, 49]): search neighbourhoods defined by tightening bounds on general integer variables to $\lfloor x \rfloor$ and $\lceil x \rceil$, where x is the value in the presently-obtained linear programming relaxation
- Relaxation-guided variable neighbourhood search [357, 356, 358]: Search neighbourhoods, which are again instances of integer programming, in the order of their relaxations' objective value.

Dual fixing is possible also in non-linear relaxations [236], again by studying the Lagrange multipliers for constraints $x_i \leq 1$ and $x_i \geq 0$.

Diving with a Different Objective

A class of primal heuristics perhaps the closest to our work presented in Chapter III is based on modifying the objective function, albeit often within the same dimension:

- Direct search methods [142, 238, 152, 403], sometimes called interior path methods [238, 152], whose best-known examples are the feasibility pump [157, 10, 47, 75], objective feasibility pump [48, 10], and their variants: Alternate steps in the direction of optimality and integrality.
- Directional rounding [248, 199, 200, 196] and linesearch diving [48] or "path relinking" [248] approaches: Try to use multiple linear programming relaxations, such as those obtained at the root node and at the present node, and try to round or dive taking the difference into account, interpreting it as a direction
- Variants of Benders' decomposition [123, 361, 417, 7]: Try to decompose the problem, for instance, into the integer and non-integer parts and alternate steps in the direction of the objective restricted to the two parts, perhaps with a term to obtain a solution close to the previous one
- Smooth and dive [261]: Try to add the weighted sum of the product of each binary variables with its complement to the objective function. Several variants of this scheme have been investigated [129], recently.

Certain attempts at modifying objectives have also appeared in work on genetic algorithms for finding feasible solutions to integer programs [243], although their computational performance does not seem to be comparable to the state-of-the art general-purpose solvers.

Diving Integrated in an LP Solver

Another class of primal heuristics, albeit perhaps of limited recent interest, is based on the accommodation of integrality constraints in the simplex method:

- Integral simplex methods [206, 234, 235]: Pivot in a simplex-like tableau, keeping values integral.
- Pivot and complement [33]: considering integrality is equivalent to all slack variables, except for upper bounds, being in the basis of the simplex algorithm, one can try to perform pivots to that effect. This can be further integrated with searching a Flip neighbourhood and bounds on the objective value [222, 3, 290, 4]
- Pivot and shift [34]: Try to pivot towards integrality of general mixed integer programs using a local branching [159] procedure

- Pivot and Gomory cut [187]: Try to choose pivots depending on their interactions with Gomory cuts

- Pivot, cut, and dive [145]: Try to pivot towards integrality and derive (so called "Benders") cuts from impossibility of feasibility of integer solutions thus reached

Arguably, there are also many other heuristics. Some are implemented in solvers, but not published: The Gurobi Solver reportedly implements more than a dozen primal heuristics. Many more are published, but not widely implemented: There is the dual multi-phase algorithm of Glover [191], or the convergent diving of Wilbaut [411]. There is also the intriguing octahedral neighbourhood search (Octane) [31, 32], which is based on the correspondence between binary vectors in $\mathbb{R}^n$ and facets of an n-dimensional octahedron. It tries to enumerate a number of the octahedron's facets in one direction, converts them to binary vectors, and checks for their feasibility. It seems, nevertheless, that the heuristics in the bulleted lists above are representative of the state of the art.

Very little is known about the suitability of these heuristics for various classes of problems, except perhaps for the survey of Berthold [48]. In particular, there are only a few theoretical analyses [337, 137, 201, 50, 52, 51, 385], and these are mostly related to the performance on packing and covering problems. In Chapter III, we hence suggest a way to circumvent the lack of insight: We propose primal heuristics designed to recognise and suit a particular class of instances.

2.3 Improvement Heuristics

Improvement heuristics are related to primal heuristics. They try to find an improving feasible solution based on the feasible solutions found previously and the linear programming relaxation obtained at a given node.

2.3.1 Background: Polynomial-time local search

The class of search problems, for which local optimality can be verified in polynomial time, known as $\mathcal{PLS}$, is closely related to improvement heuristics for integer linear programming. Let us have a language L, such as finite strings of symbols from $\{0, 1\}$, in which one can describe any instance of a

problem. Let us have set D_L of instances, for each instance $i \in D_L$ a finite set $F_L(i)$ of solutions, for each instance $i \in D_L$ and solution $s \in F_L(i)$ a non-negative integral cost $c_L(i, s)$, and again for each instance-solution pair i, s neighbourhood $N(i, s) \subseteq F_L(i)$. Polynomial-time local search ($\mathcal{PLS}$, [252]) is then defined by:

- Algorithm $\mathcal{A}_L$, which given $i \in D_L$, produces a solution $s \in F_L(i)$, and runs in time polynomial in $|i|$.
- Algorithm $\mathcal{B}_L$, which given $i \in D_L$ and string s, decides whether $s \in F_L(i)$, and if so, computes $c_L(i, s)$, both in time polynomial in $|i|$.
- Algorithm $\mathcal{C}_L$, which given $i \in D_L$ and $s \in F_L(i)$, decides if there is any solution $s' \in F_L(i)$ with better cost than that for s, and if so, produces such solution, both in time polynomial in $|i|$.

The subset of $\mathcal{NP}$ search problems, for which there exists polynomial-time local search is denoted as $\mathcal{PLS}$-problems, or just $\mathcal{PLS}$. There are a number of alternative definitions[3] of $\mathcal{PLS}$, which exhibit its central position in logic and complexity.

A $\mathcal{PLS}$-problem L is $\mathcal{PLS}$-reducible to a $\mathcal{PLS}$-problem K, if there are polynomial-time computable functions f and g such that:

- f maps instances i of L to instances $f(i)$ of K
- g maps instance-solution pairs $i, f(i)$ to a solution of i
- for all instances i of L, if s is a local optimum of $f(i)$ of K, then $g(i, s)$ is a local optimum for x.

The notion of $\mathcal{PLS}$-Completeness follows.[4]

Proposition 2.17 (Johnson el al. [252])**.** If co$\mathcal{NP} \neq \mathcal{NP}$, no problem in $\mathcal{PLS}$ is $\mathcal{NP}$-Hard.

Furthermore:

[3] Without introducing the precise definitions, let us consider the definition of Buss et al. [92] using Σ_1^b definable functions of T_2^1.

[4] Without introducing the precise definitions, let us consider, for example, the so-called Flip neighbourhoods, are complete for MaxCut, Max2SAT, Graph Partitioning, and the Travelling Salesman problem. Walser [406] wrote a book on applications of Flip neighbourhoods in integer programming.

Proposition 2.18 (Orlin el al. [331])**.** If there exists a near-exact neighbourhood for a $\mathcal{PLS}$-complete problem, then $\mathcal{PLS} = \mathcal{P}$, where we define neighbourhood N to be near-exact if the objective function value of every local optimum is worse than that of at most a polynomial number of other feasible solutions.

2.3.2 Problem Definitions

Problem 2.19. AUGMENTATION (AUGM, [375]): Given an $m \times n$ integer matrix A, an m-vector b of integers, an n-vector c of integers, and a feasible solution $x \in \mathbb{Z}^n, Ax = b$, find a feasible solution $y \in \mathbb{Z}^n$ such that $Ay = b, cy < cx$, or state that there is no such y.

2.3.3 Complexity

Firstly, it is clear that:

Proposition 2.20 (Schulz [375])**.** AUGMENTATION is $\mathcal{NP}$-Hard.

From Propositions 2.17 and 2.20, it follows that there are limitations to what can be achieved through polynomial local search improvement heuristics. Orlin el al. [331] have shown a closely related result: a problem has a fully polynomial-time approximation scheme if and only if it has a fully polynomial-time augmentation scheme, for the natural definitions of the two terms.

2.3.4 The State of the Art

In improvement heuristics, one often encounters two terms from "population-based" heuristics:

- Local search: Search within a "neighbourhood" of an integer feasible solution, using some definition of what neighbourhood is
- Mutation: Typically pseudo-random perturbation of an integer feasible solution, perhaps leading to another integer feasible solution
- Cross-over: Obtaining a new integer solution, perhaps integer feasible, by combining two or more integer feasible solutions

- Memetic approaches: After obtaining an integer feasible solution by cross-over or similar, try to improve it using local search.

Building upon these simple ideas, a wide variety of approaches have been proposed:

- Mutation of a single solution [150, 229, 367]
- Mutation and crossover of multiple solutions [40, 169, 296, 328, 265, 99, 48, 367]: try to combine solutions, while perturbing randomly and improving greedily the combined solutions, if integer feasible
- Local search within neighbourhoods given by multiple solutions [25, 367]: try to search the neighbourhoods given by distinct values of variables encountered in multiple integer feasible solutions
- Relaxation induced neighbourhoods (RINS, [131, 133]): search neighbourhoods given by fixing values of those variables that have the same value in the present-best feasible solution and the linear programming relaxation at the current node. Further studies [133] propose to use bounds on the objective function value, but not to use bounds on variables given from branching in the neighbourhoods, or to modify the objective using pseudo-costs [410].
- Local branching [248, 158, 161, 229, 159]: searching neighbourhoods within distance k from the present-best integer solution
- Distance Induced Neighbourhood Search (DINS, [186]): Combine soft-fixing of local branching with hard fixing of relaxation induced neighbourhood search.

It seems, however, that only the most simple approaches perform well across a broad range of instances. The apparent difficulty is the complexity of searching the neighbourhoods: Local branching takes the original instance and adds a very dense constraint. Unless the pre-processing, which is a high-order polynomial-time constraint-propagation algorithm, results in considerable simplification, searching the neighbourhood may, in practice, be as hard as solving the original problem. Commercial solvers, which implement local branching and relaxation induced neighbourhood search seem to have the functionality disabled by default.

2.4 Branching Heuristics

2.4.1 Background: Optimal Proof Systems

Closely related to the class of branching heuristics is the notion of optimal proof systems [359]. Broadly speaking, an important open problem in logic corresponds to the question of whether there exists a polynomial-time computable branching rule, which for every fixed feasible instance of EQUALITY-CONSTRAINED INTEGER FEASIBILITY, is not polynomially worse than any other branching rule, with respect to the number of nodes visited. This question is closely related to $\mathcal{NP} \neq \text{co}\mathcal{NP}$.

In order to understand the results, one needs to understand some definitions. In the definition of Cook and Reckhow [119], a proof system for language $L \subseteq \Sigma^*$ in alphabet Σ is a polynomial-time computable function from all strings onto theorems, $f : \Sigma^* \to L$. A string $w \in \Sigma^*$ with $f(w) = x$ is called an f-proof for $x \in L$. In the definition of Krajíček and Pudlák [272], when f and g are proof systems for L, f (p-)simulates g, if for any g-proof w there is an f-proof w_0 of length $|w_0| = |w|^{O(1)}$ computable in polynomial time, such that $f(w_0) = g(w)$. It follows that proof systems for L are quasi-ordered by the relation of simulation. A maximal element of the quasi-ordering for L is an optimal proof system for L. In general, it is unknown, however, whether an optimal proof system exists. Without introducing all the definitions, let us state two important arguments:

Proposition 2.21 (Krajíček and Pudlák [272])**.** If $\mathcal{NE} = \text{co}\mathcal{NE}$, then the set of tautologies (TAUT) has optimal proof systems. If $\mathcal{E} = \mathcal{NE}$, then the set of tautologies (TAUT) has p-optimal proof systems. If the set of tautologies (TAUT) has optimal proof systems, $\mathcal{NP} \neq \text{co}\mathcal{NP}$.

Notably, proof systems that do not introduce new variables, do not seem to be optimal [359]. On the other hand, a proof system that does introduce new variables[5] can be optimal, if there is limited advice as to what the new variables should be:

Proposition 2.22 (Cook and Krajíček [117])**.** There exists an optimal proof system which is computable in polynomial time with one bit of advice.

[5]The system in question is polynomially-bounded Extended Frege. Without introducing all the definitions, the paper uses: Provability of $\text{co}\mathcal{NP} \subseteq \mathcal{NP}/O(1)$ gives a polynomially-bounded Extended Frege proof system with constant advice. Next, it shows: Provability of $\mathcal{NP} = \text{co}\mathcal{NP}$ in a polynomial-time theory implies that Extended Frege is polynomially bounded.

and there are optimal heuristic proof systems [239], which allow for a small number of false positives.

Notice that these results were phrased in terms of tautologies and arbitrary proof systems. For Equality-constrained Integer Feasibility, similar results could be shown easily by extending the arguments of Sadowski [369] made in the context of propositional satisfiability.

2.4.2 Problem Definitions

In practical integer linear programming, the length of the proof, expressed as the number of nodes visited, is hard to work with. One hence uses myopic "proxies" for the minimisation number of nodes visited, such the maximisation of the improvement of the bound or the reduction of the dimension of the problem by fixing variables. Even the two proxies mentioned are hard to optimise. The former, as an optimisation problem, is:

Problem 2.23 ([297]). Greatest lower bound from a disjunction: Given an $m \times n$ integral matrix $A \in \mathbb{Z}^{m \times n}$, an m-vector $b \in \mathbb{Z}^m$, an n-vector $c \in \mathbb{Z}^n$, find $d \in \mathbb{Z}^n, r \in \mathbb{Z}$ such that $\min\{c^T x_L, c^T x_R\}$ is maximized, where x_L, $x_R \in \mathbb{R}^n$ are optimal solutions of linear programs $(\min c^x$ s. t. $Ax \leq b, d^T x \leq r)$ and $(\min c^x$ s. t. $Ax \leq b, d^T x \geq r + 1)$.

The decision version is:

Problem 2.24 ([297]). Lower bound from a disjunction: Given $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m, c \in \mathbb{Z}^n$, and integer k, does there exist disjunction $d \in \mathbb{Z}^n, r \in \mathbb{Z}$ such that $\min\{c^T x_L, c^T x_R\} \geq k$, where x_L, $x_R \in \mathbb{R}^n$ are optimal solutions of linear programs $(\min c^x$ s. t. $Ax \leq b, d^T x \leq r)$ and $(\min c^x$ s. t. $Ax \leq b, d^T x \geq r + 1)$.

The dimension reduction is captured by:

Problem 2.25 ([297]). Disjunctive infeasibility: Given $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m, c \in \mathbb{Z}^n$, does there exist disjunction $d \in \mathbb{Z}^n, r \in \mathbb{Z}$ such that the feasible region of $(Ax \leq b, d^T x \leq r)$ and $(Ax \leq b, d^T x \geq r + 1)$ are empty?

2.4.3 Complexity

The $\mathcal{NP}$-hardness of branching variable selection in propositional satisfiability (SAT) has been shown by Liberatore [283]. Mahajan and Ralphs [297] extended this to all the problems above:

Proposition 2.26 (Mahajan and Ralphs [297]). DISJUNCTIVE INFEASIBILITY is $\mathcal{NP}$-Complete even for binary programs using 0-1 hyperplanes.

2.4.4 The State of the Art

Currently available ILP solvers branch on individual variables whose values are fractional in the relaxations [315, 275, 284, 27, 171, 276], if one disregards special ordered sets. Trivial approaches, such as user-given priorities [166], objective function coefficient [30, 182], maximum integer infeasibility, i.e. $v - \lfloor v_i \rfloor$ closest to 0.5, and minimum integer infeasibility, i.e. non-zero minimum of $v - \lfloor v_i \rfloor$ or $v - \lceil v_i \rceil$, have been replaced only by more sophisticated criteria for the choice of one out of the variables, rather than among the much larger number of disjunctions available by branching on general hyperplanes.

Modern solvers try to estimate some "proxy" of the effect of branching on a given variable on the time to convergence:

- The change of bound, or equally, the depth of the cut: the difference between the minimum or maximum of the objective function values after branching and its present value
- The change [342]: captured by the Euclidean distance of the LP relaxation after branching from the present one, for instance
- Dimension reduction [297]: the (minimum) reduction of the dimension of the problem by variable fixing, made possibly after branching
- Inference [11, 8]: the sum of the numbers of reductions made possible in the subproblems obtained by the branching
- Entropy [189]: minimise entropy, where the entropy of a binary variable is zero, if it is integral, and $-v \log_2(v) - (1-v) \log_2(1-v)$, if its value v is fractional
- Conflict activity [7, 8]: the number of appearances of each variable in recent cuts, inspired by solvers in propositional satisfiability.

Some early approaches, such as minimum integer infeasibility branching, were based on trying to achieve infeasibility of one of the sub-problems obtained by branching. This, however, no longer seems a very appealing goal.

There are a number of approaches to estimating the values of the effect of branching on a given variable:

- Pseudo-cost branching: From a fixed value, e.g. all-ones vector, objective function coefficients, or average of pseudocosts of a small sample
- Full strong branching [25]: Estimate the value for all variables fractional in the relaxation by performing a small number of dual simplex iterations in each of the possible subproblems, so as to estimate the change of the objective function value
- Limited strong branching [9, 10, 8] and look-ahead branching [190]: Apply strong branching only to a small limited number of variables
- Reliability branching [9, 10, 8] and other hybrids of strong branching with pseudocosts [284, 9, 11]: Apply strong branching to unreliable estimates
- Information-theoretic approaches [189]: use estimates based on the entropy of the variable across all other nodes explored so far
- Learning-based approaches [264]: Use estimates learned previously on a similar set of instances.

Further differences are in the number of variables, for which to update the estimate, after each decision:

- Pseudo-cost branching [44, 166, 179]: Update only the value pertaining to the variable previously branched on, but possibly taking into account the effects of branching of the given variable elsewhere in the tree
- Full strong branching: Update estimates for all variables fractional in the relaxation
- k-Best [9, 10, 8]: Update variables fractional in the relaxation, taken in the order of a cheap estimate, until there are no new best candidate branching variables found within k latest
- Active-set branching [91, 342]: Update variables fractional in the relaxation with at least a certain number of locks.

Yet further differences are in the ways how to integrate the estimates of the effects of rounding the non-integral value up and down into one value:

- Take the value for one of the directions, up or down, constantly throughout [44, 166, 179]

- Take either the minimum or the maximum of the values for the two directions [44, 166, 179]
- Take a convex combination of maximum and minimum, for $\mu = 1/2$ [9], $1/3$ [215], $1/6$ [9, 8]:
$$f(q^-, q^+) = \mu \max\{q^-, q^+\} + (1 - \mu) \min\{q^-, q^+\}$$
- Take the adjusted product of the maximum and minimum [9, 10, 8]:
$$f(q^-, q^+) = \max\{q^-, \epsilon\} + \max\{q^+, \epsilon\}$$
- Shadow price [166, 350]: Use the shadow price of the corresponding constraint in the dual
- Guided diving [133]: round up or down depending on the value in the present-best integer solution.

It should be noted that a number of further approaches, inspired by propositional satisfiability, are now emerging. Notably, there are hybrids of strong branching with pseudocosts and inference [9, 11, 8], and the intriguing concept of resolution search [111, 227], which replaces branching as recursive partitioning of the feasible region with recursive shrinking of the feasible region. It is yet to be seen, whether any of these approaches prove to be efficient.

2.5 Bounding and Cut Generation Heuristics

A key factor in the performance of an integer programming solver is the quality of the relaxation. The rather vague term "quality" covers both the gap achieved by the bound and the possibilities to obtain integer feasible solutions by rounding or similar procedures. As one could expect, there is a trade-off between the quality of the bound and the run-time required in computing it. Although there are reasons to believe the importance of non-linear relaxations will grow, most solvers still employ a linear programming relaxation and strengthen it by the addition of further linear inequalities ("cuts").

2.5.1 Problem Definitions

In order to improve linear programming relaxations, one needs to find a valid and violated inequality, also known as a "cut":

Problem 2.27. Separation (SEP, [375]): Given $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$, and point $z \in [0,1]^n$, find vector $a \in \mathbb{Z}^n$ and integer $\beta \in \mathbb{Z}$ such that $ax \leq \beta$ for all $x \in \mathbb{Z}^n, Ax \leq b$, and $az > \beta$, or state that such a vector a and number β do not exist.

Perhaps the most important class of cuts for integer linear programming, both historically and in modern solvers, are due to Gomory [204, 109, 205]. The class, where all variables are integral, is sometimes denoted Chvátal-Gomory cuts [204], but related cuts [205] are applicable to mixed integer programming as well. A Chvátal-Gomory (CG) cut for a constraint $ax \leq b$ with multiplier vector $\lambda \in [0,1)^n$ is an inequality of the form $\lfloor \lambda^T a \rfloor x \leq \lfloor \lambda^T b \rfloor$, where $\lambda^T a \in \mathbb{Z}^n$ and $\lambda^T b \notin \mathbb{Z}^n$. In theory, one could extend an instance of integer programming with all applicable Chvátal-Gomory cuts, to obtain an instance of integer programming denoted as the elementary closure, which could again be extended with all applicable CG cuts. In k rounds of this procedure, one obtains the convex hull of integer points. In general, k is finite; for integer programming with binary variables, in particular, k is $O(n^2 \log n)$ [146].

The problem of finding a violated Chvátal-Gomory cut is known as:

Problem 2.28. Chvátal-Gomory Separation (CG-SEP, [204]): Given $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$, and point $z \in [0,1]^n$, find vector $\lambda \in [0,1)^n$ such that $\lambda^T A \in \mathbb{Z}^n$, $\lambda^T b \notin \mathbb{Z}^n$, and $(\lambda^T A)x \leq \lfloor \lambda^T b \rfloor$ for all $x \in \mathbb{Z}^n, Ax \leq b$, and $(\lambda^T A)z > \lfloor \lambda^T b \rfloor$ or state that no such vector exists.

An important special case restricts λ to $\{0, \frac{1}{2}\}^n$:

Problem 2.29. $\{0, \frac{1}{2}\}$ Separation ($\{0, \frac{1}{2}\}$-SEP, [96]): Given $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$, and point $z \in \{0, \frac{1}{2}\}^n$, find vector $\lambda \in \{0, \frac{1}{2}\}^n$ such that $\lambda^T A \in \mathbb{Z}^n$, $\lambda^T b \notin \mathbb{Z}^n$, and $(\lambda^T A)x \leq \lfloor \lambda^T b \rfloor$ for all $x \in \mathbb{Z}^n, Ax \leq b$, and $(\lambda^T A)z > \lfloor \lambda^T b \rfloor$ or state that no such vector exists.

When a user-supplied oracle is used to find a cut instead of an in-built general-purpose method, it may be necessary to verify the cut is valid:

Problem 2.30. Violation (VIOL, [375]): Given $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$, vector $a \in \mathbb{Z}^n$ and number $\beta \in \mathbb{Z}$, decide whether $ax \geq \beta$ holds for all $x \in \mathbb{Z}^n, Ax \leq b$, and if not, find a feasible solution $y \in \mathbb{Z}^n, Ay \leq b$ with $ay < \beta$.

2.5.2 Complexity

Considering the results on equality of separation and optimisation [214], it is not too difficult to see:

Proposition 2.31 (Folklore, e.g. [375])**.** SEPARATION is $\mathcal{NP}$-Hard.

Proposition 2.32 (Caprara and Fischetti, [96])**.** CHVÁTAL-GOMORY SEPARATION is $\mathcal{NP}$-Hard.

Although there are fast separation heuristics [269], even the $\{0, \frac{1}{2}\}$ special case is hard:

Proposition 2.33 (Caprara and Fischetti, [96])**.** $\{0, \frac{1}{2}\}$ SEPARATION is $\mathcal{NP}$-Hard.

It is also hard to check for validity of a cut:

Proposition 2.34 (Schulz, [375])**.** VIOLATION is $\mathcal{NP}$-Hard.

2.5.3 A Selective Overview

Unlike in previous sections, we do not attempt to provide an exhaustive overview of the literature, due to its sheer volume. We should like to point out, however, that one often observes that the number of cuts required to strengthen a weak linear programming relaxation is simply too large to use in practice. Stronger relaxations, without the addition of further inequalities, can be had by generalising linear programming. Semidefinite programming (SDP, [42, 414]) is one popular generalisation of linear programming, replacing the vector variable with a square symmetric matrix variable and the polyhedral symmetric convex cone of the non-negative orthant with the non-polyhedral symmetric convex cone of positive semidefinite matrices. The primal-dual pair in the standard form is:

$$z_p = \min_{X \in \mathbb{S}^n} \langle C, X \rangle \text{ s. t. } \mathcal{L}_A(X) = b \text{ and } X \succeq 0 \qquad \text{(P STD)}$$

$$z_d = \max_{y \in \mathbb{R}^m, S \in \mathbb{S}^n} b^T y \text{ s. t. } \mathcal{L}_A^*(y) + S = C \text{ and } S \succeq 0 \qquad \text{(D STD)}$$

where X is a primal variable in the set of $n \times n$ symmetric matrices $\mathbb{S}^n$, y and S are the corresponding dual variables, b is an m-vector, C, A_i are compatible matrices, and linear operator $\mathcal{L}_A(X)$ maps symmetric $n \times n$ matrices to

vectors in $\mathbb{R}^m$. The ith element $\mathcal{L}_A(X)_i = \langle A_i, X\rangle$, and the adjoint is $\mathcal{L}_A^*(y) = \sum_i y_i A_i$. $M \succeq N$ or $M - N \succeq$ denotes $M - N$ is positive semidefinite. Note that an $n \times n$ matrix, M, is positive semidefinite if and only if $y^T M y \geq 0$ for all $y \in \mathbb{R}^n$. As all linear combinations with non-negative coefficients of positive semidefinite matrices are positive semidefinite, $X \succeq 0$ should again be seen as a restriction to a convex cone.

One can also treat inequalities explicitly in the primal-dual pair:

$$\begin{aligned} z_p &= \min_{X \in \mathbb{S}^n} \langle C, X\rangle \\ \text{s. t. } & \mathcal{L}_A(X) = b \text{ and } \mathcal{L}_B(X) \geq d \text{ and } X \succeq 0 \qquad \text{(P SDP)} \\ z_d &= \max_{y \in \mathbb{R}^m, v \in \mathbb{R}^q, S \in \mathbb{S}^n} b^T y + d^T v \\ \text{s. t. } & \mathcal{L}_A^*(y) + \mathcal{L}_B^*(v) + S = C \text{ and } S \succeq 0 \text{ and } v \geq 0 \qquad \text{(D SDP)} \end{aligned}$$

where d is a q-vector and linear operator $\mathcal{L}_B(X)$ maps $n \times n$ matrices to q-vectors similarly to $\mathcal{L}_A$ above. This form will be used in Chapter III.

For a number of particular problems, there are very good semidefinite programming relaxations, giving rise to a number of very good, so called "spectral", algorithms [21]. When only a general binary linear programming formulation is given as an input, one can use one of the general operators for deriving semidefinite programming relaxations [293, 278, 221] listed in Table 2.1. For a number of well-known $\mathcal{NP}$-Hard problems, a linear number of applications of such operators suffices to obtain the convex hull of integer solutions [293, 278, 221]. This could be seen as an alternative approach to integer programming, albeit not a particularly successful one, so far, considering that the repeated applications of known operators do produce super-exponentially large instances of semidefinite programming.[6] The use of a fixed small number of applications of the operator in a branch-and-bound schema may still result in an exponential number of nodes visited in the worst case [352], albeit perhaps very small number of nodes in expectation. A variant of this approach, utilising one application of such operators for a particular subsets of variables and constraints, is explored in more detail in Chapter III.

[6] Notice one cannot rule out an algorithm that would optimise over such a relaxation in polynomial time, yet, as it seems.

Operator	Dimension	Reference
Lovász-Schrijver N_+^t	$\Theta(2^{t-2}n^{t+1})$	Lovász and Schrijver [293]
Moment method Q^t	$\Theta(n^{2t})$	Lasserre [278]
Block diagonal $\tilde{L}^t$	$\Theta(1/(t+1)!\, n^{t+1})$	Gvozdenović et al. [221]

Table 2.1: Operators for deriving SDP relaxations of integer linear programs, with dimensions after t iterations given by Gvozdenović et al.

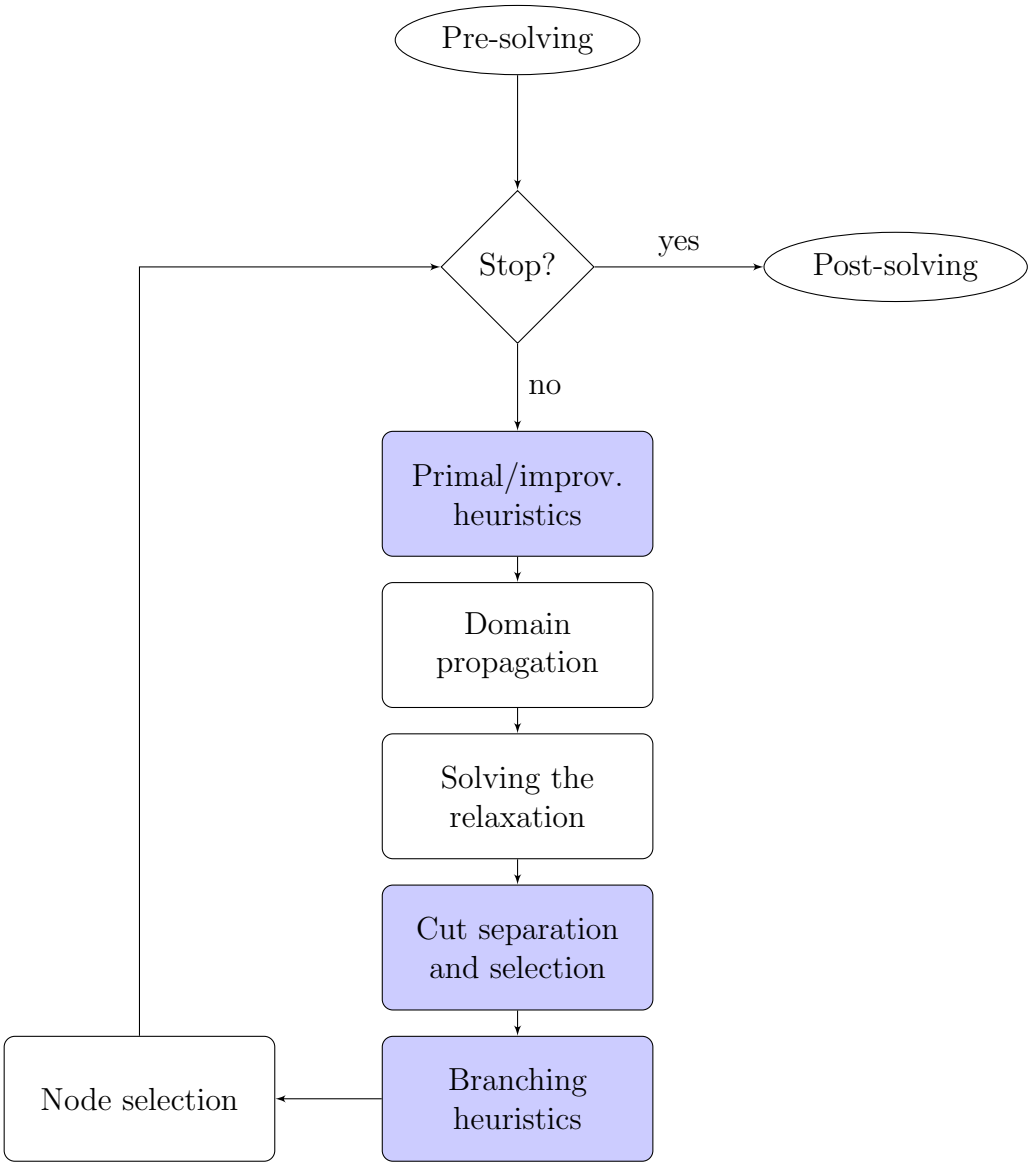


Figure 2.1: A schematic illustration of the main loop in state of the art integer linear programming solvers. The highlighted steps are discussed in further detail in this chapter.

Part II

Known Structures in Integer Programming

In mathematics,
the more abstract, the more valuable.

Martin Grötschel [211]

Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice.

Aho et al. [15]

Chapter 2 presented the branch-and-cut schema, which is employed in all state-of-the-art solvers for integer linear programming. This partpresents a number of known alternative approaches in a unifying framework, together with probabilistic arguments showing a number of them require run-time exponential in the number of variables, in expectation.

Alternative Approaches to Integer Programming

A number of alternative approaches to integer linear programming proceed in three steps:

1. A transformation of an instance of integer linear programming to a difficult subproblem and its easy to solve "complement", where the difficult sub-problem or "core" [244] need not necessarily be treated as an instance of integer linear programming

2. A solver for the difficult sub-problem, such as a dynamic programming recursion or a large-scale semidefinite program, whose run-time is hard to reduce further

3. A transformation of a solution of the difficult sub-problem to a solution of the original problem.

In theory, such a decomposition may have some appeal, as the run-time of both steps is often exponential only in some parameter(s) of the difficult subproblem, rather than in the number of variables or constraints of the original problem. This partformalises this framework and presents three well-known alternative approaches in the framework. The following theoretical

"counter-argument" is also made precise for a number of structures: there is little to be gained if the parameter is linear in the dimension of the problem in expectation.

In particular, we focus on:

- Approaches based on "width parameters" such as tree-width, branch-width, clique-width, rank-width, boolean-width, and motivated by the insight that many problems can be cast as a problem on graphs, and many graph problems are easy on trees. To find the decomposition of a graph into a tree and a mapping from the tree to the original graph minimising many width parameter is, however, very hard in practice and $\mathcal{NP}$-Hard in theory. In order to exploit the low width parameter, one can either use dynamic programming, or alternative relaxation-producing operators (e.g. Sherali-Adams), where the number of iterative applications required to reach an integer solution is bounded by a function of the parameter (e.g. tree-width)

- A number of approaches inspired by constrained single-commodity flow problems try to find a large "good submatrix", such as the largest totally unimodular submatrix, and consider the rest as the hard sub-problem. The extraction of many types of good submatrices is $\mathcal{NP}$-Hard. Dynamic programming can be used [17, 16] to exploit the good submatrix.

- An extension of the search for a good submatrix inspired by multi-commodity flow problems is to find a "block-structured" submatrix, where each matrix may turn out to be "good" in some sense. The computation of the least bandwidth of a matrix under row and column permutations is $\mathcal{NP}$-Hard, but there are good heuristics [128]. One can use the decomposition to generate cuts (MCF separator, [12]), for example.

The reasons for the focus on the three approaches are two-fold: First, the recent appearance of papers [16, 12, 45, 302] claiming their imminent applicability, and second, our ability to prove the above-mentioned negative probabilistic results.

We do not present a number of other approaches:

- Pure cutting plane approaches [420]: Formulate the convex hull of integer points as an exponentially large linear program and solve it, by representing the inequalities implicitly. Notice that branching is not

used in any related "outer approximation" approaches. Notice that the linear program is typically built up in "rounds" and the number of rounds required can be seen as the parameter.

- Approaches based on operators for deriving semidefinite programming relaxations, such as Lovász-Schrijver N_+^t [293], the block-diagonal $\tilde{L}^t$ [221], and the moment method Q^t [278]: Formulate super-exponentially large semidefinite program by repeated applications of the operator and try to solve it. Notice that the number of applications of the operator can be seen as the parameter.

- Approaches exploiting connections with Semi-algebraic Geometry (Polynomial Optimisation), notably the fact every n-dimensional polyhedron can be described by $2n$ explicit polynomial inequalities, shown by Grötschel et al. [212, 79]. The degree of the polynomial is a parameter that can be considered.

- Approaches using recursive basis reduction [282, 256, 120, 235]: Apply a certain rounding procedure and recursive basis reduction, which reduces a problem to a number of sub-problems in a lower dimension. The depth of the recursion seems to be a natural parameter.

- Approaches using basis reduction as a pre-processing [1, 341]: Apply basis reduction and use branching on integer linear combinations of vectors from the null-space of the constraint matrix to reach convergence within lower and upper bounds on variables. The dimension of the reduced basis can be considered as a parameter.

- Approaches exploiting "Gröbner bases" [398, 391] and algebraic properties of "Graver bases of the Graver bases" [372, 46] corresponding to the constraint matrix.

- Approaches exploiting connections with hard flow problems [226]: Decompose the constraint matrix into a sum of multiple matrices with the "consecutive ones" property and hence transform integer programming to a certain hard flow problem.

- Travelling salesman approaches: Billera and Sarangarajan [55] have shown every $0-1$ d-dimensional polytope is affinely equivalent to a face of the asymmetric traveling salesman polytope, albeit in a much higher dimension. Given the large amounts of work on traveling salesman polytopes [25], could one make use of the connection?

- Approaches exploiting Lovász local lemma [149, 319]: Srinivasan [385] suggests what relaxations can be rounded to integers. One could hence try to find the largest submatrix that is guaranteed to be feasible, or equally in binary integer programming, find the largest submatrix corresponding to an instance of propositional satisfiability of the neighbourhood structure that is employed in the constructive version of the lemma [319]. Notice that the constructive version uses local search.

- Approaches based on local search: One can use FLIP neighbourhoods, which are widely and successfully used in Propositional Satisfiability [406].

- Approaches based on sampling: A broad class of integer convex programs can be solved in time $O(2^d + 8^d\sqrt{n \ln n} \ln n) + d^{O(d)}\phi \ln n$ in expectation on numbers with $d^{O(1)}\phi$ bits, as $n \to \infty$ by the sampling technique of Clarkson [112]. This is based on the fact that all optima of an integer program in dimension n are determined by at most $2^n - 1$ inequalities each, which has been shown by Scarf [373, 41], and a clever use of basis reduction to find the constraints.

Many such approaches could also be cast in this framework, as suggested in Table 2.2. See Aardal et al. [2] for an analytical comparison of some of these approaches.

Table 2.2: Structures in integer linear programs.

Refs.	Structure	Algorithm $\mathcal{A}_1$	Algorithm $\mathcal{A}_2$
[302]	Tree-decompositions	Find a branch-decomposition	Cunningham-Geelen
[66, 59]	Good submatrices	Find a unimodular submatrix	Dynamic programming
[319]	Lovász local lemma	Find a submatrix, for which the lemma guarantees feasibility	Dynamic programming
[45]	Block-decomposition	Find a block-decomposition	Dantzig-Wolfe
[226]	Hard flows	Find a sum of "consecutive ones" matrices	Solve a flow problem
[212, 79]	Semi-algebraic geometry	Reformulate as polynomial optimisation	Solve it
[293, 221, 278]	Semidefinite prog.	Formulate a semidefinite program	Solve it
[420]	Pure cutting-planes	Find linear prog. cuts	Solve it
[1, 341]	Basis reduction	Basis reduction	Branching
[282, 120]	Recursive basis red.	Rounding	Recursion
[112]	Sampling	Basis reduction	Sampling

Fixed Parameter Tractability and the Notion of a Structure

Fixed Parameter Tractability has been introduced by Downey and Fellows [138]. In the definition by Flum and Grohe [164], a parameter is a function from problem instances to the set of natural numbers. A problem is fixed parameter tractable (FPT) with respect to fixed parameter p, if there exists a function f and algorithm A, such that there is an upper bound $f(p(I))n^{O(1)}$ on the run-time of algorithm A on instance I. Of particular interest are then cases, where f is singly-exponential, i.e. $O(c^{p(I)})$, which are sometimes denoted "bounded fixed-parameter tractable". Notice that, again, one fixes both the choice of the parameter and its value.

In kernelisation [220], an algorithm takes an instance i_1 of problem P and parameter p of value k and in polynomial time produces instance i_2 of problem P such that

- $|i_2| \leq |i_1|$
- i_1 has a solution if and only if i_2 has a solution
- $|i_2|$ is bounded from above by an arbitrary function g in k.

Problem P admits a polynomial kernel, if there exists kernelisation of P with parameter p of k, where $g = k^{O(1)}$. Rather trivially, problem P admits a polynomial kernel if and only if P is fixed parameter tractable, for some parameter p.

For example, when one fixes the number of variables, the algorithm of Lenstra [282] (Proposition 2.10) shows Inequality-constrained Integer Feasibility is fixed parameter tractable. When one fixes the number of constraints, algorithm of Papadimitriou [337] (Proposition 2.11) shows Equality-constrained Integer Program is fixed parameter tractable. It makes sense, however, to require p to be rather more different from the dimension of the input one denotes n. Another interesting example of a parameter, seemingly little studied so far, is the objective function value at the optimum. In many minimisation problems in combinatorial optimisation, the optimum is rather a small number k and it is easy to produce feasible solutions with value $k+1$ as well as to infer integer infeasibility of solutions with value $k-1$, which could perhaps be exploited. This, however, is not the focus of this part.

The main definitions are:

Definition 2.35. For every instance of Inequality-constrained Integer Program (Problem 2.4), consisting of $m \times n$ integer matrix A, an m-vector b of integers, and an n-vector c of integers, a *particular structure* is defined by three algorithms:

- Algorithm $\mathcal{A}_1$ finds sub-matrices $S_1, S_2, \ldots, S_s \subseteq A$ and produces the corresponding index-sets $B_i, C_i, 1 \leq i \leq s$ for rows and columns, respectively
- Algorithm $\mathcal{A}_2$ finds a solution to a problem whose input is (A, b, c, s) and $(B_i, C_i), 1 \leq i \leq s$, and whose output is T
- Algorithm $\mathcal{A}_3$ translates (A, b, c, s) and $(B_i, C_i), 1 \leq i \leq s$, to a solution of the original problem (A, b, c).

Notice that a particular structure generalises the notion of a kernel in that the intermediate form with $B_i, C_i, 1 \leq i \leq s$ does not necessarily have a more succinct description than the original instance, and specialises it to integer programming.

Definition 2.36. Two particular structures are in the *same inside* relation, if and only if for for every instance of Inequality-constrained Integer Program the index sets B_i, C_i obtained by Algorithm $\mathcal{A}_1$ and T obtained by Algorithm $\mathcal{A}_2$ are the same.

Notice that there well may be a choice between an exact Algorithm $\mathcal{A}_1$ running in exponential time and a heuristic Algorithm $\mathcal{A}_1$ running in polynomial-time, but that the same-inside relation allows us to distinguish between the two, while obscuring the immaterial differences between two exact Algorithms $\mathcal{A}_1$ producing the same output or two implementations of a heuristic. It is easy to see the same-inside relation is an equivalence. Subsequently:

Definition 2.37. A *structure* is a class of the same inside equivalence on the set of particular structures.

Ultimately, in Part III we introduce useful structures.

Chapter 3

Tree Decompositions

Many graph-related problems are easier to solve on trees than on an arbitrary graph. Intuitively, if a graph is in some sense close to being a tree, the problem could also be closer to being easy to solve, if one knew to "what tree" the graph is close to. This intuition has inspired a considerable amount of work on tree decompositions of graphs, which are pairs of a tree and a mapping from the tree to the original graph. Width parameters of graphs are then functions from graphs to integers, capturing the minimum of some property of any admissible tree decomposition. Informally speaking, for example, a graph has tree-width no more than k if and only if it can be covered by sub-graphs of size $k+1$ in a tree-like fashion. The generally applicable algorithms for integer programming [127, 302] based on such decompositions seem to have generated a great amount of interest, recently. We present an overview of what is known and an original probabilistic analysis.

3.1 The Many Definitions of Tree-width

There are now dozens of width parameters of graphs, most with many alternative definitions. The most influential width parameters, tree-width and branch-width, have been defined by Robertson and Seymour [363, 364], building upon the work of Seese [376] and others. This section focuses on tree-width.

There are a number of formal definitions of tree-width in the literature. The standard definition uses tree decompositions. Formally, a tree decomposition of $G = (V, E)$ is a pair $(\{X_i : i \in I\}, T)$, where $\{X_i : i \in I\}$ is a collection of subsets of V ("bags"), and $T = (I, F)$ is a tree such that:

- The bags cover the vertex-set, $\bigcup_{i \in I} X_i = V$
- Each edge is in at least one bag, $\forall \{u, w\} \in E, \exists i \in I : u, w \in X_i$

- Each vertex is included in bags forming a connected subtree of T, $\forall i, j, k \in I$: if j is on a ij-path in T, then $X_i \cap X_k \subseteq X_j$, or alternatively, $\forall v \in V$, $\{i \in I : v \in X_i\}$ is connected in T.

The width of a tree decomposition $(X_i : i \in I, T)$ is $\max_{i \in I} |X_i| - 1$. The tree-width of G is the minimum width over all tree decompositions of G.

In order to provide a better intuition, it is useful to reiterate a number of other definitions. The definition of Thomassen [392] start from a connected graph $G = G' \cup G''$ and its cut C, where G' and G'' are connected and $C = G' \cap G''$ is a complete subgraph. This decomposition into *simplical summands* can be performed recursively [389]. A connected graph in which every simplical summand is a complete graph K_w is called a K_w-*cockade*. The tree-width of a graph G, $\mathbf{tw}(G)$, is the least integer k such that G is a subgraph of a K_{w+1}-cockade [392].

The definition can also be phrased in terms of triangulations [69]. Graph $G = (V, E)$ is triangulated if each two non-consecutive vertices are adjacent in every cycle of length at least four. A graph $H = (V, E')$ is a *triangulation* of $G = (V, E)$ when H is triangulated and G is a subgraph of H. Tree-width of a graph G, $\mathbf{tw}(G)$, is one less than the minimum of clique sizes of H over all triangulations H of G. When one has a triangulation of G with maximum clique K_{w+1}, it is possible [69] to find a tree decomposition of G of width w in linear time.

Alternatively, tree-width can be introduced "bottom-up": For a $k > 0$, a *k-tree* is defined inductively as:

- the complete graph K_k is a k-tree
- any graph obtained from a k-tree G by adding a new vertex v and joining v to $k - 1$ vertices of some K_k subgraph of G

A *partial k-tree* is a subgraph of a k-tree. The tree-width of a graph G, $\mathbf{tw}(G)$, is the least integer k such that G is a partial k-tree.

3.2 BRANCH-WIDTH AND RELATED PARAMETERS

Branch-width [364] and related width parameters [332, 13, 85] can be presented in the following framework: Let f be a cut function of a graph G, and (T, δ) a decomposition tree of G. For every edge uv in T, $\{X_u, X_v\}$ denotes the 2-partitions of V induced by the leaf sets of the two subtrees we

get by removing uv from T. The f-width of (T, δ) is the maximum value of $f(X_u)$, taken over every edge uv of T. An optimal f-decomposition of G is a decomposition tree of G having minimum f-width. The f-width of G is the f-width of an optimal f-decomposition of G.

Outside of branch-width, it is easy to define rank-width [332] and boolean-width [13, 85] in this framework. In rank-width [332], function f is the cut-rank function:

$$f_{\mathbf{rw}} = \log_2 |\{Y \subseteq B : \exists X \subseteq A, Y = \triangle_{x \in X} N(x)\}| , \tag{3.1}$$

where neighborhood $N(x)$ are vertices adjacent to x and $\triangle$ denotes the set difference. This is the base-2 logarithm of the size of the row space over GF(2)-sums, which is the number of pairwise different vectors that are spanned by the rows of the $|A| \times |V \setminus A|$ submatrix of the adjacency matrix of G over GF(2), where $0 + 0 = 1 + 1 = 0$ and $0 + 1 = 1 + 0 = 1$. The corresponding discontiguous definition of taking a submatrix will be used throughout the paper. Boolean-width [13, 85] can then be defined similarly with

$$f_{\beta\mathbf{w}} = \log_2 |\{Y \subseteq B : \exists X \subseteq A, Y = \cup_{x \in X} N(x)\}| . \tag{3.2}$$

Informally, we take the logarithm of the number of distinct unions of the neighbourhoods of vertices. This, without much surprise, is the base-2 logarithm of the size of the row space of a binary matrix with boolean-sums ($1 + 1 = 1$). For standard definitions, please follow the references [364, 332, 13, 85].

3.3 Finding the Decomposition

3.3.1 The Problem and its Complexity

The problem of determining branch-width is hard:

Problem 3.1. Branch-decomposition: Given a $\{0, 1\}$ matrix A and an integer k, find a branch-decomposition of the column matroid of A whose branch-width is at most k, or report that no such branch-decomposition exists.

Proposition 3.2 ([380])**.** Branch-decomposition is $\mathcal{NP}$-Hard.

On the other hand, the problem is "easy", when the parameter k is not a part of the input:

Problem 3.3. TEST FOR BRANCH-WIDTH k: Given a $\{0,1\}$ matrix A, decide if there exists a branch-decomposition of the column matroid of A whose branch-width is at most k.

Proposition 3.4 ([241])**.** For any fixed k, TEST FOR BRANCH-WIDTH k is solvable in polynomial time.

Unfortunately, the proof of the result is rather non-constructive, in that the algorithm requires the list of excluded minors for branch-width k, which is finite, but unknown and known to be of superpolynomial cardinality.

3.3.2 An Integer Programming Formulation

Ulusal [397] formulated branch-width minimisation as:

$$\begin{aligned} & \min z \\ \text{s. t. } & \sum_{s \in S} x_s = |E| - 3 \\ & (|E| - 4)x_s - \sum_{s' \in S, s \odot s'} x_{s'} \leq 0 \quad \forall s \in S \\ & c_s x_s \leq z \quad \forall s \in S \end{aligned} \tag{3.3}$$

where there is x_S for each pair $S = (A, B)$ of complementary subsets of E ("separation"), with value 1 if S is used in the branch decomposition and 0 otherwise, and c is an $|E|$-vector of constants ("degree of separation"). $S \odot S'$ denotes S and S are laminar. Notice that this formulation closely resembles integer linear programming.

3.3.3 Heuristics

As far as we are aware, there are published only a few accounts of computational experience with heuristics for branch-width decomposition:

- "Caterpillar" heuristics [302]: Take the n columns of a constraint matrix in an arbitrary order and assign them to the leaves of a tree on $2n-2$ vertices and $2n-1$ edges, which is known as a "caterpillar".
- Greedy heuristics [72]: There are a number of heuristics, which sort columns in the order of degree of a corresponding graph or similar.

A computational comparison of greedy fill-in (GFI), greedy fill-in + triangulation minimization (GFI+TM), and greedy degree (GD) for tree-width is provided by Bodlaender and Koster [72].

- Local search heuristics [401]: Neighbourhood search for tree-width minimisation based on the search for triangulations of a graph has been described. It is not clear, however, whether there are any implementations and what is their performance.
- Spectral methods [121]: Reorder the matrix by the values in the Fiedler eigenvector [155] of a Laplacian of the corresponding graph. Cook and Seymour [121] report their positive experience with such methods for branch-width decompositions of planar graphs.
- Node separator approaches: Arguably, one could formulate a number of approaches based on finding "separators", small subsets of vertices, whose removal leaves the (sub)graph disconnected. There are now very well-developed heuristics (e.g. MeTiS and PaToH, [260, 97, 98]) for the partitioning, used in so-called "nested dissection" throughout numerical linear algebra. Alternatively, one could use min-cut computations, which have been used in unpublished work by Hliněný et al. and Margulies et al. [302].

See also Subsection 5.2.3 for related work on heuristics for block decomposition. Indeed, together with sparse matrix factorization, both problems are closely related to the problem of finding vertex separators. The connection between tree-width and the height of an elimination tree [285] has been made precise previously [71]. Using the known inequalities between tree-width and branch-width, one can easily extend this connection to branch-width.

3.4 Exploiting the Decomposition

The only approach to exploiting tree decompositions in integer programming known to be implemented, so far, is the algorithm of Cunningham and Geelen [127], which solves:

Problem 3.5 ([127])**.** Equality-constrained Integer Programming using a Branch-width k Decomposition of a Non-negative Matrix: Given $m \times n$ integer non-negative (!) matrix A, m-vector b of integers, n-vector c of integers, and branch-decomposition T of column matroid $M(A)$ of width k, find $x \in \mathbb{N}^n$ such that $Ax = b$ and $c^T x$ is minimised or report that no such x exists.

Algorithm 1 `Cunningham-Geelen`(A, b, c, T, k)
in the presentation of Margulies et al. [302]

Input: Non-negative integral $A \in \mathbb{Z}_+^{m \times n}$ and its branch-decomposition $T = (V, E), \nu$ of width k, non-negative integral $b \in \mathbb{Z}^m$ and integral $c \in \mathbb{Z}^n$
Output: Vector $x \in \mathbb{Z}^n$ such that $c^T x$ is minimised
for all $X \subseteq \{1, 2, \ldots, n\}$ **do**
  if $|X| = 1$ **then**
    Calculate $B(X)$
  else
    Partition X into (X_1, X_2)
    $B(X) = \emptyset$
    for all $b_1 \in B(X_1)$ and $b_2 \in B(X_2)$ **do**
      $b' = b_1 + b_2$
      if $b' \leq b$ and $b' \in \text{span}([Ab][X])$ **then**
        $B(X) = B(X) \cup b'$
      end if
    end for
  end if
end for
for all $b \in B(\{1, 2, \ldots, n\})$ **do**
  Find x corresponding to b
  Find the minimum of $c^T x$
end for
Return optimal x

`Cunningham-Geelen` uses dynamic programming. The algorithm builds up $B(\{1, 2, \ldots, n\})$ recursively: $B(X)$ from $B(X_1), B(X_2)$ for subproblems restricted to a partition of $X_1, X_2 \subset X, X_1 \cup X_2 = X, X_1 \cap X_2 = \emptyset$. Subsequently, it obtains x for each $b \in B(\{1, 2, \ldots, n\})$, evaluates the objective function at x, and picks the minimum. See Algorithm 1, where $[Ab][X]$ denotes matrix obtained by concatenation of A, b restricted to columns of X. Overall:

Proposition 3.6 (Cunningham and Geelen [127])**.** `Cunningham-Geelen` solves EQUALITY-CONSTRAINED INTEGER PROGRAM USING A BRANCH-WIDTH k DECOMPOSITION in pseudo-polynomial time $O((\max(b) + 1)^{2k} mn + m^2 n)$.

Margulies et al. [302] has only recently started working on the first known implementation. They report positive results on infeasible instances with

branch-width of the constraint matrix at most six, problems with square constraint matrix of branch-width at most six and small numbers of feasible solutions and trivial compact linear programming relaxations, and certain related classes of problems.

Further means of exploiting low width measures could be based on the work of Bienstock and Ozbay [53]. They show, broadly speaking, that for a graph of tree-width k, k applications of Sherali-Adams operator suffice to obtain the convex hull of feasible solutions for linear relaxations of various specific graph problems. It can be conjectured that this is the case more generally. Notice that this approach requires solving very large linear programs, but it does not require the tree decomposition.

3.5 An Analysis

Considering that it is not clear what integer programs do have low width measures, it seems important to ask the following questions:

- Does a random binary matrix have low branch-width?
- What is the tree-width, branch-width, and rank-width of a random binary matrix?
- What classes of binary matrices do have low branch-width?

We set out to study width parameters via the rank of certain submatrices of random matrices over GF(2). We denote $M(m,n,p)$ to be the family of probability spaces over $m \times n, m \leq n$ matrices over GF(2), where each element of a matrix is chosen independently to be 1 with probability p and 0 with probability $1-p$. Initially, we also use a simple random model $M(m,n,\frac{1}{2})$, denoted $M(m,n)$ for $n \in 3,6,9,\ldots$. For event A, we denote the probability that A happens by $\mathbb{P}(A)$. A happens asymptotically almost surely in a family of probability spaces indexed with n, if $\mathbb{P}(A)$ goes to 1 as n goes to the large limit.

First, we state a proposition derived from a more general theorem of Blömer, Karp, and Welzl [67], which is reminiscent of Shannon's switching game [381]:

Proposition 3.7 (Blömer et al. [67])**.** The probability that an $m \times n, m \leq n$ matrix $M_{m,n}$ drawn randomly from $M(m,n)$ has rank less than $m-d$ is $2^{-\Omega((n-m+d)(m+d))}$.

We use this to derive a simple lemma:

Lemma 3.8. Asymptotically almost surely, the minimum rank of $\frac{n}{3} \times \frac{2n}{3}$ submatrices of $M_{n,n}$ drawn randomly from $M(n,n)$ is bounded from below by $\Omega(n)$.

Proof. Let us denote the minimum rank among $\frac{n}{3} \times \frac{2n}{3}$ submatrices $S_{\frac{n}{3},\frac{2n}{3}}$ in an $n \times n$ matrix $M_{n,n}$ over GF(2) drawn from $M(n,n)$ by μ and let us study the probability of μ being greater than an arbitrary $\frac{n}{6}$. Using the union bound (in 3.5), the $\binom{n}{\frac{n}{3}} \approx O(2^{2n})$ bound on the number of submatrices of interest (in 3.6), and Proposition 3.7 (in 3.7):

$$\begin{aligned}
\mathbb{P}(\mu \leq \frac{n}{6}) &= \mathbb{P}(\text{rank}(N) \leq \frac{n}{6} \quad \forall S_{\frac{n}{3},\frac{2n}{3}} \in M_{n,n}) && (3.4)\\
&\leq \sum_{S_{\frac{n}{3},\frac{2n}{3}} \in M_{n,n}} \mathbb{P}(\text{rank}(S_{\frac{n}{3},\frac{2n}{3}}) \leq \frac{n}{6}) && (3.5)\\
&= \binom{n}{\frac{n}{3}} \mathbb{P}(\text{rank}(S_{\frac{n}{3},\frac{2n}{3}}) \leq \frac{n}{6}) && (3.6)\\
&\approx O(2^{2n}2^{-n^2}) && (3.7)
\end{aligned}$$

Clearly,

$$\lim_{n\to\infty} 2^{2n}2^{-n^2} = 0. \tag{3.8}$$

□

Now, we can state the main result using a simple model $G(n, \frac{1}{2})$ of random graphs of n vertices, where each edge appears independently with probability $\frac{1}{2}$.

Theorem 3.9. Asymptotically almost surely, the rank-width of a graph drawn randomly from $G(n, \frac{1}{2})$ is bounded from below by $\Omega(n)$.

Proof. The minimax theorem of Robertson and Seymour [364] linking branch-width and tangles, translated to rank-width by Oum [333], implies there exists an edge in any decomposition tree, which corresponds to a partition (V_1, V_2) of n vertices of G, such that $\frac{n}{2} \geq |V_1| \geq \frac{n}{3}$ and $\frac{n}{2} \leq |V_2| \leq \frac{2n}{3}$. But then the rank of the $\frac{n}{3} \times \frac{2n}{3}$ submatrix of the minimum rank in the adjacency matrix of graph G, given by Lemma 3.8, is a lower bound on the value of the rank-width of G. Notice that we need not consider skew-symmetric matrices in Lemma 3.8, as the sub-matrix is not necessarily symmetric. □

This theorem has been the first announced[1] probabilistic result on rankwidth, and has sparked further research [280].

Given the trivial upper bound of n on rank-width of a graph on n vertices, it is easy to see this lower bound is tight:

Corollary 3.11. Asymptotically almost surely, the rank-width of a graph drawn randomly from $G(n, \frac{1}{2})$ is $\Theta(n)$.

Notice the following inequalities:

Proposition 3.12 ([240])**.** $\mathbf{nlcw}(G) \geq \mathbf{cwd}(G) \geq \mathbf{rw}(G)$
$\mathbf{tw}(G) + 1 \geq \mathbf{bw}(G) \geq \mathbf{rw}(G)$

Using these inequalities between the values of the parameters, we can state the following:

Corollary 3.13. Asymptotically almost surely, the tree-width of a graph drawn randomly from $G(n, \frac{1}{2})$ is $\Theta(n)$.

Corollary 3.14. Asymptotically almost surely, the branch-width of a graph drawn randomly from $G(n, \frac{1}{2})$ is $\Theta(n)$.

Corollary 3.15. Asymptotically almost surely, the clique-width of a graph drawn randomly from $G(n, \frac{1}{2})$ is $\Theta(n)$.

Corollary 3.16. Asymptotically almost surely, the NLC-width of a graph drawn randomly from $G(n, \frac{1}{2})$ is $\Theta(n)$.

Consider the following proposition:

[1] A draft describing this work was posted in arxive in August 2009 and presented at the fourth workshop on Graph Classes, Optimization, and Width Parameters in Bergen, Norway, in October 2009. Lee et al. [280] presented their results at the same workshop, but posted their draft to arxive only in January 2010. The final version of their main theorem reads:

Proposition 3.10 (Lee et al. [280])**.** $G(n, p)$ is a model of random graphs on n vertices, where each edge appears independently with probability p. For a random graph $G_{n,p}$ drawn randomly from $G(n, p)$, the following holds asymptotically almost surely:

1. if $p \in (0, 1)$ is a constant, then $\mathbf{rw}(G_{n,p}) = \lceil n/3 \rceil - O(1)$,
2. if $1/n \ll p \leq \frac{1}{2}$, then $\mathbf{rw}(G_{n,p}) = \lceil n/3 \rceil - o(n)$,
3. if $p = c/n$ and $c > 1$, then $\mathbf{rw}(G_{n,p}) \geq rn$ for some $r = r(c)$, and
4. if $p \leq c/n$ and $c < 1$, then $\mathbf{rw}(G_{n,p}) \leq 2$.

Proposition 3.17 (Bodlaender [68])**.** If the tree-width of $G = (V, E)$ is at most k, then $|E| \leq k|V| - \frac{1}{2}k(k+1)$.

The corollary for tree-width should not be very surprising. Indeed, stronger probabilistic results on tree-width have been provided by Bodlaender and Kloks [266]. Independently, tree-width of random NK landscapes [177] and random graphs [175, 176] have been studied by Gao. For NLC-width and clique-width of random graphs, Johansson [250] has also proven stronger results. Nevertheless, Corollary 3.14 has been the first announced probabilistic result on branch-width of random graphs.

Chapter 4

Unimodularity and Generalisations

Everyone wants a paper on total unimodularity.

H. Paul Williams [412]

In the previous chapter, we have shown that exploiting branch-width decompositions is related to exploiting certain "good submatrices" of the constraint matrix. The search for "good matrices", for which there is a polynomial time algorithm for integer programming, is much older, though. This direction of research has yielded hundreds of papers and solvers for many important specific problems, albeit rather less in terms of well-performing general-purpose solvers. As in the previous chapter, we present an overview of known results and some simple counting arguments.

4.1 Good Matrices

The most important classes of "good matrices" can be outlined as follows:

Proposition 4.1 (Totally unimodular matrices)**.** Let A be an $m \times n$ $0, \pm 1$ matrix. The following are equivalent:

- each square submatrix of A has determinant 0, 1, or -1
- no square submatrix of A has determinant 2 or -2 [93]
- the determinant of each non-singular $m \times m$ submatrix of full row rank matrix $[IA]$ is ± 1 [404]
- each non-singular submatrix of A has a row with an odd number of non-zero components [93]

- each non-singular submatrix B of A has the greatest common devisor of entries of yB 1 for each $\{0, \pm 1\}$ vector y [100]
- the sum of entries in any square submatrix of A with even row and column sums is divisible by four [93]
- each collection of columns of A can be split into two parts so that the sums of the columns in one part minus the sum of the columns in the other part is a vector with entries only in 0, 1, and -1 [188]
- the polyhedron $\{x : Ax \leq b, x \geq 0\}$ has only integral vertices for any integral vector b [242]
- the polyhedron $\{x : a \leq Ax \leq b, c \leq x \leq d\}$ has only integral vertices for any integral vectors a, b, c, d [374]
- the value $\max\{cx : x \in Ax \leq b\} = \min\{yb : yA = c, y \geq 0\}$ is achieved by integer vectors x, y, if finite, for any integral vectors b, c [242].

For instance, the incidence matrix of a bipartite graph is totally unimodular, i.e. there is a bipartite graph G on m vertices with n edges such that $Av, e = 1$ if and only if edge e touches vector v [374]. Further examples of totally unimodular matrices are network matrices:

Definition 4.2 (Tutte [374])**.** Let M be an $m \times n$ $\{0, \pm 1\}$ matrix. M is a network matrix, if and only if there exists a directed graph $D = (V, A), |A| = n$ and a directed tree $T = (V, A_T)$, $|A_T| = m$ and for $(u, v) \in A, a \in A_T$:

$$M_{a,(u,v)} = \begin{cases} +1 & \text{if } (u,v)\text{-path in } T \text{ passes through } a \text{ forwards} \\ -1 & \text{if } (u,v)\text{-path in } T \text{ passes through } a \text{ backwards} \\ 0 & \text{if } (u,v)\text{-path in } T \text{ does not pass through } a. \end{cases}$$

Ultimately, Seymour [378] has formulated necessary and sufficient conditions for a matrix to be totally unimodular. For its clarity, we give a little known formulation of the characterisation [379] in terms of unimodular hypergraphs, which is also due to Seymour [379]. Notice that a hypergraph on vertices $V, |V| = n$ with edges $E, |E| = m$ is given by an arbitrary binary $m \times n$ matrix $M = (m_{i,j})$. M is denoted the incidence matrix, and $m_{e,v} = 1$ if vertex v is in edge e and 0 otherwise. A hypergraph with a totally unimodular incidence matrix is denoted as unimodular. For a directed tree T with edges $e_1, \ldots, e_n$ and directed paths $P_1, \ldots, P_m$, a hypergraph with vertex set $e_1, \ldots, e_n$ and edges $P_1, \ldots, P_m$ is called the tree hypergraph. Clearly, there are bijections between tree hypergraphs and network matrices and unimodular hypergraphs and totally unimodular matrices.

Proposition 4.3 (Seymour [379])**.** When H, H' are unimodular, constructions (1–6) listed below yield unimodular hypergraphs. Any unimodular H can be constructed by means of the constructions (1–6) listed below, starting from tree hypergraphs and their transposes and copies of H_5 [395, 58]. For hypergraphs $H = (V, E), H' = (V', E'), E = (E_i \text{ s. t. } i \in I), E' = (E_i \text{ s. t. } i \in I'), V \cap V' = \emptyset, I \cap I' = \emptyset$, the constructions are:

1. Construct hypergraph with vertices $V \cup V'$ and edges $E_i, i \in I \cup I'$.
2. For $v \in V, j' \in I'$, construct hypergraph with vertices $V \setminus \{v\} \cup V'$ and edges
   - $E_i, i \in I, v \notin E_i$
   - $(E_i \setminus \{v\}) \cup E_j, i \in I, v \in E_i$
   - $E_i, i \in I', i \neq j'$
3. For $u, v \in V, j \in I, E_j = \{u, v\}$, and for $j', k', l' \in I'$ such that $E_{k'} \cap E_{l'} = \emptyset$, $E_{k'} \cup E_{l'} = E_{j'}$, construct hypergraph with vertices $V \setminus \{u, v\} \cup V'$ and edges
   - $E_i, i \in I, u, v \notin E_i$
   - $(E_i \setminus \{u\}) \cup E_{k'}, i \in I, u \in E_i, v \notin E_i$
   - $(E_i \setminus \{v\}) \cup E_{l'}, i \in I, u \notin E_i, v \in E_i$
   - $(E_i \setminus \{u, v\}) \cup E_{j'}, i \in I \setminus \{j\}, u, v \in E_i$
   - $E_i, i \in I' \setminus \{j', k', l'\}$
4. As above (3), except a transpose of the result for H^T and H'^T
5. For $v \in V, j, k \in I, v \notin E_j, E_k = E_j \cup \{v\}$, and for the analogous $v' \in V', j', k' \in I'$, construct hypergraph with vertices $(V \setminus \{v\}) \cup (V' \setminus \{v'\})$ and edges
   - $E_i, i \in I \setminus \{j\}, v \notin E_i$
   - $E_i, i \in I \setminus \{j\}, v' \notin E_i$
   - $(E_i \setminus \{v\}) \cup E_j, i \in I \setminus \{k\}, v \in E_i$
   - $(E_i \setminus \{v'\}) \cup E_j, i \in I' \setminus \{k'\}, v' \in E_i$
6. For $u, v \in V, j, k, l \in I, E_l = \{u, v\}, u \notin E_j, E_k = E_j \cup \{u\}$, and no edge of H contains u except for E_k, E_l. For $u', v' \in V', j', k', l' \in I'$ defined analogously, construct hypergraph with vertices $(V \setminus \{u, v\}) \cup (V' \setminus \{u', v'\})$ and edges

- $E_i, i \in I \setminus \{j,k,l\}, v \notin E_i$
- $E_i, i \in I' \setminus \{j',k',l'\}, v' \notin E_i$
- $(E_i \setminus \{v\}) \cup E_j, i \in I \setminus \{l\}, v \in E_i$
- $(E_i \setminus \{v'\}) \cup E_j, i \in I' \setminus \{l'\}, v' \in E_i$.

As will be shown in Section 4.4, totally unimodular matrices are rather rare. One hence studies generalisations of total dual unimodularity, which lead to broader classes of matrices. The natural generalisation are matrices containing a large totally unimodular submatrix.

4.2 FINDING GOOD SUBMATRICES

4.2.1 The Problem and its Complexity

Note the following:

Problem 4.4. TEST FOR INTEGRALITY: Let A be a rational matrix and b a compatible rational vector. Decide if the polyhedron determined by $Ax \geq b$ has only integral vertices.

Proposition 4.5 ([374])**.** TEST FOR INTEGRALITY is strongly $\mathcal{NP}$-Complete.

Despite this proposition, one can show:

Problem 4.6. TEST FOR TOTAL UNIMODULARITY: Let A be a $0, \pm 1$ matrix. Decide if $Ax \geq b$, $x \geq 0$ has only integral vertices for any integral b.

Proposition 4.7 ([374])**.** TEST FOR TOTAL UNIMODULARITY is in $\mathcal{P}$.

The proof relies on the Seymour's decomposition of regular matroids [378] (see Proposition 4.3) and it is not clear whether there are any implementations. Similarly:

Problem 4.8. NETWORK MATRIX RECOGNITION: Given a $\{0, \pm 1\}$ matrix A, decide whether elementary row operations and nonzero variable-scaling can transform A into a network matrix.

Proposition 4.9 (Bixby and Cunningham [62])**.** NETWORK MATRIX RECOGNITION is in $\mathcal{P}$.

This result is based on Tutte's algorithm [396] for testing whether a given binary matroid is graphic, but an efficient implementation has been demonstrated [62]. Although one can test a matrix for being either a network matrix or totally unimodular, a large totally unimodular submatrix is hard to find.

Bartholdi [37] shows that the complexity of testing the existence of submatrices of certain properties can be stated quite generally: The matrix property $q(A)$ is hereditary, if $q(A)$ implies $q(A')$ for all submatrices $A' \subset A$. A property is non-trivial, if it holds for an infinite number of matrices and fails for an infinite number of matrices. A quality is a non-trivial property that is hereditary and holds for permutation matrices. Qualities include, for example, total unimodularity and permutability to a network matrix. [1] For any quality:

Problem 4.10. Test for Submatrix with a Quality q: Given $0,1$ matrix A and integers k_1, k_2, decide if A contains a $k_1 \times k_2$ submatrix with property q.

Proposition 4.11 (Bartholdi [37])**.** Test for Submatrix with a Quality q is $\mathcal{NP}$-Complete.

The complexity of detecting a totally unimodular submatrix, in particular, can be shown from the correspondence between independent sets and network submatrices.

Proposition 4.12 (Harary [231], Gülpinar [217])**.** For $\{-1,0,1\}$ matrix A, the size of the network submatrix is $\max_{S \subseteq V}\{\alpha((G(A)^S)^-)\}$, where vertices of $G = (V, E^-, E^+)$ correspond to rows of A, there is a negative edge $(i,j) \in E^-$ if and only if $a_{ik} = a_{jk} \neq 0$ for some column k, there is a positive edge $(i,j) \in E^+$ if and only if $a_{ik} = -a_{jk} \neq 0$ for some column k, G^S is the graph obtained from G by changing signs of each edge in the cut between S and $V \setminus S$, and α denotes the size of the largest independent set.

Based on this correspondence, Gülpinar [216, 217] have proposed graph-theoretical algorithms for extracting a totally unimodular submatrix. There is a growing body of related work, drawing connections between bi-directed graphs and certain matrices [24, 353, 270, 321, 322], for which there are guarantees of half-integral optimal solutions to linear programs $Ax \leq b$ [24].

[1] Without introducing the definitions, let us state that qualities include many other properties, which could be exploited: balanced, perfect, permutable to consecutive or circular ones. Similar results hold also for column submatrices, row submatrices, and principal submatrices.

4.2.2 An Integer Linear Programming Formulation

Naturally, there is an integer programming formulation of extracting the largest network sub-matrix [156]. There is binary variable y_i for each vertex $i \in V$ and y_i is one if and only if i belongs to the network sub-matrix. When one denotes the set of cycles in G with odd numbers of negative edges and no parallel edges S, the formulation reads:

$$\begin{aligned} \max & \sum_{i \in V} y_i \\ \text{s. t.} \quad & \sum_{i \in C} y_i \leq |C| - 1 && \forall \text{ odd negative cycle } C \in S && (4.1) \\ & y_i + y_j \leq 1 && \forall (i,j) \in E^- \cap E^+ && (4.2) \\ & y_i \in \{0,1\} \end{aligned}$$

Notice that set S can be exponentially larger than the set of vertices and, hence, the performance of any implementation depends on the quality of the separation routine for the corresponding constraint (4.1).

4.2.3 The Heuristics

There is also a considerable body of work on heuristics for finding network submatrices, which are totally unimodular submatrices. Typically, the heuristics focus on the subset of rows, such that there is at most one coefficient 1 and at most one coefficient -1. Outside of pre-processing and heuristic scaling, which tries to convert columns with 2 non-zero coefficients to ± 1 rows, the heuristic are based on one of:

- Row-scanning deletion [198, 80, 82]: Pick all rows. For each row in turn, decide if together with the remaining rows, it forms a network submatrix. Is so, stop. If not, discard it. Optionally, a second-pass [83] may attempt to reinsert previously discarded rows.
- Column-scanning deletion [66, 65, 218]: Pick all columns. For each column in turn, decide if together with the remaining columns, it forms a network submatrix. Is so, stop. If not, discard it.
- Row-scanning addition [66, 218]: Start with an empty set of rows. For

each row in turn, decide if it can be added so that the chosen rows still form a network submatrix. Is so, add it. If not, continue.

- Translation to graph theory [219, 217]: Based on Proposition 4.12, one can use a wide variety of heuristics for finding large independent sets.

There are a number of implementation issues in exploiting sparsity, determining a suitable order of processing, as well as the application of scaling of a column. Of particular importance is scaling by -1, which is denoted "reflection" in the literature. A good overview[2] with empirical comparison is provided by Bixby and Fourer [66, 65].

4.3 Exploiting Good Submatrices

The obvious use of a network submatrix is to construct a basis for simplex-based linear programming using a network flow algorithm [219], or to replace the pricing routine [299]. A comprehensive approach to the use of network submatrices in branch-and-bound solvers for integer programming with 0/1 constraint matrices has been developed in a series of papers [17, 16] by Agha et al.

4.4 An Analysis

Totally unimodular matrices are clearly a subset of $\{0, \pm 1\}$ matrices, but need not be sparse. Totally unimodular matrices need not have full rank, and indeed, most do not, in simple random models. Although there are many known special cases of unimodular matrices and any particular matrix can be tested for total unimodularity using the algorithm of Seymour [374], little seems to be known about the limits of performance of general-purpose solvers based on the structure, such as those proposed [17, 16] by Agha et al. In particular, it seems to be interesting to ask the following question:

- What is the probability that a random $\{0, \pm 1\}$ matrix is totally unimodular?
- What is the largest totally unimodular submatrix in a random $\{0, \pm 1\}$ matrix?

[2] Another good source may be the dissertation of Ahn [14], which I have not been able to obtain, though.

Notice that constructions (C1–C6) of Proposition 4.3 suggest a recursion for the number of $m \times n, \{-1, 0, 1\}$ matrices, which are totally unimodular and distinct up to the permutations of rows and columns. It seems difficult to obtain a closed-form solution, though.

We make only a small step in this direction by bounding the number of totally unimodular matrices from below:

Lemma 4.13. The number $U(n)$ of distinct unimodular matrices with n columns, up to permutations of rows and columns, satisfies:

$$U(n) \geq \sum_{i=0}^{n-1} n^{i-1} \sum_{j=1}^{i} (-1)^{i-j} \frac{j^{n-2}}{(j-1)!(i-j)!} \tag{4.3}$$

Proof. The number $U(n)$ of distinct unimodular matrices is clearly bounded from below by the number of distinct network matrices. The number of distinct network matrices corresponds to the number of distinct hypertrees $T(n)$. Warme [407] has shown the exact number of distinct hypertrees $T(n)$ is

$$T(n) = \sum_{i=0}^{n-1} \left\{ {n-1 \atop i} \right\} n^{i-1}, \tag{4.4}$$

where $\left\{ {n \atop k} \right\}$ is the Stirling number of the second kind. From the standard formula for the Stirling number of the second kind:

$$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n = \sum_{j=1}^{k} (-1)^{k-j} \frac{j^{n-1}}{(j-1)!(k-j)!}, \tag{4.5}$$

we have the result. □

Unfortunately, Lemma 4.13 is not sufficient to prove:

Conjecture 4.14. Asymptotically almost surely, the probability that an $m \times n$ matrix, whose entries are independently equiprobably drawn from the set $\{0, \pm 1\}$, is unimodular is 0.

Still, it seems likely, considering the number of network matrices $T(n)$ is much less than the number of matrices on three elements.

Chapter 5

The Bordered Block-Diagonal Form

Perhaps the most promising "known structure" is related to bordered block-diagonal matrices. The performance of general-purpose solvers based on the structure is not quite clear yet, but the preliminary results of Bergner et al. [45] seem to be very encouraging. As in the previous two chapters, we present an overview of known results and a probabilistic analysis.

5.1 Dantzig-Wolfe Decomposition and Multi-Commodity Flows

Dantzig-Wolfe decomposition [135, 174] is a conceptually simple technique for decomposing a problem. Let us see the constraint matrix of the problem to decompose as partitioned into rows corresponding to "easy" constraints E and "hard" constraints H. Clearly, the corresponding polyhedron $P = \{x \in \mathbb{R}^n \mid Ex \leq e, Hx \leq h\}$, is an intersection of $P_E = \{x \in \mathbb{R}^n \mid Ex \leq e\}$ with $P_H = \{x \in \mathbb{R}^n \mid Hx \leq h\}$. Let us consider the convex hull of integer points inside P_E, and particularly set V of vertices of the convex hull of integer points inside P_E. It is well known that each point in the convex hull of P_E can be written as a convex combination of V. Considering that polyhedron P is bounded and there is a vector of non-negative coefficients $\lambda \in \mathbb{R}^{|V|}$, one obtains:

$$P = P_{DW} = \{x \in \mathbb{R}^n \mid Hx \leq h, x = \sum_{v \in V} \lambda_v v, \sum_{v \in V} \lambda_v = 1\}.$$

Notice that one may obtain a stronger bound by optimising over P_{DW} than by optimising over P only if the convex hull of integer points inside P_E is not equal to P_E.

The constraint matrix is often reordered into a bordered or double-bordered diagonal form, and the block-diagonal part is considered to form the "easy"

constraints, where the well-known special forms are:

Block-diagonal Bordered block-diag. Double-bordered block-diag.

$$\begin{bmatrix} D^1 & & & \\ & D^2 & & \\ & & \ddots & \\ & & & D^k \end{bmatrix} \quad \begin{bmatrix} D^1 & & & \\ & D^2 & & \\ & & \ddots & \\ & & & D^k \\ G^1 & G^2 & \cdots & G^k \end{bmatrix} \quad \begin{bmatrix} D^1 & & & & F^1 \\ & D^2 & & & F^2 \\ & & \ddots & & \vdots \\ & & & D^k & F^k \\ G^1 & G^2 & \cdots & G^k & H \end{bmatrix}$$

Furthermore, one often considers focuses on matrices, where the blocks on the diagonal are unimodular. This way, it is not necessary to solve an integer program to obtain each vertex of the convex set, but also the bound obtained by optimising over P_{DW} cannot improve over optimisation over P.

Let us consider the following simple example:

Problem 5.1 ([151]). TWO-COMMODITY INTEGRAL FLOW: Given directed finite graph $G = (V, E)$ with vertices $V = 1, 2, \ldots, n$ and edges $E \subset \{(u, v)$ s. t. $1 \leq u \neq v \leq n\}$, capacity function $c : E \to \mathbb{Z}^+$ sources $s_1, s_2 \in V$ (not necessarily distinct), terminals $t_1, t_2 \in V$ (not necessarily distinct), requirements $R_1, R_2 \in \mathbb{Z}^+$, decide if there exist flow functions $f_1, f_2 : E \to \mathbb{Z}^+$ such that (1) flow is bounded by capacity: $\forall u \to v \in E, f_1(u \to v) \geq 0, f_2(u \to v) \geq 0, f_1(u \to v) + f_2(u \to v) \leq c(u \to v)$ (2) flow is conserved: f$\forall i \in \{1, 2\} \forall v \in V \setminus \{s_i, t_i\}, \sum_{u \to v \in E} f_i(u \to v) = \sum_{v \to w \in E} f_i(v \to w)$ (3) flow is bounded from below: $\forall i \in \{1, 2\} \sum_{s_i \to v \in E} f_i(s_i \to v) - \sum_{v \to s_i \in E} f_i(v \to s_i) \geq R_i$.

Proposition 5.2 (Even et al. [151]). TWO-COMMODITY INTEGRAL FLOW is $\mathcal{NP}$-Complete even for constant capacity function.

Notice the structure of the constraint matrix corresponding to TWO-COMMODITY INTEGRAL FLOW. For each commodity, there is a block on the diagonal. There are only $|E|$ constraints linking all commodities, so as to enforce the capacity-bound.

5.2 REORDERING IN A BORDERED BLOCK-DIAGONAL FORM

Can one, however, reorder a general matrix A into the (bordered) block-diagonal form? How difficult is it? In Section 5.4, we show there are indeed cases when this is possible, but there may well be a considerable border.

5.2.1 The Problem and its Complexity

In this section, we review on the complexity of the problem:

Problem 5.3 ([78]). Bordered Block-Diagonal Decomposition: Given an integral matrix A and integers β, κ, find the least integer b such that A can be reordered by row- and column-permutations to a matrix, where there are non-zero entries only in β blocks of capacity at most κ each on the diagonal and in b rows of the border.

In order to reason about the complexity of the problem, let us introduce:

Problem 5.4 ([84]). Disconnecting graphs by removing vertices: Given an undirected graph $G = (V, E)$ with vertices $V = 1, 2, \ldots, n$ and edges $E \subset \{(u, v) \text{ s. t. } 1 \leq u < v \leq n\}$, find the least integer $1 \leq r \leq |V|$, such that there exists a partition $V = V_1 \cup V_2 \cup R$ where there are no edges $(v_1, v_2) \in E, v_1 \in V_1, v_2 \in V_2$ and $|R| = r$.

Proposition 5.5 (Bui and Jones [84]). Disconnecting graphs by removing vertices is $\mathcal{NP}$-Hard.

Marx [305] has studied variants of the problem and shown which variants are fixed-parameter tractable.

Given the correspondence of the special case of $\beta = 2$ of Bordered Block-Diagonal Decomposition to Disconnecting graphs by removing vertices, it is easy[1] to see:

Lemma 5.6. Bordered Block-Diagonal Decomposition is $\mathcal{NP}$-Hard.

5.2.2 An Integer Linear Programming Formulation

Naturally, there is also an integer programming formulation [77, 78], with binary variable $x_{i,b}$ one if row i belongs to block b:

[1] The author is not aware of this result being known previously, but due to its simplicity, acknowledges it may well have been.

$$\max \sum_{i=1}^{m} \sum_{b=1}^{\beta} x_{i,b}$$

$$\text{s. t.} \quad \sum_{b=1}^{\beta} x_{i,b} \leq 1 \qquad \forall \text{ rows } i = 1, \ldots, m \qquad (5.1)$$

$$\sum_{i=1}^{m} x_{i,b} \leq k \qquad \forall \text{ blocks } b = 1, \ldots, \beta \qquad (5.2)$$

$$x_{i,b} + x_{j,b'} \leq 1 \qquad \forall 1 \leq b \neq b \leq \beta, 1 \leq i \neq j \leq m \qquad (5.3)$$

$$A_{ik} \neq 0 \neq A_{jk} \text{ for some } 1 \leq k \leq n$$

$$x_{i,b} \in \{0, 1\}$$

Notice that the formulation is compact, i.e. the number of both variables and constraints in the partitioning instance is polynomial in the number of variables of the original instance to partition. Cuts have been described by [77, 78, 330].

5.2.3 The Heuristics

There are a number of heuristics:

- Maximum transversal [408, 233]: A number of non-zeros is chosen so that no two non-zeros are chosen in any single row or column [141]. The square submatrix corresponding to the chosen rows and columns is reordered [386] so that the chosen non-zeros appear on the main diagonal and the matrix is blocked. Rows and columns not chosen initially are merged in where appropriate.

- Node separator approaches [184, 366, 78, 29]: Consider the corresponding graph and partition it recursively by finding "separators", small subsets of vertices, whose removal leaves the (sub)graph disconnected. There are now very well performing heuristics (MeTiS and PaToH) for the partitioning, used in so called "nested dissection" throughout numerical linear algebra.

- Packing and covering approaches [77, 78]: Reformulate the problem as set packing or set covering and used known heuristics or exact column generation methods.

- Spectral methods [28]: Reoder the matrix by the values in the Fiedler eigenvector [155] of a Laplacian of the corresponding graph.

Rothberg and Hendrickson [366] and the recent dissertation of Tebboth [390] detail their computational experience.

5.3 Exploiting the Bordered Block-Diagonal Form

There is a large body of work on exploiting the bordered block-diagonal in linear algebra for linear programming solvers. Notably, it is essential in parallelisation (one block per core). See [366] for an overview. In an integer programming solver, the situation is more involved. One can use the structure to generate cuts for the multiple knapsack problem [154] or, equivalently, the generalised assignment problem [207] or, equivalently, a certain packing problem $\sum_{j=1}^{n} a_{ij}x_{ij} \leq b_i$, $i \in M = \{1, \cdots, m\}$ with special order set constraints $\sum_{i=1}^{m} x_{ij} \leq 1$, $j \in N = \{1, \cdots, n\}$, where $x_{ij} \in \{0, 1\}$, $i \in M$, $j \in N$ [207]. There are cuts based on both one and more knapsack constraints. For more than one knapsack constraint, cycle, generalised cover, and $(1, k)$-configuration inequalities, seem to be quite powerful, but the do not seem to be any recent computational studies. Based on a long history of work on exploiting network submatrices in multi-commodity flow problems [307, 308, 299], Achterberg and Raack [12] have proposed an separation algorithm for separation of mixed integer rounding cuts of Marchand and Wolsey [300, 301], where inequalities are aggregated such that base inequalities correspond to cuts of the detected network. The MCF-separator of Achterberg and Raack [12] is also most closely related to the topic of the following, main chapter of the thesis.

In a recent pioneering work, Bergner et al. [45] have attempted to automate the complete branch-and-price decomposition.

5.4 An Analysis

In order to put the results into perspective, it seems important to ask the following questions:

- What is the probability of a matrix being permutable to a block-diagonal form?
- What is the expected size of the least border one can obtain by permu-

tation to a border or double-bordered diagonal form?

Intuitively, it is easy to see that for any non-trivial m, n, the number of $m \times n$ block-diagonal matrices is less than the number of $m \times n$ matrices in general. Simplistically, for k square blocks of equal size in an $k^2 \times k^2$ binary matrix, one would have for any fixed k without reordering: $\lim_{n\to\infty} k2^{k^2}/2^{k^4} = 0$. Perhaps slightly less simplistically, we consider reordering, or rather matrices distinct up to permutations of rows and columns, in this section.

Formally, let us use the following random model of matrices for m, n. On a set M of all binary $m \times n$ matrices, the relation R of matrix $A \in M$ being permutable to matrix $B \in M$ by permutations of rows and columns is clearly an equivalence. Let us have a probability space, where the sample space are the classes of "up to permutations of rows or columns" equivalence R on $k^2 \times k^2$ matrices. It is known[2] that:

Proposition 5.7 (Míšek [314, 313])**.** The number of classes of "up to permutations of rows or columns" equivalence R of $m \times n$ binary matrices is:

$$S(m,n) = \sum_{\alpha\in A} \frac{1}{\alpha_1!\alpha_2!\ldots\alpha_m!1^{\alpha_1}2^{\alpha_1}\ldots m^{\alpha_m}} \sum_{\chi\in X} \prod_{i=1}^{2^m} \left(\frac{\chi_i + c_i(\alpha) - 1}{\chi_i}\right), \quad (5.4)$$

where $c_i(\alpha)$ is the number of cycles of length i in permutation α, A, X are the set of permutations whose cycles of length j, k are α_j, χ_k $(j = 1, 2, \ldots, m; j = 1, 2, \ldots, 2^m)$, respectively, such that:

$$\alpha_1 + 2\alpha_2 + \ldots + m\alpha_m = m \quad (5.5)$$

$$\chi_1 + 2\chi_2 + \ldots + m\alpha_m = n. \quad (5.6)$$

This is related to previous results of Harary [230, 232] and uses the method of Pólya [354]. We want to show:

Theorem 5.8. Asymptotically almost surely, the probability that a binary $k^2 \times k^2$ matrix is permutable to a block-diagonal matrix with k blocks of equal dimensions is 0, considering both row and column permutations.

Proof. One can apply the same reasoning as above, looking at the proportion of block-diagonal matrices with k blocks of equal size "up to permutations of rows or columns", within $m \times n$ binary matrices "up to permutations of rows

[2]This result is often referenced, for instance in The On-Line Encyclopedia of Integer Sequences, but it does not seem to have been translated to English previously.

or columns". Let us denote the event of the class of the equivalence being block-diagonal with k blocks of equal dimensions B:

$$\begin{aligned}\mathbb{P}(B) &= \lim_{n\to\infty} \frac{kS(m,n)}{S(km,kn)} \\ &= \lim_{n\to\infty} \frac{(k-1)\sum_{\alpha\in A} \frac{1}{\alpha_1!\alpha_2!\ldots\alpha_m!1^{\alpha_1}2^{\alpha_1}\ldots m^{\alpha_m}} \sum_{\chi\in X}\prod_{i=1}^{2^m}\binom{\chi_i+c_i(\alpha)-1}{\chi_i}}{\sum_{\alpha\in A} \frac{1}{\alpha_{m+1}!\alpha_{m+2}!\ldots\alpha_{km}!1^{\alpha_1}2^{\alpha_1}\ldots(km)^{\alpha_m}} \sum_{\chi\in X}\prod_{i=1}^{2^{km}}\binom{\chi_i+c_i(\alpha)-1}{\chi_i}}\end{aligned} \tag{5.7}$$

for $c_i(\alpha), A, X$ defined as above. As it is easy to see that $S(m,n)$ grows superlinearly in both m and n, $\lim_{n\to\infty} kS(m,n)/S(km,kn) = 0$. The result follows. □

Notice that a number of generalisations of Theorem 5.8 may use the same "proof technique". It, hence, seems that one should apply such methods only to instances, where it is reasonable to expect the constraint matrix to be (bordered) block-diagonal.

Part III

Useful Structures in Integer Programming

An integer programming solver
is one big bag of tricks.

Bob Bixby [57]

While an idea that provides a big benefit for one problem class can be quite useful, [...], one practical difficulty is that MIP practitioners are typically unaware that they are confronting a problem of that class. At a minimum, a method should be able to recognize models to which it can be applied, ideally introducing little or no overhead when the model does not fit the mold.

Bob Bixby and Ed Rothberg [56]

Chapter 2 presented the branch-and-cut schema, which is employed in all state of the art solvers for integer linear programming, and Part II presented a number of incompatible alternatives. This partpresents an approach, which is by definition compatible with the branch-and-cut schema, yet makes it possible to "exploit hidden structure" as many of the incompatible alternatives do. This is illustrated on three "useful structures".

The key definition is a restriction of the notion of a structure in integer programs:

Definition 5.9. A particular structure in an integer program (Definition 2.35) is *useful* if it defines algorithms $\mathcal{A}_1, \mathcal{A}_3$ and non-trivial algorithms for two or more of the following:

- Inequality-constrained Integer Program (Problem 2.4)
- Augmentation (Problem 2.19)
- Greatest lower bound from a disjunction (Problem 2.23)
- Separation (Problem 2.27)

Algorithm $\mathcal{A}_2$ is fixed to be the usual branch-and-cut, using the two or more the (primal, improvement, branching, bounding or separation) heuristic subroutines.

By non-trivial, we mean making use of the augmented input $s, (B_i, C_i), 1 \leq i \leq s$, rather than (A, b, c) only. Subsequently, we define a structure to be

Useful structure	Primal	Improvement	Branching	Bounding
Variable aggregation	✓	✓	✓	✓
Colouring component	✓	✓	✓	✓
Precedence constraints			✓	✓

Table 5.1: The availability of primal, improvement, branching, and bounding heuristics for integer linear programming exploiting the listed useful structures.

useful, if it includes a particular structure that is useful. See Figure 5.1 for an illustration of the algorithm schema. As suggested in Table 5.1, where there is the structure, it is often possible to improve upon all four algorithms, so as to achieve improvements in both the run-time and the quality of the solutions, branching disjunctions, or cuts.

Throughout this chapter, we use the following definitions: The support of a vector $z \in \mathbb{Z}^n$ is integer sequence $S(z) = i \in N \ : \ z_i \neq 0$. A block of n variables in constraint z is a complete integer subsequence, which is composed of successive integers $\{i, i+1, i+2, \ldots, i+n-1\} \subset S(z)$. Notice that different permutations of variables give different supports and blocks. A constraint with support S is compatible with partition P of the set $\{1, 2, \ldots, n\}$, if one can partition S into elements of P. Notice that compatibility is preserved under the permutation of variables. A submatrix of an $m \times n$ matrix $M = (m_{i,j})$ is given by two index-sets $B \subseteq \{1, 2, \ldots, m\}$ and $C \subseteq \{1, 2, \ldots, n\}$. It corresponds to a $|B| \times |C|$ matrix with entries $(m_{b,c}), b \in B, c \in C$. A block of matrix M is given by two index-sets $B \subseteq \{i, i+1, \ldots, i+|B|-1\}$ and $C \subseteq \{j, j+1, \ldots, j+|C|-1\}$, which are complete integer subsequences. A row-block of matrix M is given by the index-sets $B \subseteq \{i, i+1, \ldots, i+|B|-1\}$, where B is required to be a complete integer subsequence and all columns are selected, $C \subseteq \{1, 2, \ldots, n\}, |C| = n$. Subsequently, component extraction problems try to find the maximum submatrix with a particular property, whereas block extraction problems try to find the maximum satisfying block.

A number of related approaches were studied in the early 1980s under the banner of "hidden structure discovery", "model analysis and simplification" [59, 192], and "dynamic factorisation" [208, 81], although none seem to be currently implemented in modern branch-and-cut solvers. Notice that our approach is, however, compatible with branch-and-cut solvers by definition. The stress on the compatibility with branch-and-cut solvers as described in

Chapter 2 and the particular combinatorial objects exploited seem to be original.

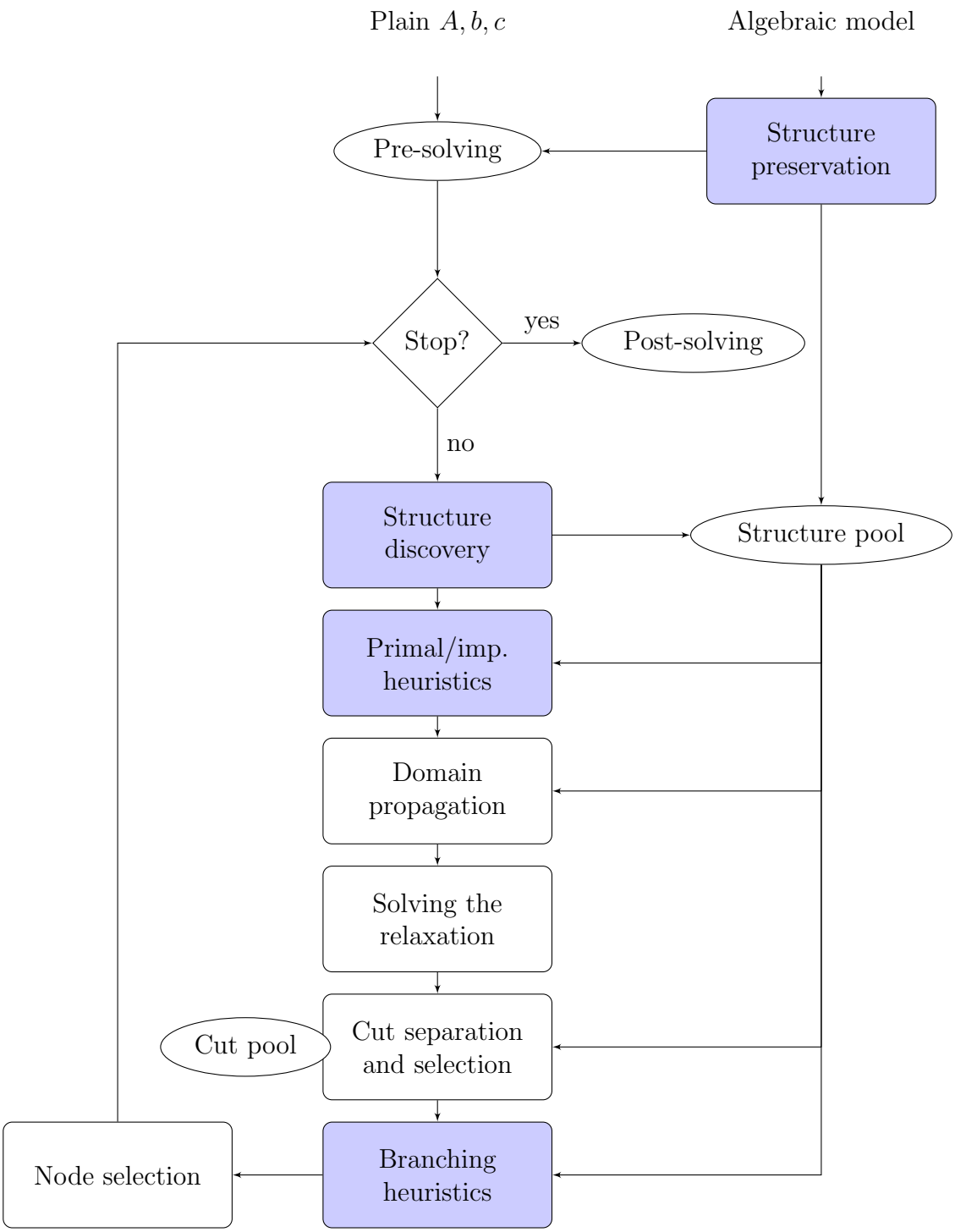


Figure 5.1: A schematic illustration of the proposed integer linear programming solvers. The highlighted boxes are discussed in this section in further detail.

Chapter 6

Variable Aggregations

Many problems combine multiple sub-problems, each with a vector variable in a compact integer linear programming formulation. The natural compact formulation of the integrated problem uses a vector variable of length given by the product of the lengths of the vector variables of the sub-problems. The number of constraints in the natural compact formulation of the integrated problem is given by the sum of sub-problems' constraints and a number of additional complicating constraints. Notice that for c sub-problems with variables of equal lengths, the constraints will be $\Omega(\sqrt[c]{n})$ dense. See, for example, Problem 1.2 with 2 sub-problems. Such "integrated" problems often pose a challenge to general-purpose solvers.

A natural approach to solving such problems is to solve one, hopefully more important, sub-problem first, while ignoring the complicating constraints, and use the solution to constrain the search for the solution of the original problem. Under certain conditions, this approach can provide both lower and upper bounds. The search for "one sub-problem" corresponds to an aggregation of variables, which is studied in this chapter.

6.1 Aggregating Variables in Integer Programs: An Introduction

Broadly speaking, the approach has three stages:

1. Aggregate binary variables: A reasonable aggregation can be obtained by solving instances of variants of set covers or set partitioning, relaxing some constraints or not
2. Translate and solve: Use aggregated variables to replace those employed to derive them in both the constraints and the objective and solve the "aggregated core"
3. Solve the original problem: Use the solution of the "aggregated core" to

constraints subsets of variables in the original problem, apply certain reductions, and try to solve it with limits on the number of nodes to explore.

Optionally, one can also identify parts of the model implementing functions such as "maximum over a set of binary variables" or "absolute value of a variable", where the value is used only in the objective function, and remove the constraints and related "auxiliary" variables. Such extensions will be studied only in Section 6.5.

Based on the compatibility with constraints, one can distinguish a number of properties of aggregation:

- *Arbitrary*: When newly introduced variables replace those employed to derive them, one may be able to derive feasible solutions, but this is not guaranteed. One may still use newly introduced variables together with those employed to derive them, in order to derive lower-bounds.
- *Bound-preserving*: When newly introduced variables replace those employed to derive them, any cuts and lower-bounds derived from the aggregated core can be translated to the original instance
- *Feasibility-preserving*: When newly introduced variables replace those employed to derive them, a feasible solution in the aggregated core corresponds to a feasible solution of the original instance, although there need not be a bijection between the sets of solutions
- *Optimality-preserving*: When newly introduced variables replace those employed to derive them, optimal solution in the aggregated core corresponds to an optimal solution of the original instance, although there need not be a bijection between the sets of solutions
- *Solutions-preserving*: When newly introduced variables replace those employed to derive them, there is a bijection between feasible solutions of the aggregated core and feasible solutions of the original instance
- *Equivalent*: The aggregation is bound-preserving, feasibility-preserving, and optimality-preserving.

Notice that arbitrary aggregations may relax any number of constraints and, hence, provide arbitrarily poor feasible solutions, if anything at all. A bound-preserving aggregation of Problem 1.2, for example, uses variables $x_{p,c} = 1$, if course c is taught at period p, and 0 otherwise, while dropping any

constraints and terms in the objective not compatible with the partition, resulting in a relaxation of the original problem. In order to obtain a solution-preserving and bound-preserving aggregation of the same problem, one only needs to include constraints restricting the number of events timetabled for each period. This provides a much better relaxation of the original problem, but still not an optimality-preserving aggregation. Notice that only heuristics for deriving equivalent aggregations seem to have so far been implemented [416] in state-of-the-art integer programming solvers.

6.2 Finding Aggregations

6.2.1 The Problem and its Complexity

Problem 6.1. r-Relaxed k-Part Variable Aggregation: Given an $m_1 \times n$ integer matrix A_1, an $m_2 \times n$ integer matrix A_2, an $m_3 \times n$ integer matrix A_3, an m_1-vector b_1 of integers, an m_2-vector b_2 of integers, an m_3-vector b_3 of integers, and an n-vector c of integers, corresponding to an integer linear program with constraints $A_1x = b_1$, $A_2x \leq b_2$, $A_3x \geq b_3$, x integral, and objective c^Tx, decide if there exists a partition P of $\{1, 2, \ldots, n\}$ into at most k subsets such that the support of the objective c and the support of all but r constraints (A_1, A_2, A_3) can be partitioned into elements of P.

Problem 6.2. Equivalent Variable Aggregation: Given an $m_1 \times n$ integer matrix A_1, an $m_2 \times n$ integer matrix A_2, an $m_3 \times n$ integer matrix A_3, an m_1-vector b_1 of integers, an m_2-vector b_2 of integers, an m_3-vector b_3 of integers, and an n-vector c of integers, corresponding to an integer linear program with constraints $A_1x = b_1$, $A_2x \leq b_2$, $A_3x \geq b_3$, x integral, and objective c^Tx, find the largest integer k such that there exists partition P of $\{1, 2, \ldots, n\}$ into k subsets such that the support of all constraints (A_1, A_2, A_3), and the objective c can be partitioned into elements of P.

Problem 6.3. Restricted Variable Aggregation: Given an $m_1 \times n$ integer matrix A_1, an $m_2 \times n$ integer matrix A_2, an $m_3 \times n$ integer matrix A_3, an m_1-vector b_1 of integers, an m_2-vector b_2 of integers, an m_3-vector b_3 of integers, and an n-vector c of integers, corresponding to an integer linear program with constraints $A_1x = b_1$, $A_2x \leq b_2$, $A_3x \geq b_3$, x integral, and objective c^Tx, find the largest integer k such that there exists partition P of $\{1, 2, \ldots, n\}$ into k subsets that are singletons the supports of set-packing constraints in $A_2x \leq b_2$, where the row of A_2 is 0/1 and the corresponding elements in b_2 is 1, such that the support of all constraints (A_1, A_2, A_3), and the objective c can be partitioned into elements of P.

Theorem 6.4. r-RELAXED k-PART VARIABLE AGGREGATION is $\mathcal{NP}$-Hard. EQUIVALENT VARIABLE AGGREGATION is $\mathcal{NP}$-Hard. RESTRICTED VARIABLE AGGREGATION is $\mathcal{NP}$-Hard.

Proof. RESTRICTED VARIABLE AGGREGATION is clearly a special case of EQUIVALENT VARIABLE AGGREGATION, which is a special case of r-RELAXED k-PART VARIABLE AGGREGATION with $r = 0$. The proof for RESTRICTED VARIABLE AGGREGATION is a simple reduction from GRAPH COLOURING (Problem 7.20): for each vertex to colour, we introduce one variable, for each edge (u, v) in the graph to colour, we introduce one equality constraint $u + v = 1$. □

Theorem 6.5. For any $\epsilon > 0$, RESTRICTED VARIABLE AGGREGATION is $\mathcal{NP}$-Hard to approximate within $n^{1-\epsilon}$.

Proof. The result is obtained easily by the same simple reduction from GRAPH COLOURING (Problem 7.20) as above and the result of Zuckerman [423]. □

Both results, albeit correct, rely on the trivial case of there being a single sub-problem. It is interesting to observe, what happens for a moderate number c of sub-problems. Clearly, the graph to colour becomes dense. For c sub-problems with variables of equal lengths, the graph to colour will be $\Omega(\sqrt[c]{n})$ dense. For sub-problems with varying lengths, the graph will only be denser. Consequently, in Lemma 6.6, we are able to show that $\sqrt{n}$ approximation is possible in polynomial time, with high probability, under the mild assumption c is constant as $n \to \infty$ and the constraints in the sub-problems are less than three-quarters dense.

6.2.2 An Integer Programming Formulation

In order to formulate the aggregation as an integer program, we first describe three families of suggestions:

- S_0 is the set of single-variable singletons
- S_1 is the union of S_0 with the family of support sets of all set-packing constraints
- S_1^+ is the power-set of S_1
- S_2 is the union of S_1 with the family of support sets of all constraints

- S_2^+ is the power-set of S_2
- S_3 is the union of S_2 with the family of complete integer subsequences in the support sets of all set-packing constraints
- S_4 is the union of S_2 with the family of complete integer subsequences in the support sets of all constraints

Notice that the cardinality of families varies widely: $|S_2| = n + m$, whereas $|S_3|$ is $O(2^{n-1})$ in general, albeit perhaps much less on structured instances. The formulations of Equivalent Variable Aggregation and Restricted Variable Aggregation differ only in the family of suggestions they use: Restricted Variable Aggregation uses S_1^+, whereas Equivalent Variable Aggregation uses $S = S_2^+$. Later, we show empirically that S_3 is sometimes a good proxy for S_1^+.

The family of suggestions is the index set of a decision variable:

$$u_s = \begin{cases} 1 & \text{if suggestion } s \text{ is "accepted"} \\ 0 & \text{otherwise} \end{cases} \qquad \forall s \in \mathcal{S} \tag{6.1}$$

The "aggregating" integer program reads:

$$\min \sum_{s \in \mathcal{S}} u_s = o \tag{6.2}$$

$$\sum_{u_s \text{ s. t. } v \in u_s} u_s = 1 \qquad \forall\, v \in \mathcal{V} \tag{6.3}$$

$$\sum_{u_s \in N} < |N| \qquad \forall\, N \in \mathcal{N} \tag{6.4}$$

The objective o determines the number variables in the "aggregated core" solved subsequently.

6.2.3 Heuristics

The use of r-Relaxed k-Part Variable Aggregation as a sub-problem is difficult not only because the problem is difficult, *per se*, but also because it is difficult to decide what parameters r, k to use. We hence propose to use methods for Restricted Variable Aggregation as heuristic methods for Equivalent Variable Aggregation and r-Relaxed k-Part Variable Aggregation with an *a priori* unknown r.

Simple Order-Dependent Heuristics The simplest heuristics for RESTRICTED VARIABLE AGGREGATION are based on the following observations: First, the order of variables used in the input to the solver is very likely to be not arbitrary, especially if the instance is generated from an algebraic model. Second, any algorithm for GRAPH COLOURING (Problem 7.20) is suitable also for RESTRICTED VARIABLE AGGREGATION. A natural class of "first-fit" graph colouring heuristics takes vertices in a fixed order, and greedily assigns vertices to an arbitrary colour class which forms an independent set together with the vertex under consideration. It is $\mathcal{NP}$-Hard to find the best possible order of vertices, but one could expect the order of variables in the input to the solver to provide one. Although for any number of vertices n, there exists a graph G_n on n vertices, on which "first-fit" greedy graph colouring performs poorly [273], "first-fit" algorithms perform well in a number of random models of graphs [273], with high probability. One can hence hope that a greedy order-dependent heuristic for RESTRICTED VARIABLE AGGREGATION may work well.

Order-Dependent Approximation Algorithms More in order to show the quality of the solutions obtained using "first-fit" heuristics for RESTRICTED VARIABLE AGGREGATION, than in order to present a practically usable algorithm, we present the following:

Lemma 6.6. When the constraints in the sub-problems are less than three-quarters dense on the average, and the number of sub-problems is much less than the dimension of the integrated problem, $k \ll n$, there exists an $O(\sqrt{n})$ approximation algorithm for RESTRICTED VARIABLE AGGREGATION, running in expected polynomial time.

Proof. This is a simple extension of the results of Coja-Oghlan and Taraz [114] for colouring dense graphs to variable aggregation. Notice that the assumptions on the constraints guarantee that the graph to colour has density d in the range $\ln^6(n) \leq d \leq 3/4$. The algorithm is based on a first-fit algorithm and a certain semidefinite program presented in Section 7.3.3, whose objective value ϑ produces a lower bound on the value of the solution obtained by the first-fit algorithm. Coja-Oghlan and Taraz [114] extend the result of Juhasz [254] (Proposition 7.3) to show that the number of graphs where the first-fit algorithm produces a solution with cost more than $O(\sqrt{n})$ worse than a ϑ lower bounded is low enough for an exponential run-time on those instances not to be enough to make the expected run-time, across all instances, super-polynomial. □

Order-Oblivious Heuristics There is a long history of spectral methods in graph partitioning [155, 22, 309, 121]. Naturally, such methods apply to both Restricted Variable Aggregation and Equivalent Variable Aggregation, and are independent on the order of variables or constraints in the input. The most simple spectral method colours the vertices greedily in the order of the corresponding values in the Fiedler eigenvector [155], corresponding to the second smallest eigenvalue of a Laplacian matrix of the corresponding graph. Various improvements of the method have been proposed, notably based on iterative splitting of partitions, until all partitions are independent sets. In Algorithms 2 and 3, we present a variant of the method due to Alon and Kahale [22] and McSherry [309].There are distributed implementations [263] of such methods.

Algorithm 2 `CProj`(A, t, τ) of McSherry [309]

Input: Non-negative integral $A \in \mathbb{Z}_+^{m+1 \times n}$, thresholds t, τ
Output: Projection matrix P_c
Mark all columns of A as unclassified
while there are at least t unclassified columns **do**
  Pick a random unclassified column i
  Find indices of columns close to i, $T_i = \{j \ : \ |P_{A^T}(A_i - A_j)| \leq \tau\}$, where A_u is the uth column of A, and mark them as classified
end while
Assign each unclassified column to the T_i closest in projection A^T
Return the projection P_c onto the span of eigenvectors of T_i

Further Heuristics Unlike in the previous three paragraphs, which presented heuristics and approximation algorithms for Restricted Variable Aggregation as heuristics for Equivalent Variable Aggregation, one may attempt to tackle Equivalent Variable Aggregation or r-Relaxed k-Part Variable Aggregation directly. One may also attempt to find a set covering and replace the pair-wise non-zero intersections with singletons.

6.3 Exploiting Aggregations

Initially, we focus on obtaining solutions and bounds. In Section 6.5, we extend the exploitation to branching heuristics and cut-generation. Unlike in Section 6.2, where we have stated the integer programming formulation

Algorithm 3 `AggregationByProjections`(A, τ)
based on the work of Alon and Kahale [22] and McSherry [309]

Input: Non-negative integral $A \in \mathbb{Z}_+^{m+1 \times n}$, which is obtained by adding the objective vector atop constraints to partition, and real threshold τ
Output: Aggregation T as a partition of {1, 2, . . . , n}
Initialise partition $T = \emptyset$
Randomly divide $\{1, 2, \ldots, n\}$ into two non-empty subsets and denote C_1, C_2 the corresponding column-submatrices of A
Let $P_1 = \texttt{CProj}(C_1, n/10, \tau)$, $P_2 = \texttt{CProj}(C_2, n/10, \tau)$
Let $P = [P_1(C_2)P_2(C_1)]$, using one submatrix to project the other
while there are non-aggregated variables **do**
    Pick index $v_i \in \{1, 2, \ldots, n\} \setminus_{t \in T} t$ of an non-aggregated variable
    Pick indices $T_i = \{u \in \{1, 2, \ldots, n\} \setminus_{t \in T} t \ : \ |P_{v_i} - P_u| \leq \tau\}$ of non-aggregated variabes and mark them as aggregated
    $T = T \cup \{T_i\}$
end while
Return partition T

of deriving the aggregation only for completeness, we actually propose to run the integer programming solver recursively on two non-trivial integer programs – in the "sub-MIP" spirit [160] – in this section, albeit only for a given number of nodes and not necessarily to optimality.

The Aggregated Core The "aggregated core" is the following integer program on o binary variables v, which correspond to subsets of supports of packing constraints, and possibly further general integer variables g:

$$\min c_v^T v + c_v^T \tag{6.5}$$
$$A_{1,v} v + A_{1,g} g = b_1 \tag{6.6}$$
$$A_{2,v} v + A_{2,g} g \leq b_2 \tag{6.7}$$
$$A_{3,v} v + A_{3,g} g \geq b_3 \tag{6.8}$$

In order to formulate the aggregated core, one needs to translate constraints $A_{i,g}$ to obtain $A_{i,v}$ and $A_{i,g}$. A number of constraints, including set-packing constraints in A_2, may be translated to lower and upper bounds or omitted completely.

In order to solve the aggregate core, we call the integer programming solver recursively, with an upper limit on the number of nodes to be explored.

The Dive Using the value of a feasible solution v, g of the "aggregated core", one can formulate the "dive" as a restriction of the original instance.

$$\min c^T x \tag{6.9}$$
$$A_1 x = b_1 \tag{6.10}$$
$$A_2 x \leq b_2 \tag{6.11}$$
$$A_3 x \geq b_3 \tag{6.12}$$
$$\sum_{i \in p} x_i = v_p \qquad \forall p \in P \tag{6.13}$$

In order to solve the dive, we call the integer programming solver recursively. We set an upper limit on the number of nodes to be expired (1000).

6.4 An Analysis

There are four important questions:

- When does there exist an aggregation into k parts, relaxing at most k variables?
- Can one extract an aggregation into k parts, relaxing at most k variables, using heuristics running in time polynomial in the number of variables and constraints, when it exists?
- When does the feasibility of a solution to the aggregated core guarantee the existence of a solution to the dive?
- When does the aggregated core provide a valid lower bound?

We answer the first two questions probabilistically, using two random models corresponding to two well-known models of random graphs [73]. In model $G(n, p)$ of random graphs on n vertices (see p. 39), each edge appears with probability p, independently of any other edge. In model $P(n, k, p)$, there is a planted k-colouring ψ, which can be seen as a function $\{1, 2, \ldots, n\} \to \{1, 2, \ldots, k\}$ determining the partition into colour classes, and edges, which appear between two vertices in two different colour classes with probability p, independently of each other.

The answer to the first questions is based on the following:

Proposition 6.7 (Matula and Kučera [306])**.** The least number of colours required to colour a graph drawn randomly from $G(n, p)$ is $(1+o(1))\frac{n}{2\log_{1/(1-p)} n}$.

The answer to the second questions is based on the following:

Proposition 6.8 (McSherry [309])**.** Let us have an instance with planted k-colouring ψ drawn randomly from $P(n, k, p)$, where the size of each colour class is linear in n. There exists a constant c such that for sufficiently large n and $p \geq c \log^3(n/\epsilon)/n$, one can recover ψ with probability $1 - \epsilon$ using Algorithm 2.

We will answer the latter two questions, in turn, for aggregations obtained using RESTRICTED VARIABLE AGGREGATION, EQUIVALENT VARIABLE AGGREGATION, and certain special cases of R-RELAXED K-PART VARIABLE AGGREGATION.

The aggregated core obtained using EQUIVALENT VARIABLE AGGREGATION is, indeed, equivalent, that is bound-preserving, feasibility-preserving, and optimality-preserving.

RESTRICTED VARIABLE AGGREGATION is bound-preserving only when the coefficients in the objective function have all the same sign. It can be feasibility-preserving, when the constraints dropped are key performance indicators:

Definition 6.9. A simplifying operation transforms an integer program into an integer program with fewer variables and constraints, such that each feasible solution of the smaller integer program is feasible for the original integer program. Key performance indicators are variables and constraints obtained in a closure of a collection of simplifying operations.

Notable simplifying operations include:

- ABS-VAL REMOVAL: Find variable v with non-zero coefficients only in constraints of the form $ax \leq cv, -ax \leq cv$ for some positive vector a, whose support does not include v, and a constant $c > 0$
- MIN-MAX REMOVAL: Find variable v with non-zero coefficients only in constraints of the form $ax \leq cv$ ($ax \geq cv$), for some positive vector a, whose support does not include v, and a constant $c > 0$
- TANGLED MIN-MAX REMOVAL: Find variable v with non-zero coefficients only in constraints of the form:

$$a_i x_i \leq c_i v \forall i \in I \tag{6.14}$$

$$\sum_{i \in I} a_i x_i \geq c_0 v \tag{6.15}$$

for some index set $I \subseteq \{1, 2, \ldots, n\}$ and a positive vector a, whose support does not include v, and constants $c_i > 0, 0 \leq i \leq |I|$, in a minimisation problem. In a maximisation problem, the inequality signs are inverted.

Algorithmically, the closure of Abs-val Removal and Min-max Removal can be obtained by finding strongly connected components in an auxiliary directed graph:

Problem 6.10 (Folklore)**.** Strongly Connected Components: Given a directed graph $G = (V, A)$ with vertices $V = 1, 2, \ldots, n$ and edges $E \subset \{(u, v) \text{ s. t. } u \neq v\}$, find the largest integer $1 \leq m \leq |V|$, such that there exists a partition (P_i) of P in $V = P \cup X, |P| = m$, such that for each two distinct $u, v \in P_i$ there exists a $u - v$ path and a $v - u$ path, where for any $u, v \in V$, $u - v$ path is a sequence of arcs $(u, w_i), (w_i, w_{i+1}), \ldots, (w_k, v) \in A$.

The vertices V of the auxiliary directed graph $G = (V, A)$ correspond to variables. For each inequality $ax \leq cv$ or $ax \geq cv$, where c is positive, a is a positive vector without u in its support, and for each u in the support, there is $(u, v) \in A$. For any other inequality or equality with u, v in its support, there are arcs $(u, v), (v, u) \in A$. Simple approaches [388, 103, 172] to solving the problem are based on depth-first search, run in time $O(|V| + |E|)$, but are inherently sequential. A number of parallel algorithms [162, 36] and algorithms requiring only an implicit representation of the graph [36] have been proposed recently.

6.5 Extensions

The concepts of key performance indicators and simplifying operations, could be employed as means of pre-processing prior to the formulation of the input to methods for Restricted Variable Aggregation or similar.

Integral key performance indicators can sometimes be used very efficiently in the generations of cuts of the form "key performance indicator is greater or equal than the ceiling of its current value" in minimisation problems. Furthermore, it seems beneficial to use cuts generated in solving the "aggregated core" in the dive and the original instance.

A similar procedure could arguably be used to branch on a variables in the aggregated "core", which would correspond to hyperplanes with support of non-zero cardinality in the original instance. Notice that for aggregations

derived from RESTRICTED VARIABLE AGGREGATION, one can obtain only branching disjunctions known as a "special ordered set". The choice of the disjunction to use can, however, be informed by the analogy with graph colouring. Notice that this is related to a rich history, of branching on specific multi-variable disjunctions, when the aggregated core corresponds to the specific problem, but this rather general branching disjunction seems novel.

6.6 COMPUTATIONAL EXPERIENCE

It seems difficult to provide a comprehensive account of the computational performance of the approach, yet we aim to illustrate:

- What is the trade-off between the number of constraints relaxed, gap closed in solving the aggregating problem (6.4–6.4), and the quality of bound the obtained in solving the aggregated core (6.5–6.5)?
- What is the computational complexity of finding an aggregation (6.4–6.4)?
- What are the effects of pre-processing?

The first question, in particular, seems to be impossible to answer in general. In an extensive study conducted on examples from timetabling, related to Example 1.2, Burke, Mareček et al. [88] have illustrated the trade-off in a considerable detail. The framework for the development of integer programming decompositions in timetabling developed in the process is freely available on-line[1]. Their conclusion, perhaps unsurprising, is that the quality of the bound deteriorates quickly with the number of constraints relaxed. The run-time required in order to solve either the aggregated core or the dive to optimality does not seem to be monotone in the number of constraints relaxed.

The answer to the second question has two parts. The optimal solution of the aggregating problem (6.4–6.4) is, indeed, hard to find. An exact solver for automating decompositions of integer programming, in general, is freely available on-line[2]. A considerable improvements could be expected from a column-generation solver for the set partitioning (6.4–6.4). It should be stressed, however, that the optimal solutions are not required. The polynomial-time

[1] Available at http://memos-solvers.sf.net/ (May 25, 2011)
[2] Available at http://philipaa.sf.net/ (May 25, 2011)

heuristics, such as `AggregationByProjections` (Algorithm 3), which provide sub-optimal solutions, seem to be much more promising.

Finally, although the simplifying operations may seem trivial, Burke, Mareček et al. [88] report that, in an encoding of instances from Track 3 of the International Timetabling Competition 2007 in integer programming, which is an extension of Example 1.2, about nine variables out of ten correspond to key performance indicators, as defined above. Consequently, pre-processed using a closure of the three simplifying operations has an immense impact in this case.

Chapter 7

Colouring Components

Some of the most important applications of integer programming are in Scheduling and Timetabling, such as Example 1.2. It could perhaps be argued that such problems are defined by having either mutual-exclusion constraints, which force pairs of events not to overlap in time, or precedence constraints, which force pairs of events not to overlap and to take place in a given sequence, or both. This section proposes methods for dealing with mutual-exclusion constraints. The following Chapter 8 describes methods for dealing with precedence constraints.

7.1 Graph Colouring and Integer Programming: An Overview

Mutual exclusion constraints correspond to:

Problem 7.1 (Folklore)**.** Graph Colouring: Given an undirected graph $G = (V, E)$ with vertices $V = 1, 2, \ldots, n$ and edges $E \subset \{(u, v)$ s. t. $1 \leq u < v \leq n\}$, find the least integer $1 \leq \chi(G) \leq |V|$, such that there exists a partition $P = (P_i)$ of V such that $|P| = \chi$ and for no edge $(u, v) \in E$ there exists a P_i with both u and v in P_i. The partition P corresponds to the "same-colour equivalence" and P_i are called "colour classes" or "independent sets". Integer $\chi(G)$ is called the chromatic number of G.

There are a number of integer programming formulations of Graph Colouring. See Table 7.1. Let us restrict ourselves to the problem of finding a feasible graph colouring, rather than minimising the number of colours used, initially, as it occurs in Timetabling and easier to state. The most usual compact formulation uses variable:

$$x_{v,c} = \begin{cases} 1 & \text{if vertex } v \text{ is coloured with colour } c \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

subject to $k\,|E|$ constraints:

$$\sum_{c=1}^{k} x_{v,c} = 1 \quad \forall \text{ vertices } v \in V \tag{7.2}$$

$$x_{u,c} + x_{v,c} \leq 1 \quad \forall \text{ colours } c \in K \quad \forall \text{ edges } \{u,v\} \in E \tag{7.3}$$

The corresponding linear programming relaxation does, however, provide a lower bound typically bounded from above by two. One can show:

Proposition 7.2 (Caprara [95])**.** In the worst-case, the linear programming lower bound $\lceil \min\{\sum_{i=1}^{n} y_i \mid (x,y) \in P\} \rceil$ is $1/(p+1)$ for partitioning polyhedra P defined by:

$$\sum_{j=1}^{n} a_{lj} x_{ij} \leq y_i \; \forall j \in 1 \ldots n, l \in 1 \ldots p, \tag{7.4}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \; \forall j \in 1 \ldots n, \tag{7.5}$$

$$0 \leq x_{ij} \leq y_i \leq 1. \tag{7.6}$$

One can strengthen the bound, but neither sophisticated cut-generation routines in in-built in leading general-purpose solvers nor problem-specific brand-and-cut codes [115, 418, 311] are particularly well-performing.

On the other hand, using a formulation in non-linear convex extension of linear programming, known as semidefinite programming, one can obtain bound ϑ where:

Proposition 7.3 (Juhasz, [254])**.** With probability $1 - o(n)$, graph G on n vertices drawn randomly from the random model $G(n,p)$, where each edge appears with probability p independently of any other edge (cf. p. 55), has

$$\vartheta(G) \geq \frac{1}{2}\sqrt{\frac{1-p}{p}}\sqrt{n} + O(n^{\frac{1}{3}} \log n) \tag{7.7}$$

Coja-Oghlan [113] has made these results much more precise, showing that ϑ of random graphs from $G_{n,p}$ is concentrated around its mean, what its value is, and that it has an exponential tail. Notice that there are near-matching upper bounds [6], with high probability. The issue seems to be in integration of such bounds into solving complex, integrated problems. In this section, we are hence exploring methods for finding the graph colouring components and exploiting them by, among others, harnessing ϑ-like bounds.

Based on	Variables	Constraints	Selected references
Vertices	$k\,\lvert V\rvert$	$\lvert V\rvert + k\,\lvert E\rvert$	Méndez-Díaz and Zabala
("The Usual Compact")			[115, 418, 311]
Binary Encoding	$\lceil \log_2 k \rceil\,\lvert V\rvert$	Exp. many	Lee [281]
Max. Indep. Sets	Exp. many	$\lvert V\rvert + 1$	Trick and Mehrotra [310]
Any Indep. Sets	Exp. many	$\lvert V\rvert + 1$	Hansen et al. [228]
Precedencies	$\mathcal{O}(\lvert V\rvert^2)$	$\lvert E\rvert$	Williams [413]
Acyclic Orientations	$\lvert E\rvert$	Exp. many	Barbosa et al. [35]
Asymmetric Repres.	$\mathcal{O}(\lvert E\rvert)$	$\mathcal{O}(\lvert V\rvert\,\lvert E\rvert)$	Campelo et al. [94]
Supernodes	$k\,\lvert Q\rvert$	$\lvert Q\rvert + k\,\lvert E'\rvert$	Present author [89]

Table 7.1: Integer linear programming formulations of colouring a graph $G = (V, E)$ (Problem 7.20). Cited from Burke, Mareček et al. [89], who describe the formulations in more detail.

7.2 Finding the Colouring Component

In many problems routinely solved using general-purpose integer linear programming solvers, such as Timetabling Problem with Objective f (Problem 1.2), the mutual-exclusion constraints are used together with many other constraints. In order to employ non-trivial methods for exploiting the mutual-exclusion component, one needs to identify it.

7.2.1 The Problem and its Complexity

Let us state the problem formally:

Problem 7.4. Colouring Component Extraction: Given positive integers n, m_1, m_2, m_3, an $m_1 \times n$ integer matrix A_1, an $m_2 \times n$ integer matrix A_2, an $m_3 \times n$ integer matrix A_3, an m_1-vector b_1 of integers, an m_2-vector b_2 of integers, and an m_3-vector b_3 of integers, corresponding to an integer linear program with constraints $A_1 x = b_1$, $A_2 x \leq b_2$, $A_3 x \geq b_3$, x integral, find the largest integer r such that there exists a graph $G = (V, E)$ and a partition C of $\{1, 2, \ldots, n\}$ into k subsets of equal cardinality $|V|$ such that

- for each vertex $v \in V$ there are k coefficients 1 in a row of A_1, which has no elements other than 0 and 1 and corresponds to 1 in b_1 ("set partitioning constraint") at columns denoted $x_{v,c}, c = 1, 2, \ldots, k$, and
- for each partition $c \in C$ and for each edge $\{u, v\} \in E$ there are 1 at

columns $x_{u,c}$ and $x_{v,c}$ in a row of A_2, which has no elements other than 0 and 1 and corresponds to 1 in b_2 ("set packing constraint"), and

- for no vertex $v \in V$ there are two columns x_{v,c_1} and x_{v,c_2}, $c_1, c_2 \in c \in C$, with coefficient 1 in any single chosen set packing constraint, and
- the total number of rows involved is r.

Notice that by maximising the number of rows involved, r, we in fact maximise the bilinear function $|V| + k|E|$ in variables k, V, E. One could, perhaps, like to maximise the lower bound one can obtain from the colouring component. Unless the graph is sub-linearly sparse, however, the number of edges in the graph dominates $|V| + k|E|$, which seems to be a good proxy for the lower bound.

The problem resembles a number of $\mathcal{NP}$-Hard problems: one needs to find a subset of variables and a subset of set partitioning constraints, so that the index set of the subset of variables can be partitioned into the support sets of the subset of set partitioning constraints that are of equal cardinality. The objective to maximise is not, however, the cardinality of the index set as in set partitioning, but rather the sum of dimensions of identity submatrices in A_2 corresponding to the support set of each of the subset of set partitioning constraints. (Notice each edge corresponds to two identity submatrices in A_2, and the submatrix for all edges corresponds to the transpose of a network matrix.) Finding such a submatrix in A_2 is the optimisation version of the TEST FOR SUBMATRIX WITH A QUALITY q, which asks whether A_2 contains a $k_1 \times k_2$ submatrix with quality "permutability to the transpose of a network matrix". (See Problem 4.10 and Proposition 4.11.) Nevertheless, TEST FOR SUBMATRIX WITH A QUALITY q does not seem to be easy to use in a proof of $\mathcal{NP}$-Hardness of COLOURING COMPONENT EXTRACTION, as it would require $n!/k_2!/(n-k_2)!$ set-partitioning constraint with support of cardinality k_2. We hence introduce a special case:

Problem 7.5. COLOURING RECOGNITION: Given integers $n, m_1, m_2, k \geq 3$ and two 0/1 matrices A_1, A_2 in dimensions $m_1 \times n$ and $m_2 \times n$, respectively, decide if there exists a graph $G = (V, E)$ and a partition C of $\{1, 2, \ldots, n\}$ into k subsets of equal cardinality $|V|$ such that

- each vertex $v \in V$ there are k coefficients 1 in a row of A_1, at columns $cv + c - 1, c = 1, 2, \ldots, k$, and
- for each partition $c \in C$ and for each edge $\{u, v\} = e \in E$ there are 1 at columns $cu + c - 1$ and $cv + c - 1$ in a row of A_2 and

- for no vertex $v \in V$ and no two partitions $c_1, c_2 \in C$ there are two columns $c_1 v + c_1 - 1$ and $c_2 v + c_2 - 1$ with coefficient 1 in any single row of A_2.

In the proofs, we employ the reduction from:

Problem 7.6 (Folklore [148]). Equitable Graph Colouring: Given an undirected graph $G = (V, E)$ with vertices $V = 1, 2, \ldots, n$ and edges $E \subset \{(u, v) \text{ s. t. } 1 \leq u < v \leq n\}$, find the least integer $1 \leq m \leq |V|$, such that there exists a partition $P = (P_i)$ of V such for each two $P_i, P_j \in P$ their cardinalities $|P_i|, |P_j|$ differ by at most one, and for no edge $(u, v) \in E$ there exists a $P_i \in P$ with both u and v in P_i.

Problem 7.7 (Folklore [148]). l-Bounded k-Colouring: Given integers $k, l \geq 3$, an undirected graph $G = (V, E)$ with vertices $V = 1, 2, \ldots, n$ and edges $E \subset \{(u, v) \text{ s. t. } 1 \leq u < v \leq n\}$, decide if there exists a partition $P = (P_i), i = 1, 2, \ldots, k$ of V such that $|P_i| \leq l$, and for no edge $(u, v) \in E$ there exists a $P_i \in P$ with both u and v in P_i.

Proposition 7.8 (Folklore, [70]). Equitable Graph Colouring is $\mathcal{NP}$-Hard. l-Bounded k-Colouring is $\mathcal{NP}$-Hard.

in order to show:

Lemma 7.9. Colouring Recognition is $\mathcal{NP}$-Hard.

Proof. For Colouring Recognition, there is a reduction from l-Bounded k-Colouring: Let us have a graph $G = (V, E)$. We construct an instance of Colouring Recognition on $k|V|$ variables. There are $|V|$ dense rows in A_1, given by $J - e \otimes I$, where J is the all 1 matrix, e is the k-vector of 1, and I is the $|V| \times |V|$ identity matrix. There are $k|E|$ rows in $A_2 = I \otimes F$, where I is the $k \times k$ identity matrix and F is the matrix, which has a row for each edge $\{u, v\} \in E$, with two ones at positions u, v. The colour classes of size at most l in graph G correspond to partitions C of size l each in Colouring Recognition. □

Theorem 7.10. Colouring Component Extraction is $\mathcal{NP}$-Hard.

Proof. Colouring Recognition is a special case of Colouring Component Extraction. □

7.2.2 An Integer Programming Formulation

Albeit perhaps of limited utility, there is, indeed, an integer linear programming formulation of COLOURING RECOGNITION. Given an $m_1 \times n$ binary matrix A_1 whose entries correspond to equalities with 1 in the right-hand-side vector, an $m_2 \times n$ matrix A_2 whose entries correspond to inequalities with 1 in the right-hand-side vector, the list of binary variables $B \subseteq \{1, 2, \ldots, n\}$, and an integer k, the variables are:

$$u_b = \begin{cases} 1 & \text{if variable } b \in B \text{ is in the colouring component} \\ 0 & \text{otherwise} \end{cases} \tag{7.8}$$

$$x_r = \begin{cases} 1 & \text{if equality } r \in A_1 \text{ is in the colouring component} \\ 0 & \text{otherwise} \end{cases} \tag{7.9}$$

$$y_r = \begin{cases} 1 & \text{if inequality } r \in A_2 \text{ is in the colouring component} \\ 0 & \text{otherwise} \end{cases} \tag{7.10}$$

$$a_{b,v,c} = \begin{cases} 1 & \text{if variable } b \in B \text{ is assigned to } v, c \\ 0 & \text{otherwise} \end{cases} \tag{7.11}$$

$$p_{b_1,b_2} = \begin{cases} 1 & \text{if variables } b_1, b_2 \in B \text{ are in the colouring component} \\ 0 & \text{otherwise} \end{cases} \tag{7.12}$$

$$d_{b_1,b_2} = \begin{cases} 1 & \text{if variables } b_1, b_2 \in B \text{ represent different colours} \\ 0 & \text{otherwise} \end{cases} \tag{7.13}$$

$$e_j = \begin{cases} 1 & \text{if colour } j \in C \text{ is used} \\ 0 & \text{otherwise} \end{cases} \tag{7.14}$$

$$f_i = \begin{cases} 1 & \text{if vertex } i \in V \text{ is used} \\ 0 & \text{otherwise} \end{cases} \tag{7.15}$$

where we use the notation $i \in A_2$ to denote taking of rows of A_2.

The objective function is clearly to maximise $\sum_{r \in A_1} x_r + \sum_{r \in A_2} y_r$. The variables are subject to:

$$f_i \geq f_{i+1} \quad \forall\, i \in V \tag{7.16}$$
$$e_j \geq f_{j+1} \quad \forall\, j \in V \tag{7.17}$$
$$a_{b,i,j} \leq e_j \quad \forall\, b \in B, i \in V, j \in C \tag{7.18}$$
$$a_{b,i,j} \leq f_i \quad \forall\, b \in B, i \in V, j \in C \tag{7.19}$$
$$\sum_{b\in B}\sum_{i\in V} a_{b,i,j} \geq \sum_{b\in B}\sum_{i\in V} a_{b,i,j+1} \quad \forall\, j \in C \tag{7.20}$$
$$\sum_{b\in B}\sum_{j\in C} a_{b,i,j} \geq \sum_{b\in B}\sum_{j\in C} a_{b,i+1,j} \quad \forall\, i \in V \tag{7.21}$$
$$\sum_{b\in B}\sum_{i\in V}(a_{b,i,j} - a_{b,i,j+1}) \leq M(e_j - e_{j+1}) \quad \forall\, j \in C \tag{7.22}$$
$$\sum_{b\in B}\sum_{j\in C}(a_{b,i,j} - a_{b,i+1,j}) \leq M(f_i - f_{i+1}) \quad \forall\, i \in V \tag{7.23}$$
$$\sum_{i\in V}\sum_{j\in C} a_{b,i,j} = u_b \quad \forall b \in B \tag{7.24}$$
$$\sum_{i\in V} a_{b,i,j} \leq u_b \quad \forall b \in B, \forall\, j \in C \tag{7.25}$$
$$\sum_{j\in C} a_{b,i,j} \leq u_b \quad \forall b \in B, \forall\, i \in V \tag{7.26}$$
$$\sum_{i\in V}(a_{b_1,i,j} - a_{b_2,i,j}) \leq d_{b_1,b_2} \quad \forall\, b_1, b_2 \in B, j \in C \tag{7.27}$$
$$\sum_{i\in V}(a_{b_2,i,j} - a_{b_1,i,j}) \leq d_{b_1,b_2} \quad \forall\, b_1, b_2 \in B, j \in C \tag{7.28}$$
$$u_{b_1} + u_{b_2} - 1 \leq p_{b_1,b_2} \quad \forall\, b_1, b_2 \in B, b_1 \neq b_2 \tag{7.29}$$

$$d_{v_1,v_2} \leq p_{v_1,v_2} + x_r - 1 \quad \forall\, r \in A_1, v_1, v_2 \in \mathrm{supp}(r), j \in C \tag{7.30}$$
$$1 - d_{v_1,v_2} \geq p_{v_1,v_2} + y_r - 1 \quad \forall\, r \in A_2, v_1, v_2 \in \mathrm{supp}(r), i \in V \tag{7.31}$$
$$0 \geq x_r \geq 1, 0 \geq u_b \geq 1 \quad \forall\, r \in A_1, b \in B \tag{7.32}$$
$$0 \geq y_r \geq 1, 0 \geq f_v \geq 1 \geq 1 \quad \forall\, r \in A_2, v \in V \tag{7.33}$$
$$0 \geq e_c \geq 1, 0 \geq a_{b,v,c} \geq 1 \quad \forall\, b \in B, v \in V, c \in C \tag{7.34}$$
$$0 \geq p_{b_1,b_2} \geq 1, 0 \geq d_{b_1,b_2} \geq 1 \quad \forall\, b_1, b_2 \in B \tag{7.35}$$

where we assume $i + 1$ for i on the extreme of an interval results in the expression being evaluated to zero. Considering the problem does not seem to have been studied, yet, this seems to be the first description of the polyhedron.

Algorithm 4 `ColouringComponentByCounting`(A_1, b_1, A_2, b_2)

Input: 0/1 matrices A_1, A_2 in constraints $A_1x = 1$, $A_2x \leq 1$
Output: Submatrix S of $[A_1; A_2]$ of dimension $f \times g$
$S = []$
$N = \{|S(z)| \mid z \text{ is a row of } A_1\}$
for n_i repetitions of number k_i in N, in the descending order of k_in_i **do**
    For the subset set-partitioning constraints corresponding to n_1 repetitions of number k_1, we find the non-overlapping subset of supports of that maximises the number of rows used
    Update S if it had more rows than previously
end for
Return S

7.2.3 Heuristics

Counting Heuristics A simple heuristic is based on counting non-zeros in set-partitioning constraints. When one sorts the numbers, one can find n_1 repetitions of number k_1, such that k_1n_1 is maximised, in time linear in the number of numbers. For the subset set partitioning constraints corresponding to n_1 repetitions of number k_1, one can try to find the non-overlapping subset of supports of that maximises the number of rows used. In order to do so, one may use a greedy set-partitioning heuristic and hope that such a set-partitioning solution will also provide a good solution with respect to the total number of rows. If necessary, this can be repeated for n_i repetitions of number k_i, in the descending order of k_in_i. See Algorithm 4.

Matching Heuristics Another simple heuristic is based on the expose-and-merge approach of Matula and Kučera [306]: Iteratively, auxiliary bipartite graphs are constructed, perfect matching is sought therein. See Algorithm 5.

7.3 EXPLOITING THE COLOURING COMPONENT

There is a rich history of methods for graph colouring and extensions, although the extensions studied are often very specific, rather than the general setting "colouring is represented by this vector and there are some arbitrary further equalities and inequalities given by these two matrices and two vec-

Algorithm 5 `ColouringRecognitionByMatching`(A_1, b_1, A_2, b_2)

Input: 0/1 matrices A_1, A_2 in constraints $A_1x = 1$, $A_2x \leq 1$
Output: "Yes" or "no"
Construct an auxiliary graph $G = (V, E)$, where vertices correspond to columns of A_1, A_2, and for each row of A_2 and two coefficients 1 therein, there is an edge in E between the corresponding vertices
for each integer k such that $|V| = ck$ and there are c rows in A_1 with more than k coefficients 1 **do**
  Initialize list L with vertices V in a fixed, random order
  Partition list L into $L_1, L_2, \ldots, L_c$, such that $|L_i| = k$
  Initialize a family of sets P with singletons from L_1
  for each $L_i \in L_2, L_3, \ldots, L_k$ **do**
    Construct an auxiliary bipartite graph B_i on $k + k$ vertices, representing L_i and sets in P, where there is an edge (v, w), if and only if there are no edges between the vertex of L_i represented by v and any of the vertices in the set of P represented by w
    if there is a perfect matching M in H **then**
      for each edge $(v, w) \in M$ **do**
        Add the vertex in L_i represented by v to the set in P represented by w
      end for
    else
      Continue with another k
    end if
  end for
end for
if partition P gives an instance of graph colouring **then**
  return "Yes"
  return "No"
end if

tors", which we focus on. We first survey a variety of long-established methods, which can perform surprisingly well, at least when there are no further constraints. Subsequently, in Sections 7.3.2 and 7.3.3, we study two reformulations of graph colouring and the means of accommodating the general extension suggested above.

7.3.1 Long-Established Means of Exploitation

There is a long history of work on methods for graph colouring. For an excellent survey of the pioneering work up until 1984, see Shneider [383]. Much of this early work has focused on the ordering of vertices in greedy algorithms and the closely related branching rules for branch and bound solvers based on:

Proposition 7.11 (Zykov [424])**.** For any graph $G = (V, E)$, and any two non-adjacent vertices $u, v \in V$, the chromatic number $\chi(G)$ is the minimum of $\chi(G_1)$ and $\chi(G_2)$, where:

- $G_1 = (V \setminus \{v\}, \{\{f(x), f(y)\} \mid \{x, y\} \in E, f \text{ replaces } v \text{ by } u\})$
- $G_2 = (V, E \cup \{\{u, v\}\})$.

It is known [210] that for an arbitrary ordering of vertices, and for any polynomial number of such orderings, sampled randomly, the limit of the ratio of the number of colours used to the number of colours required goes to 2 as the number of vertices goes to the large limit in random graphs with constant edge-probability. This does not compare well with the expose-and-merge approach of Matula and Kučera [306], which uses less than $(\frac{4}{3} + \delta)\chi_n$ colours in random graphs with constant edge-probability with high probability. (See [170] for an extensive survey of related probabilistic results.) Nevertheless, there is empirical evidence [383, 104] that some orderings perform reasonably well on dense graphs. Notably, "highest saturated degree first" [104] is widely accepted as a benchmark and can be used as both a primal and a branching heuristic. Such methods, albeit imminently useful, are however outside of the scope of this thesis.

7.3.2 Automated Reformulations in Linear Programming

In this section, we propose a reformulation, based on multi-colouring of a particular type of clique partition. Let us state some well-known definitions and facts first.

Problem 7.12 (Folklore, [225])**.** GRAPH MULTICOLOURING: Given an undirected graph $G = (V, E)$ with vertices $V = 1, 2, \ldots, n$ and edges $E \subset \{(u, v) \text{ s. t. } 1 \leq u < v \leq n\}$, and a demand function $f : V \to \mathbb{N}$, find the least integer k, such that there exists a mapping $c : V \to 2^{\{1,2,\ldots,k\}}$, such

that for all $v \in V : |c(v)| = f(v)$ and for all $\{u, v\} \in E$, $c(u) \cap c(v) = \emptyset$. We also require $\bigcup_{v \in V} c(v) = K$.

Definition 7.13. The *clique partition* of graph $G = (V, E)$ is a partition Q of vertices V, such that for all sets $q \in Q$, all $v \in q$ are pairwise adjacent in G. We use $v \in q \in Q$ to denote that vertex v in the original vertex-set V is an element of a clique represented by q in the clique partition Q.

Graph Multicolouring is clearly $\mathcal{NP}$-Hard as a special case of graph colouring. Minimum Clique Partition, the problem of finding the minimum cardinality of a clique partition $\bar{\chi}(G)$, is also $\mathcal{NP}$-Hard in general graphs and as hard to approximate as graph colouring itself. Indeed, $\bar{\chi}(G) = \chi(\bar{G})$, where $\chi(\bar{G})$ is the minimum number of colours needed to colour the complement graph. Notice V is a valid, if trivial, clique partition of graph G. Next, we introduce the *indistinguishability relation* between vertices of a graph:

Definition 7.14. Two vertices $u, v \in V$ of a graph $G = (V, E)$ are *indistinguishable*, if and only if they are adjacent and have identical closed neighbourhoods; that is: $\{w \mid \{u, w\} \in E\} \cup \{u\}$ is the same as $\{w \mid \{v, w\} \in E\} \cup \{v\}$.

This relation has been discovered independently a number of times. The earliest mention known to us are the "closed" sets of Gallai [173]. It has also been studied in game theory [382], partially ordered sets [203], in connection with mass elimination [185], under names such as externally related sets [223], supervariables [139], prototype vertices [147], and autonomous sets [316]. Muller and Spinrad [320] suggest the term "module". It is easy to observe that the relation is an equivalence, that is reflexive, symmetric, and transitive. As such, we use it to define a particular type of clique partition:

Definition 7.15. The *reversible clique partition* Q of a graph $G = (V, E)$ is the clique partition of minimum cardinality such that each *supernode* $q \in Q$ represents a class of equivalence for an indistinguishability relation on G.

This means that for each supernode $q \in Q$ of the reversible clique partition (Q, E'), each two vertices $u, v \in q$ are indistinguishable. Reversible clique partition is closely related to modular [126] and substitution decomposition [316], ordinal sum [368], and X-join [223]. It is also a special case of 2-join of the strong perfect graph theorem [108]. As usual, we will be interested also in the graph induced by the clique partition:

Definition 7.16. The *graph induced by reversible clique partition* Q of graph $G = (V, E)$ is the graph $G' = (Q, E')$, where $E' = \{\{q_u, q_v\} | \{u, v\} \in E, q_u, q_v \in Q, q_u \neq q_v, u \in q_u, v \in q_v, \}$.

Algorithm 6 `ReversibleCliquePartition`(G)

1: **Input:** Graph $G = (V, E)$
2: **Output:** Reversible clique partition Q of G
3: Construct an auxiliary graph $H = (V, F)$, where there is an edge $\{u, v\} \in F$, if and only if there is an edge $\{u, v\} \in E$ and vertices u and v are indistinguishable in G
4: Run depth-first search on H to obtain collection Q of connected components of H
5: Return Q

The use of the word *induced* in this context is reasonable, because it corresponds to a subgraph induced by taking a subset of the original vertex set with a single (arbitrary) representative of each supernode. The "reversibility" of the clique partition is, indeed, rather a strict requirement. As the name suggests, it makes it possible to reconstruct the original graph. It also makes it possible to employ the usual[1] polynomial-time algorithm for finding classes of an equivalence relation. See Algorithm 6.

From the fact that the indistinguishability relation is an equivalence, we can easily deduce that:

Theorem 7.17. For all graphs G and integers $k > 0$, there is a multicolouring M of the graph induced by reversible clique partition Q of G with k colours and demand function implied by the cardinality of sets in Q, as produced by Algorithm 6 in time $\mathcal{O}|V|\ |E|$, if and only if there is a colouring C of G with k colours.

Subsequently, the standard integer programming formulation of vertex multicolouring can also be used as a formulation of vertex colouring. Given the graph $G' = (Q, E')$ induced by reversible clique partition Q of graph $G = (V, E)$ together with the demand function $f : V \to \mathbb{N}$, specifying the number $f(q)$ of colours to attach to each vertex $q \in Q$ out of the set $K = \{1, \ldots, k\}$, this is the integer programming formulation with $k\,|Q|$ binary variables:

$$x_{q,c} = \begin{cases} 1 & \text{if colour } c \text{ is included in the set assigned to } q \in Q \\ 0 & \text{otherwise} \end{cases} \tag{7.36}$$

[1] Alternatively, one can also use the algorithm of Muller and Spinrad [320] to improve the run-time to $O(|V|^2)$.

Figure 7.1: Example of a graph and a partition of its vertex-set into supernodes. Notice supernodes B' and C' need to be assigned two distinct colours each, distinct from the colour(s) assigned to A' and D'. Within each supernode, colours can be interchanged freely.

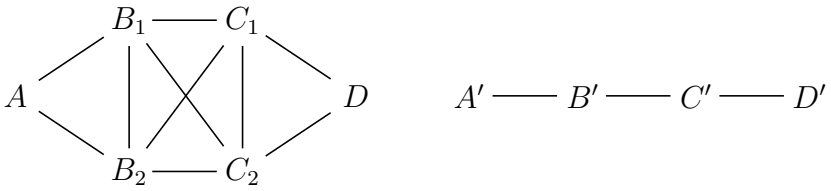


Figure 7.2: The standard and the proposed encoding of a particular colouring of the graph from Figure 7.1:

(a) The Standard Encoding

Vertex	Colour			
	1	2	3	4
A	1	0	0	0
B_1	0	1	0	0
B_2	0	0	1	0
C_1	0	0	0	1
C_2	1	0	0	0
D	0	1	0	0

(b) The Proposed Encoding

Partition	Colour			
	1	2	3	4
A	1	0	0	0
B	0	1	1	0
C	1	0	0	1
D	0	1	0	0

subject to $|Q| + k\,|E'|$ constraints:

$$\sum_{c=1}^{k} x_{q,c} = f(q) \qquad \forall q \in Q \tag{7.37}$$

$$x_{u',c} + x_{v',c} \leq 1 \qquad \forall c \in K \quad \forall \{u', v'\} \in E' \tag{7.38}$$

See Figure 7.2 for an example. It is easy to see that there exists a proper vertex colouring of $G = (V, E)$ with k colours, if and only if there exists a multicolouring of a reversible clique partition (Q, E') of G with k colours, which exists if and only if the integer programming formulation has a feasible solution for the given instance. When a graph has only a trivial reversible clique partition, this formulation is reduced to the standard formulation (7.1–7.3) of graph colouring. Otherwise, the proposed formulation avoids some symmetries inherent in the standard graph colouring formulation, which assigns unique colours (or "labels") to individual vertices. If there were a trivial inte-

ger programming solver, using neither bounding, nor cuts, this formulation should reduce its search space by a factor of $\prod_{q \in Q} |q|!$ when compared to the standard formulation (7.1–7.3). Although it is much more difficult to predict run times in modern integer programming solvers, it is obvious that there are $k(|V| - |Q|)$ fewer variables, in the proposed formulation than in the standard one. It seems that the number of constraints is also reduced, often by more than $k(|V| - |Q|)$, without making the constraint matrix considerably denser. Notice also that with minor modification of `ReversibleCliquePartition` (Algorithm 6), one can accommodate arbitrary constraints outside of the graph colouring component, even when variables outside of the graph colouring component are involved.

7.3.3 Automated Non-Linear Relaxations

Much more promisingly still, one may derive and hope to solve non-linear programming relaxations, as mentioned in Proposition 7.3. The relaxations we focus on are based on semidefinite programming, as introduced in Section 2.5.3.

The Relaxations Semidefinite programming relaxations of graph colouring are based on the primal $n \times n$ matrix variable, which should be seen as:

$$X_{u,v} = \begin{cases} t & \text{if vertex } u \text{ is in the same colour class as } v \\ 0 & \text{otherwise.} \end{cases} \tag{7.39}$$

for a positive value t. Considering Proposition 7.11, this is sometimes known as the Zykovian representation. Notice that such matrix X has the "block diagonal property":

Proposition 7.18 (Folklore)**.** For any value of X, there exists a permutation matrix P, such that

$$P^T M P = \begin{bmatrix} J_{c_1} & 0 & & 0 \\ 0 & J_{c_2} & & 0 \\ & & \ddots & \\ 0 & 0 & & J_{c_s} \end{bmatrix} \tag{7.40}$$

where J_c is the $c \times c$ matrix of all ts.$P^T M P$ is then denoted a direct sum of J_c.

The constraints can be derived by considering:

Proposition 7.19 (Dukanovic and Rendl [140])**.** For any symmetric 0-1 matrix M there exists a permutation matrix P such that $P^T MP$ is the direct sum of s all-ones matrices if and only if there is a vector of all-ones on the diagonal and the rank of M is s and M is positive semidefinite.

Notice that an $\mathcal{NP}$-Hard decision problem related to graph colouring (cf. Problem 7.6, Proposition 7.8) can hence be seen as a rank-constrained problem. Consequently, the rank-constrained problem is $\mathcal{NP}$-Hard, and only a relaxation replacing the constraint on the rank of the matrix variable with a constraint to positive semidefinite matrices is an SDP. Alternatively, one can see theta as an eigenvalue bound, where the largest eigenvalue $\lambda_{\max}(A) = \min\{t \text{ s. t. } tI - A \succeq 0\}$ for an identity matrix I. Perhaps most "fundamentaly", one could see theta as a relaxation of the co-positive programming formulation of graph colouring, recently proposed by Bomze [74]. A number of other derivations have been surveyed by Knuth [267].

The complication is that the relaxation above is not a semidefinite program in the standard form (P STD, D STD). Notice that the scalar variable t has been introduced only for clarity. As long as the entries on the diagonal of the matrix variable are constrained to be equal, any one of them can be used instead. One can either introduce new scalar slack variables and convert inequalities to equalities, or one can design solvers treating inequalities explicitly. There remains the constraint $Y - J \succeq 0$ to deal with. The mechanistic approach, employed by automated model transformation tools [286], for instance, is to double the dimension by introducing a new variable X, set $Y - J = X, X \succeq 0$. For an arbitrary vertex $w \in V$, one obtains:

$$\begin{aligned} & \max Y_{w,w} && (7.41)\\ \text{s. t. } & X_{u,v} = -1 \quad \forall \{u,v\} \in E && (\text{A1})\\ & X_{v,v} = X_{w,w} \quad \forall v \in V \setminus \{w\} && (\text{A2})\\ & Y_{u,v} - X_{u,v} = 1 \quad \forall u,v \in V && (\text{W})\\ & X \succeq 0 \end{aligned}$$

An alternative approach is to optimise:

$$\begin{aligned} & \max X_{w,w} + 1 && (7.42)\\ \text{s. t. } & X_{u,v} = -1 \quad \forall \{u,v\} \in E && (\text{A1})\\ & X_{v,v} = X_{w,w} \quad \forall v \in V \setminus \{w\} && (\text{A2})\\ & X \succeq 0 \end{aligned}$$

Any formulation involving $Y \succeq J$ can be easily transformed in this fashion. Later, we describe solvers specialised to solving SDP in this extended form. As it turns out, treating W explicitly is not much of a burden in such specialised methods, as the additional inequalities (W) correspond to $n^2 \times 2n^2$ matrix $W = [I, -I]$, where I is $n^2 \times n^2$ identity matrix and $[A, B]$ denotes concatenation of compatible matrices A, B, which is particularly easy to work with.

Extensions It seems clear that one should be able to strengthen the semidefinite programming relaxation towards the chromatic number constraint by the addition of linear equalities and inequalities:

Problem 7.20. GRAPH COLOURING WITH CONSTRAINTS IN ZYKOVIAN REPRESENTATION: Given an undirected graph $G = (V, E)$ with vertices $V = 1, 2, \ldots, n$ and edges $E \subset \{(u, v)$ s. t. $1 \leq u < v \leq n\}$, integers $m_1, m_2, m_3 > 0$, an $m_1 \times n^2$ integer matrix A_1, an $m_2 \times n^2$ integer matrix A_2, an $m_3 \times n^2$ integer matrix A_3, an m_1-vector b_1 of integers, an m_2-vector b_2 of integers, find the least integer $1 \leq m \leq |V|$, such that there exists a partition $P = (P_i)$ of V such that for no edge $(u, v) \in E$ there exists a P_i, $1 \leq |P_i| \leq m$, with both u and v in P_i, and there exists an $n \times n$ matrix $X = (X_{u,v})$, such that $X_{u,v}$ is positive if and only if there exists P_i with $u, v \in P_i$, and $A_1 X = b_1, A_2 X \leq b_2, A_3 X \geq b_3$.

In many timetabling applications, there is, for example, an additional linear inequality, which bounds the number of uses of a colour. In this extension, known as m-bounded colouring (Problem 7.20), we expect row-sums and column-sums in the binary-valued variable to be bounded from above by m:

$$\max t \tag{7.43}$$

$$\text{s. t. } Y_{vv} = t \quad \forall v \in V \tag{E1}$$

$$Y_{uv} = 0 \quad \forall \{u, v\} \in E \tag{E2}$$

$$\sum_{u \in V} Y_{uv} \leq tm \quad \forall v \in V \tag{IN}$$

$$Y - J \succeq 0 \tag{PSD}$$

where Y is an $n \times n$ matrix variable and t is a scalar variable. Notice that this constraint is particularly easy to extract and particularly easy to work with in specialised solvers. For a graph on n vertices and m edges:

- First m equalities (A1) correspond to $m \times n^2$ matrix A_1. $A_1 A_1^T = I$, where I is an identity matrix of the same dimension as A_1.
- Further $n-1$ equalities (A2) correspond to $n-1 \times n^2$ matrix A_2. $A_2 A_2^T = J + I$, where I and J are $(n-1) \times (n-1)$ identity and all-ones matrices. $(A_2 A_2^T)^{-1}$ is $-\frac{1}{n} J + I$. For $(n-1)$- vector y, $A_2^T y$ is an $n \times n$ matrix, with $[(\sum_i y_i)\,(-y_1)(-y_2)\cdots(-y_{n-1})]$ on the diagonal and zeros elsewhere. For positive X, $\mathcal{L}_{A_2}(X) = [2, 4, \cdots, 2(n-1)]$ of dimension $(n-1)$.
- Inequalities (IN) correspond to $n \times n^2$ matrix $B = I \otimes e$, where e is the row-vector of n ones. Hence, $BB^T = nI$. For an n-element column-vector v, $B^T v = (v \otimes e)^T = [v_1 e\ v_2 e\ \cdots\ v_n e]^T$, where e is the row-vector of n ones.
- The elements of the objective matrix C are zeros except for $C_{1,1} = 1$. Hence $\mathcal{L}_{A_1}(C) = 0$, where 0 is the zero m-vector. $\mathcal{L}_{A_2}(C) = e$, where e is the $(n-1)$-vector of ones.

As will be detailed in Section 7.3.4, these properties allow us to design efficient solvers that do not store A_1, A_2, B explicitly. It is also clear such a simple constraint can be easily identified and translated into the Zykovian representation.

In more complex timetabling applications, such as Example 1.2, one may also want to represent the assignment of events to rooms. In contrast to the integer linear programming formulation (p. 4), in semidefinite programming, one can represent the assignment of events to rooms by:

$$R_{v,r} = \begin{cases} 1 & \text{if vertex } v \text{ is assigned (room) } r \\ 0 & \text{otherwise.} \end{cases} \tag{7.44}$$

and use constraints such as:

$$\sum_{1 \le r \le m} R_{v,r} = t \quad \forall\, v \in V \tag{7.45}$$

$$R_{v,r} + R_{v,r'} \le t \quad \forall\, v \in V, 1 \le r, r' \le m, r \ne r' \tag{7.46}$$

$$R_{u,r} + R_{v,r} + Y_{u,v} \le 2t \quad \forall\, u, v \in V, u \ne v, 1 \le r \le m \tag{7.47}$$

$$R_{v,r} = 0 \quad \forall\, v \in V, 1 \le r \le m, p_v \ge r_r \tag{7.48}$$

$$R_{v,r} = 0 \quad \forall\, v \in V, 1 \le f \le f_{\max}, \quad (v,f) \in F, 1 \le r \le r, (r,f) \notin G \tag{7.49}$$

$$R_{v,r} \ge 0 \quad \forall\, v \in V, 1 \le r \le m \tag{7.50}$$

In timetabling language: each event is in exactly one room (7.45, 7.46), events in the same timeslot do not share rooms (7.47), and $R_{v,r} = 0$ if the event-room combination does not match the event's room-feature or capacity requirements (7.48,7.49). One could devise algorithms for extracting complicating constraints of the usual integer linear programming formulation of Example 1.2, and translate them to the constraints (7.45–7.50) in the Zykovian representation. It is not clear, however, whether the translation of arbitrary constraints to the Zykovian representation of Problem 7.20 can ever be fully automated, especially when variables outside of the graph colouring component are involved.

Solving the Relaxation Computationally efficient methods for semidefinite programming are under active development. Recently, there has been a considerable interest in the development of augmented Lagrangian methods [143, 144] for semidefinite programming [298, 86, 422, 355, 409, 202]. In computing theta bounds, such methods are superior to interior point methods [409]. We specialise the method to the extended formulation:

$$z_p = \min_{X \in \mathbb{S}^n} \langle C, X \rangle \tag{7.51}$$

$$\text{s. t.} \quad \mathcal{L}_{A_1}(X) = b_1 \text{ and } \mathcal{L}_{A_2}(X) = b_2 \text{ and } \mathcal{L}_B(X) \geq d \text{ and } X \succeq 0$$

$$z_d = \max_{y_1 \in \mathbb{R}^m, y_2 \in \mathbb{R}^p, v \in \mathbb{R}^q, S \in \mathbb{S}^n} b_1^T y_1 + b_2^T y_2 + d^T v \tag{7.52}$$

$$\text{s. t.} \quad \mathcal{L}_{A_1}^*(y_1) + \mathcal{L}_{A_2}^*(y_2) + \mathcal{L}_B^*(v) + S = C \text{ and } S \succeq 0 \text{ and } v \geq 0,$$

where $\mathbb{S}^n$ is the set of $n \times n$ symmetric matrices and $X \succeq 0$ denotes X is positive semidefinite (See p. 32.) The augmented Lagrangian of the dual (7.52) is:

$$\begin{aligned} L_\mu(X, y_1, y_2, v, S) = &- b_1^T y_1 - b_2^T y_2 - d^T v \\ &+ \langle X, \mathcal{L}_{A_1}^*(y_1) + \mathcal{L}_{A_2}^*(y_2) + \mathcal{L}_B^*(v) + S - C \rangle \\ &+ \frac{1}{2\mu} ||\mathcal{L}_{A_1}^*(y_1) + \mathcal{L}_{A_2}^*(y_2) + \mathcal{L}_B^*(v) + S - C||_F^2 \end{aligned} \tag{7.53}$$

This extends the approach of Wen et al. [409], who treat linear inequalities (P SDP) explicitly, and can be specialised to the additional constraints encountered in graph colouring.

The multiple splitting is elaborated in Algorithm Schema 7. Notice that finding the minimiser of the augmented Lagrangian for v in Line 9 corresponds to solving a convex quadratic program. The most computationally demanding step, which is computing the minimiser of S in Line 10, consists

of computing the spectral decomposition of the term subtracted from S on the right hand side [387]. One can show:

Proposition 7.21 (Wen et al. [409])**.** Algorithm Schema 7 converges to optimal solution for any $\mu \geq 0$ and any semidefinite program in standard form (P STD).

Using considerably more involved and, so far unpublished, arguments of Goldfarb and Ma [202] and well-known lower bounds [325], it seems that one can show:

Conjecture 7.22. Algorithm Schema 7 converges to solution with ϵ error in a number of iterations asymptotically optimal in ϵ, for any semidefinite program in the extended form (7.52).

The conjecture can be proven easily, if the arguments of Goldfarb and Ma [202] are correct, which seems to be the case, indeed.

Algorithm 7 `AugmentedLagrangianMethod`$(A_1, A_2, B, C, b_1, b_2, d)$

Input: Instance I $= (A_1, A_2, B, C, b_1, b_2, d)$ of SDP (7.52)

Output: Primal solution Y, computed up to a certain precision

Initialise $X^k \succeq 0$ with a heuristically obtained colouring and set counter $k = 0$

Compute matching values of dual variables $y_1^k, y_2^k, v^k \geq 0$, and $S^k \succeq 0$

while the precision is insufficient **do**

Increase iteration counter k

Update y_1^{k+1} $= \operatorname{argmin}_{y_1 \in \mathbb{R}^m} L_\mu(X^k, y_1, y_2^k, v^k, S^k)$
$= -(A_1 A_1^T)^{-1}(\mu(A_1(X^k) - b_1) + A_1(A_2^T(y_2^k) + B^T(v^k) + S^k - C))$

Update y_2^{k+1} $= \operatorname{argmin}_{y_2 \in \mathbb{R}^m} L_\mu(X^k, y_1^{k+1}, y_2, v^k, S^k)$
$= -(A_2 A_2^T)^{-1}(\mu(A_2(X^k) - b_2) + A_2(A_1^T(y_1^{k+1}) + B^T(v^k) + S^k - C))$

Update v^{k+1} $= \operatorname{argmin}_{v \in \mathbb{R}^q, v \geq 0} L_\mu(X^k, y_1^{k+1}, y_2^{k+1}, v, S^k)$
$= \operatorname{argmin}_{v \in \mathbb{R}^q, v \geq 0} \left(\left(B\left(X^k + \frac{1}{\mu}\left(A_1^T(y_1^{k+1}) + A_2^T(y_2^{k+1}) + S^k - C \right) \right) - d \right)^T v + \frac{1}{2\mu} v^T (BB^T) v \right)$

Update S^{k+1} $= \operatorname{argmin}_{S \succeq 0} L_\mu(X^k, y_1^{k+1}, y_2^{k+1}, v^{k+1}, S)$
$= \operatorname{argmin}_{S \in \mathbb{S}^n, S \succeq 0} \left\| S - \left(C - A_1^T(y_1^{k+1}) - A_2^T(y_2^{k+1}) - B^T(v^{k+1}) - \mu X^k \right) \right\|_F^2$

Update X^{k+1} $= X^k + \frac{A_1^T(y_1^{k+1}) + A_2^T(y_2^{k+1}) + B^T(v^{k+1}) + S^{k+1} - C)}{\mu}$ any step-length $\mu \geq 0$

end while

Return X

Algorithm 8 `IterativeRounding`(X)
based on Karger, Motwani, and Sudan [258]

Input: Matrix variable X of the solution to the SDP (7.51) of dimensions $n \times n$, bound m, number $a_{\max}$ of randomisations to test, constraint matrices A_1, A_2 and compatible vectors b_1, b_2 of Graph Colouring with Constraints in Zykovian Representation
Output: Partition P of the set $V = 1, 2, \ldots, n$
Compute vector $v, X = v^T v$ using Cholesky decomposition
for Each attempted randomisation $a = 1, \ldots, a_{\max}$ **do**
  Initialise $P_a = \emptyset, i = 1, X = V$
  while There are uncoloured vertices in X **do**
    Pick a suitable $c = \sqrt{\frac{2(k-2)}{k \log_e \Delta}}$ for Δ being the maximum degree of the vertices in X
    Generate a random vector r of dimension $|X|$
    Pick $R_i \subseteq X$ of at most m elements in the descending order of $v_i r_i$, where (1) positive and (2) independent of previously chosen and, (3) $\mathcal{L}_{A_1}(X) = b_1$ and (4) $\mathcal{L}_{A_2}(X) = b_2$ and (5) $\mathcal{L}_B(X) \geq d$
    Update $P_a = P_a \cup \{\{R_i\}\}, X = X \setminus R_i$, $i = i + 1$
  end while
end for
Return P_a of minimum cardinality

Recovering an Assignment Since the seminal paper of Karger, Motwani, and Sudan [258], there has been a continuing interest in algorithms recovering a colouring from semidefinite relaxations. Typically, such algorithms are based on simple randomised iterative rounding of the semidefinite programming relaxation. One such algorithm, specialised to Graph Colouring with Constraints in Zykovian Representation is displayed in Algorithm Schema 8.

7.3.4 Computational Experience

Linear Reformulation In order to illustrate the performance of the multicolouring reformulation, we report:

- the dimensions of reversible clique partitions obtained from graphs in the DIMACS collection [251], a benchmark in graph colouring, available

on-line[2]

- performance gains on graph colouring components from timetabling application and complete instances of timetabling problems. Four timetabling instances were taken from Udine Course Timetabling [178] and eighteen more instances were obtained using a pseudo-random generator[3] developed by the author. Their dimensions are listed in Table C.2.

All instances were passed to IBM/ILOG CPLEX in LP format as generated from sources in Zimpl, the free algebraic modelling language [268], and are available on-line in Zimpl format.

Firstly, we have obtained reversible clique partitions from DIMACS graphs. To illustrate the effects of pre-processing of the original graph on the size of the reversible clique partition, in Table C.1, we list the sizes first without using any preprocessing (under Q), as well as after some pre-processing that is specific to graph colouring, but not specific to the transformation (under Q'). This preprocessing included the removal of vertices of degree less than a lower bound on the chromatic number, vertices connected to all other vertices in the graph, and each vertex u whose neighbourhood is a subset of the neighbourhood of another non-adjacent vertex v. For details of the pre-processing and the source code used, please see the author's website[4].

Secondly, we compared the performance of the standard formulation of graph colouring (7.1–7.3) and the performance of the multicolouring reformulation (7.36–7.38) on the graph colouring component of instances of Udine Course Timetabling. We report run-time of CPLEX 10 restricted to use only a single thread on a Intel Pentium 4 processor at 3.20 GHz. Default parameter settings were used, outside of the settings for symmetry breaking that are described below and the settings imposing the time limit of one hour on run time per instance. Notice that both formulations use the same amount of information (cliques found in the conflict graph), only expressed in terms of different decision variables. From the results reported in Table C.3, it seems that with the exception of a single random instance (rand16) and one heavily constrained real-life instance (udine4), the proposed formulation performs considerably better.

Next, we compared the performance of the formulations of Udine Course Timetabling, differing only in the formulation of the underlying graph colour-

[2]Available at http://mat.gsia.cmu.edu/COLOR/ (May 25, 2011)
[3]Available at http://cs.nott.ac.uk/~jxm/timetabling/generator/ (May 25, 2011)
[4]Available at http://cs.nott.ac.uk/~jxm/colouring/supernodal/ (May 25, 2011)

ing component. Notice that the CPLEX run time necessary to reach optimality was two orders of magnitude higher than in the previous experiment looking for feasible colouring. Whether the performance gains observed in the graph colouring component alone would still be manifested, was thus not clear. As is summarised in Table C.4, however, the new formulation again seems to perform considerably better, reducing CPLEX run times approximately by factor of four, where it is possible to reach optimum within one hour using both formulations.

We have also studied the effects of symmetry breaking implemented in CPLEX on the performance of both formulations. In all previous experiments, both formulations were run using no built-in symmetry breaking in CPLEX. Table C.5 compares these results (denoted -SB) with results obtained with symmetry breaking built-in in CPLEX 10.0 set to aggressive (denoted +SB). Again, the new formulation using no built-in symmetry breaking performs better than the standard formulation using aggressive built-in symmetry breaking.

These results are rather encouraging, although the performance gains are limited only to graphs, where it is possible to obtain a reversible clique partition of V, whose cardinality is considerably less than $|V|$. This is not the case in triangle-free graphs and many dense random graphs, often used in benchmarking general graph colouring procedures. In many real-world applications, the graphs seem to be, however, highly structured, and the structure is worth exploiting.

Non-Linear Reformulation In order to illustrate the performance of the semidefinite programming relaxations, we report the performance of the bound for l-Bounded k-Colouring (Problem 7.7) on two data sets:

1. Conflict graphs from two timetabling benchmarks: From instances used in Track 3 of the International Timetabling Competition 2007 [76, 90], we have extracted course-based conflict graphs, where there is an edge between two vertices, if there is a curriculum prescribing the enrollment in both of the corresponding courses, or if a single teacher teaches both courses.

2. Knesser graphs of Lovász [292] and the "forbidden intersections" graphs of Frankl and Rödl [168]: Knesser graph $K(n,k)$, $n > k > 1$, has $\binom{n}{k}$ vertices, corresponding subsets of $\{1,2,\ldots,n\}$ of cardinality k. Two vertices are adjacent if the corresponding subsets are disjoint. Lovász has shown [292] that the chromatic number of $K(n,k)$ is exactly $n-2k+$

2, despite the fact $K(n,k)$ that has no triangle for $n > 3k$. Similarly, forbidden intersections graph $F(m,\gamma)$, $m \geq 1, 0 < \gamma < 1$, such that $(1-\gamma)m$ is an even integer, has 2^m vertices, corresponding to sequences of m bits (zeros and ones). Two vertices are adjacent, if the corresponding sequences differ in precisely $(1-\gamma)m$ bits.

The focus on l-BOUNDED k-COLOURING makes the data easier to interpret. However, one can use arbitrary other additional constraints.

First, on conflict graphs from timetabling applications, we show the strength of the bounds on real-life instances. We have computed the best possible vertex colouring, without any bound on the number of uses of a colour, and took the size of the largest colour class to be C. Subsequently, we have obtained lower bounds, upper bounds, and optima for $(C-1)$-bounded colouring, $(C-2)$-bounded colouring, etc. Table C.6 lists the results.

Next, on "forbidden intersections" [101] and Knesser graphs [258], we show the weakness of the bound in some cases. It is known that the theta bound of Lovász and related semidefinite programming relaxations of graph colouring perform poorly: the lower bound is $O(1)$ as n grows, whereas the actual chromatic number grows with n. Table C.7 shows that the lower bound gets tighter as the bound on the number of uses of a colour gets tighter. It should be noted that there is a large difference between clique and chromatic numbers in both Knesser and forbidden intersection graphs, which makes them quite unlike conflict graphs encountered in timetabling applications. Although semidefinite programming lower bounds for graph colouring are weak on these graphs, they do tighten, as the bound on the number of uses of colours tightens.

Finally, let us remark that an important aspect of implementing the augmented Lagrangian method is problem-specific simplification of the linear algebra involved. In relaxations of l-BOUNDED k-COLOURING (Problem 7.7) of a graph on n vertices and m edges, one can:

- avoid computing $(A_1 A_1^T)^{-1}$ and compute $A_1^T y_1$ in time $O(m)$
- compute $(A_2 A_2^T)^{-1}$ in time $O(n^2)$ and compute $A_2^T y_2$ in time $O(n)$
- compute $(AB^T)^{-1}$ in time $O(n)$ and compute $B^T v$ in time $O(n)$
- evaluate the augmented Lagrangian and its gradient at a given v in time $O(n^2)$.

This allows for an efficient implementation of `AugmentedLagrangianMethod` (Algorithm 7) using a variant of limited memory BGFS seach [163] with projection to non-negative v to minimise the quadratic program on Line 9. The bulk of the run-time is hence spent in an eigenvalue decomposition in Line 10, where our particular implementation relies on Intel Math Kernels implementation of the Lanczos algorithm [163], although much faster methods [224] have been proposed recently. The convergence of our implementation is currently too frail for us to present any results.

Chapter 8

Precedence-Constrained Components

Nothing will be more central to computational science in the next century than the art of transforming a problem that appears intractable into another whose solution can be approximated rapidly.

Nick Trefethen and David Bau [394]

It has been suggested (p. 93) that scheduling problems are defined by having either mutual-exclusion constraints or precedence constraints or both. Following a chapteron mutual-exclusion constraints and preceding *Future Work*, this chapterpresents our work in progress on precedence-constrained problems.

8.1 Scheduling, Packing, and Integer Programming: An Overview

Let us consider:

Problem 8.1. The Precedence-Constrained Scheduling (PCS): Given integers $r, n \geq 1$, amounts $a_i > 1$ of resources $i = 1, 2, \ldots, r$ available, resource requirements of n jobs $D \in \mathbb{R}^{n \times r}$, and p pairs of numbers $P \subseteq \{(i, j) \mid 1 \leq i < j \leq n\}$ expressing job i should be executed prior to executing j, find the largest integer k so that k jobs can be executed using the resources available.

This problem on its own has numerous important applications, notably in resource extraction [54, 318], where it is known as the open-pit mine production scheduling problem. It also underlies many other applications, such as realistic models of transportation, which consider both weight and volume of

vehicle load [181, 294]. The usual compact linear programming formulations do, however, provide particularly weak lower bounds. One can strengthen the bound, but even sophisticated problem-specific cuts used in conjunction with general-purpose cuts in-built in modern solvers do not result in particularly well-performing solvers. We, hence, present a formulation with a pseudo-polynomial number of variables, which provides strong bounds, and an adaptive discretisation method for obtaining it. We do not, however, present automated methods for extraction of the precedence-constrained component.

Clearly, there is a packing component to Precedence-Constrained Scheduling (Problem 8.1). An important special case in dimension three with rotations around combinations of axes in multiples of 90 degrees is still NP-Hard even to approximate [105]. Let us fix the order of six such allowable rotations in dimension three arbitrarily and define:

Problem 8.2. The CONTAINER LOADING PROBLEM (CLP): Given dimensions of a large box ("container") $x, y, z > 0$ and dimensions of n small boxes $D \in \mathbb{R}^{n\times 3}$ with associated values $w \in \mathbb{R}^n$, and specification of the allowed rotations $r = \{0,1\}^{n\times 6}$, find the greatest $k \in \mathbb{R}$ such that there is a packing of small boxes $I \subseteq \{1, 2, \ldots, n\}$ into the container with value $k = \sum_{i\in I} w_i$. The packed small boxes I may be rotated in any of the allowed ways, must not overlap, and no vertex can be outside of the container.

Problem 8.3. The VAN LOADING PROBLEM (VLP): Given dimensions of a large box ("van") $x, y, z > 0$, maximum mass $p \geq 0$ it can hold ("payload"), dimensions of n small boxes $D \in \mathbb{R}^{n\times 3}$ with associated values $w \in \mathbb{R}^n$, mass $m \in \mathbb{R}^n$, and specification of the allowed rotations $r = \{0,1\}^{n\times 6}$, find the greatest $k \in \mathbb{R}$ such that there is a packing of small boxes $I \subseteq \{1, 2, \ldots, n\}$ into the container with value $k = \sum_{i\in I} w_i$ and mass $\sum_{i\in I} m_i \leq p$. The packed small boxes I may be rotated in any of the allowed ways, must not overlap, and no vertex can be outside of the container.

Although there are a number of excellent heuristic solvers, the progress in exact solvers for the Container Loading Problem has been limited, so far.

The Formulation of Chen/Padberg Chen et al. [102] introduced an integer linear programming formulation using the relative placement indicator:

Symbol	Meaning
n	The number of boxes.
H	A fixed axis, in the set $\{X, Y, Z\}$.
α	An axis of a box, in the set $\{1, 2, 3\}$.
$L_{\alpha i}$	The length of axis α of box i.
$l_{\alpha i}$	The length of axis α of box i halved.
D_H	The length of axis H of the container.
w_i	The volume of box i in the CLP.

Table 8.1: Notation used in this chapter.

$$\lambda_{ij}^H = \begin{cases} 1 & \text{if box } i \text{ precedes box } j \text{ along axis } H \\ 0 & \text{otherwise} \end{cases},$$

$$\delta_{\alpha i}^H = \begin{cases} 1 & \text{if box } i \text{ is rotated so that axis } \alpha \text{ is parallel to fixed } H \\ 0 & \text{otherwise} \end{cases},$$

$$x_i^H = \text{ absolute position of box } i \text{ along axis } H.$$

$$\max \sum_{i=1}^{n} \sum_{H} w_i \delta_{1i}^H \tag{8.1}$$

$$\text{s.t.} \quad \sum_{H} \delta_{2i}^H = \sum_{H} \delta_{1i}^H \tag{8.2}$$

$$\sum_{H} \delta_{1i}^H = \sum_{\alpha} \delta_{\alpha i}^H \tag{8.3}$$

$$L_{1j(i)} \lambda_{j(i)i}^H + \sum_{\alpha} l_{\alpha i} \delta_{\alpha i}^H \leq x_i^H \tag{8.4}$$

$$x_i^H \leq \sum_{\alpha} (D_H - l_{\alpha i}) \delta_{\alpha i}^H - L_{1j(i)} \lambda_{ij(i)}^H \tag{8.5}$$

$$D_H \lambda_{ji}^H + \sum_{\alpha} l_{\alpha i} \delta_{\alpha i}^H - \sum_{\alpha} (D_H - l_{\alpha j}) \delta_{\alpha j}^H \leq x_i^H - x_j^H \tag{8.6}$$

$$x_i^H - x_j^H \leq \sum_{\alpha} (D_H - l_{\alpha i}) \delta_{\alpha i}^H - \sum_{\alpha} l_{\alpha j} \delta_{\alpha j}^H - D_H \lambda_{ij}^H \tag{8.7}$$

$$\sum_{H} (\lambda_{ij}^H + \lambda_{ji}^H) \leq \sum_{H} \delta_{1i}^H \tag{8.8}$$

$$\sum_{H} (\lambda_{ij}^H + \lambda_{ji}^H) \leq \sum_{H} \delta_{1j}^H \tag{8.9}$$

$$\sum_{H} \delta_{1i}^H + \sum_{H} \delta_{1j}^H \leq 1 + \sum_{H} (\lambda_{ij}^H + \lambda_{ji}^H) \tag{8.10}$$

$$\sum_{i=1}^{n} \sum_{H} \left(\prod_{\alpha} L_{\alpha i} \right) \delta_{1i}^H \leq \prod_{H} D_H \tag{8.11}$$

$$\delta_{\alpha i}^H \in \{0,1\}, \lambda_{ij}^H \in \{0,1\}$$

$$L_{1i} \leq L_{2i} \leq L_{3i}, j(i) \text{ such that } L_{1j(i)} = \max\{L_{1j}\} \text{ for } 1 \leq i \neq j \leq n.$$

Fasano [153] suggested minor improvements to the formulation. Padberg [334] has studied its properties. In particular, he identified the subsets of constraints with the integer property. Despite the interesting theoretical properties, modern integer programming solvers fail to solve instances larger than 10–20 boxes using this formulation, as evidenced by Table C.8. This is far from satisfactory.

Extensions to the Formulation of Chen/Padberg Arguably, the formulation could be strengthened further by the addition of valid constraints.

We have identified three classes of such constraints. Compound constraints stop combinations of certain boxes being placed next to each other if they would violate the domain constraint. For example:

$$\delta^H_{ki} + \delta^H_{mj} + \lambda^H_{ij} \leq 2 \quad \text{if } l_{ki} + l_{mj} > D_H \quad \forall k, m \in \{1,2,3\}, H \in \{X,Y,Z\} \\ \text{and } 1 \leq i \neq j \leq n \tag{8.12}$$

The example for 2 boxes can be easily extended to 3 or more boxes "in a row." One can also attempt to break symmetries in the problem. If boxes i and j (where $i < j$) are identical (i.e. $L_{\alpha i} = L_{\alpha j}\ \forall \alpha$):

$$\lambda^H_{ij} = 0 \text{ and } \sum_H \delta^H_{1i} \geq \sum_H \delta^H_{1j} \tag{8.13}$$

Furthermore, if a box has two or more sides of the same length then we can limit the rotations. We define function f, which provides a canonical mapping of $\{X,Y,Z\}$ to $\{1,2,3\}$:

$$\sum_H (f(H) \cdot \delta^H_{ki}) \geq \sum_H (f(H) \cdot \delta^H_{mi}) \quad \text{if } l_{ki} = l_{mi} \\ \forall 1 \leq i \leq n \text{ and } 1 \leq k < m \leq 3 \tag{8.14}$$

Nevertheless, whilst the addition of these constraints improves the performance somewhat, the formulation remains impractical.

8.2 Finding the Precedence-Constrained Component

In order to reformulate the problem automatically, the solver needs to identify the precedence-constrained part. In previous sections, we have looked for a maximum component, corresponding to an arbitrary submatrix. In this section, we focus on a maximum row-block, which in the definition on page 76 is the submatrix whose rows' index set is a complete integer subsequence.

8.2.1 The Problem and its Complexity

Let us state the problem of extracting the Chen/Padberg relaxation formally:

Problem 8.4. Precedence-Constrained Component/Row-Block Extraction: Given positive integers d, m_1, m_2, an $m_1 \times d$ integer matrix A_1, an $m_2 \times d$ integer matrix A_2, an m_1-vector b_1 of integers, and an m_2-vector b_2

of integers, corresponding to a mixed integer linear program with constraints $A_1 y = b_1$, $A_2 y \leq b_2$, find the largest integer n ("the maximum number of boxes"), such that:

- there exists a $4n \times 9n$ submatrix/row-block E of A_1, corresponding only to binary variables, which we denote δ, with zero coefficients elsewhere in the rows
- there exists a $9n(n-1)/2+6n+1 \times 3n(n-1)/2+12n$ submatrix/row-block F of A_2, corresponding to $9n$ binary variables δ as before, $3n(n-1)/2$ binary variables denoted λ, and $3n$ continuous variables denoted x, with zero coefficients elsewhere in the rows
- E contains n rows with exactly 2 non-zero coefficients ± 1, corresponding to (8.2), and 0 in the right-hand side b_1
- E contains $3n$ rows with exactly 4 non-zero coefficients ± 1, corresponding to (8.3), and 0 in the right-hand side b_1
- F contains $6n$ rows with exactly 5 non-zero coefficients, some not necessarily ± 1, corresponding to (8.4–8.5), and 0 in the right-hand side b_2
- F contains $6n(n-1)$ rows with exactly 9 non-zero coefficients, some not necessarily ± 1, corresponding to (8.6–8.7), and 0 in the right-hand side b_2
- F contains $2n(n-1)$ rows with exactly 9 non-zero coefficients ± 1, corresponding to (8.8–8.9), and 0 in the right-hand side b_2
- F contains $n(n-1)$ rows with exactly 12 non-zero coefficients ± 1, corresponding to (8.10), and 1 in the right-hand side b_2
- F contains 1 rows with exactly $3n$ non-zero coefficients, not necessarily ± 1, corresponding to (8.11), and a positive number in the right-hand side b_2.

Notice that by maximising the number of rows involved, we also maximise the number of boxes, as number $r = 9n(n-1)/2+6n+1$ of rows is determined by number n of boxes. The following can be seen easily:

Lemma 8.5. PRECEDENCE-CONSTRAINED COMPONENT EXTRACTION is in $\mathcal{NP}$.

Conjecture 8.6. Precedence-Constrained Component Extraction is $\mathcal{NP}$-Hard.

Lemma 8.7. Precedence-Constrained Row-Block Extraction is in $\mathcal{P}$.

Proof. A polynomial-time algorithm for extracting the precedence-constrained block can clearly rely on there being a polynomial number of blocks of the required size. See Algorithm 9. □

8.2.2 Algorithms

Algorithm 9 displays a very general algorithm schema for Precedence-Constrained Row-Block Extraction. Notably, the test of Line 10 requires elaboration. First, one needs to partition the block into the 5 families of rows (8.4–8.11). Some rows (8.8–8.9, 8.4–8.5, 8.11) are clearly determined by the numbers of non-zeros (9, 5, and 3). One can distinguish between others (8.6–8.7 and 8.10) by their right-hand sides. The test as to whether the rows represent the constraints (8.4–8.11) is based on identifying the variables. Continuous variables x are, however, identified easily and binary variables δ are determined in Line 6. What remains are variables λ.

8.3 Exploiting the Precedence-Constrained Component

8.3.1 A Space-Indexed Reformulation

We propose a reformulation, where the large box is discretised into units of space, whose dimensions are given by the largest common denominator of the respective dimensions of the small boxes.[1] The small boxes are grouped by their dimensions into types $t \in \{1, 2, \ldots, n\}$. A_t is the number of boxes of type t available. The formulation uses the space-indexed binary variable:

[1]This work has been first presented at the 7th ESICUP Meeting in Buenos Aires, Argentina, June 6–9, 2010. Subsequently, the approach has become quite popular [255, 136].

Algorithm 9 `PrecedenceConstrainedBlock`(A_1, b_1, A_2, b_2)

Input: $A_1 x = 1, A_2 x \leq 1$, that is $m_1 \times d$ matrix A_1 and $m_2 \times d$ matrix A_2, m_1-vector b_1, m_2-vector b_2
Output: Integer k and blocks E, F of A_1, A_2
Set $k_{\max}$ to the largest integer k such that there are k subsequent rows in A_1 with exactly 6 non-zero elements, all ± 1
for integer $k = 4n$ from $k_{\max}$ down to 4 **do**
  for $4n \times 9n$ block E in A_1 such that all rows have 6 non-zeros ± 1 **do**
    if E cannot represent constraints (8.2–8.3) **then**
      Continue
    end if
    for $7n + 9n(n-1)/2 \times 3n^2 + 9n$ block F in A_2 **do**
      if F cannot be partitioned into (8.4–8.11) **then**
        Continue
      end if
      return n, E, F
    end for
  end for
end for

$$\mu^t_{x,y,z} = \begin{cases} 1 & \text{if a box of type } t \text{ is placed such that its lower-back-left} \\ & \text{vertex is at coordinates } x, y, z \\ 0 & \text{otherwise} \end{cases}$$

Without allowing for rotations, the formulation reads as follows:

$$\max \sum_{x,y,z,t} \mu^t_{xyz} w_t \tag{8.15}$$

$$\text{s.t.} \sum_t \mu^t_{xyz} \leq 1 \quad \forall x, y, z \tag{8.16}$$

$$\mu^t_{xyz} = 0 \quad \forall x, y, z, t \text{ where } x + L_{1t} > D_X \text{ or } y + L_{2t} > D_Y \text{ or } z + L_{3t} > D_Z \tag{8.17}$$

$$\sum_{x,y,z} \mu^t_{xyz} \leq A_t \quad \forall t \tag{8.18}$$

$$\sum_{x',y',z'} f(x, y, z, x', y', z', t) \leq 1 \quad \forall x, y, z, t \tag{8.19}$$

where

$$f(x,y,z,x',y',z',t) = \begin{cases} \mu^t_{x'y'z'} & \text{if } x \leq x' \leq x + L_{1t} \text{ and} \\ & \quad y \leq y' \leq y + L_{2t} \text{ and} \\ & \quad z \leq z' \leq z + L_{3t} \\ 0 & \text{otherwise} \end{cases} \tag{8.20}$$

The constraints are very natural: No region in space may be occupied by more than one box type (8.16), boxes must be fully contained within the container (8.17), there may not be more than A_t boxes of type t (8.18), and boxes cannot overlap (8.19). There is one non-overlapping constraint (8.19) for each discretised unit of space and type of box. In order to support rotations, new box types need to be generated for each allowed rotation and linked via set packing constraints, which are similar to Constraint 8.18. In order to extend the formulation to the van loading problem, it suffices to add the payload capacity constraint:

$$\sum_{x,y,z,t} \mu^t_{xyz} m_t \leq p \tag{8.21}$$

8.3.2 Adaptive Discretisation

In order to reduce the number of regions of space, and thus the number of variables in the formulation, a sensible space-discretisation method should be employed. The greatest common divisor (GCD) reduction can be applied on a per-axis basis, finding the greatest common divisor between the length of the container for an axis and all the valid lengths of boxes that can be aligned along that axis and scaling by the inverse of the GCD. This is trivial to do and is useful when all lengths are multiples of a large number, which may be common in certain situations.

In other situations, this may not reduce the number of variables at all, and it may be worth tackling the optimisation variant of:

Problem 8.8. The Discretisation Decision: Given integer k, dimensions of a large box ("container") $x, y, z > 0$ and dimensions of n small boxes $D \in \mathbb{R}^{n\times 3}$ with associated values $w \in \mathbb{R}^n$, and specification $r = \{0,1\}^{n\times 6}$ of what rotations are allowed, decide if there are k points, where the lower-bottom-left vertex of each box can be positioned in any optimum solution of Container Loading Problem. Formally, $k = |I_X||I_Y||I_Z|$, where index-

sets I_X, I_Y, I_Z refer to the set $\{1c, 2c, \ldots, \max\{x, y, z\}c^{-1}\}$, where c is the greatest common divisor across elements in D.

Considering the following:

Problem 8.9 (Papadimitriou [339])**.** The decision whether the optimum of an instance of KNAPSACK is unique is Δ_2^p-Complete, where Δ_2^p is the class of problems that can be solved in polynomial time using oracles from $\mathcal{NP}$.

Theorem 8.10. DISCRETISATION DECISION is Δ_2^p-Hard.

Proof. One could check for the uniqueness of the optimum of an instance of KNAPSACK (Problem 2.2) using any algorithm for DISCRETISATION DECISION (Problem 8.8). □

We can, however, use non-trivial non-linear space-discretisation heuristics. To do this we use the same values as before on a per-axis basis, i.e. the lengths of any box sides that can be aligned along the axis. We then use dynamic programming to generate all valid locations for a box to be placed. See Algorithm 10. For example, given an axis of length 10 and box lengths of 3, 4 and 6, we can place boxes at positions 0, 3, 4, 6, and 7. 8, 9 and 10 are also possible, but no length is small enough to still lie within the container if placed at these points. This has reduced the number of regions along that axis from 10 to 5. An improvement on this scale may not be particularly common in practice, though the approach can help where the GCD is 1. It is obvious that this approach can be no worse than the GCD method at discretising the container. This also adds some implicit symmetry breaking into the model. Notice that the algorithm runs in time polynomial in the size of the output it produces, but this is may be exponential in the size of the input and considerably more than the size of the best possible output.

8.3.3 Computational Experience

We have tested the formulations above on three sets of instances:

- 3D Pigeon Hole Problem instances, Pigeon-n, where $n + 1$ unit cubes are to be packed into a container of dimensions $(1 + \epsilon) \times (1 + \epsilon) \times n$.
- SA and SAX datasets, which are used to test the dependence of solvers' performance on parameters of the instances, notably the number of

Algorithm 10 `DiscretisationDP`(M, n, D, r)

Input: Dimensions $M \in \mathbb{R}^3$ of the container, dimensions of n small boxes $D \in \mathbb{R}^{n\times 3}$, and allowed rotations $r = \{0,1\}^{n\times 6}$
Output: Integer k and k possible positions
$P = \emptyset$
for axis with limit $m \in M$ **do**
  $L = \{d \mid d \in D$ may appear along this axis, given allowed rotations $r\}$
  $P =$ closure of $P \cup \{p + l \mid p \in P, l \in L, p + l \leq m\}$,
  optionally pruning $p \in P$ that cannot occur due to the dependence of the axes and the fact each box can be packed at most once
end for
return $|P|, P$

boxes, heterogeneity of the boxes, and physical dimensions of the container. There is 1 pseudo-randomly generated instance for every combination of container sizes ranging from 5–100 in steps of 5 units cubed and the number of boxes to pack ranging from 5–100 in steps of 5. The SA datasets are perfectly packable, i.e. are guaranteed to be possible to load the container with 100% utilisation with all boxes packed. The SAX are similar but have no such guarantees; the summed volume of the boxes is greater than that of the container.

- Van-loading instances, generated so as to be similar to those used in pricing subproblems of branch-and-cut-and-price approaches to vehicle routing with both load weight and volume considerations [181, 294]. The van-loading and van-loading-X instances use containers of $10\times10\times30$ and $10\times10\times50$ units, respectively, representing the approximate ratios of a small commercial van and a larger freight truck. 2, 4 or 6 out of 6 pseudo-randomly chosen orientations are allowed. The load weights and knapsack values are generated pseudo-randomly and independently of the volume of the small boxes. 10 instances were generated for each class of problem.

All the tests were performed on a 64-bit computer running Linux, which was equipped with 2 quad-core processors (Intel Xeon E5472) and 16 GB memory. The solvers tested were IBM ILOG CPLEX 12.2.0, Gurobi Solver 4.0, and SCIP 2.0.1 [9] with CLP as the linear programming solver. The

instances are available[2] on-line.

For the 3D Pigeon Hole Problem, results obtained within one hour using the three solvers and the Chen/Padberg formulation are shown in Table C.8, while Table C.9 compares the results of Gurobi Solver on both formulations. None of the solvers managed to prove optimality of the incumbent solution for twelve unit-cubes within an hour using the Chen/Padberg formulation, although the instance of linear programming (after pre-solve) had only 616 rows and 253 columns and 4268 non-zeros. In contrast, the space-indexed formulation allowed Gurobi Solver to solve Pigeon-10000000 within an hour, where the instance of linear programming had 10000002 rows, 10000000 columns, and 30000000 non-zeros.

For the SA and SAX datasets, Figures C.1–C.2 summarise solutions obtained within an hour using the Chen/Padberg formulation, the space-indexed formulation, and a metaheuristic approach described by Allen et al. [18]. In the case of the SAX dataset, the volume utilisation is given with respect to the tightest upper bound found by any of the solvers. Finally, for the van-loading instances, the results of Gurobi Solver with the space-indexed model after one hour can be seen in Table C.10. As there were 10 instances tested for each class of problem, the mean number of boxes, number of unique box types and mean gap between solution value and bound are also given.

Overall, the space-indexed relaxation provides a particularly strong upper bound. The mean integrality gap, or the ratio of the difference between root linear programming relaxation value and optimum to optimum has been 10.49 % and 0.37 % for the Chen/Padberg and the space-indexed formulation, respectively, on the SA and SAX instances solved to optimality within the time limit of one hour. The mean gap, or the ratio of the difference between the linear programming relaxation value and the value of the best solution found to the bound, left after an hour on van-loading instances was 1.5%. This should not be particularly surprising, as similar discretised relaxations proved to be very strong in scheduling problems corresponding to one-dimensional packing [384, 399, 336] and can be shown to be asymptotically optimal for various geometric problems both in two dimensions [338] and in higher [421] dimensions. It, indeed, seems that adaptive discretisations, pioneered by time-indexed formulations previously and space-space formulation in this section, seem to be the technique for deriving strong relaxations for precedence-constrained problems.

[2] Available at http://discretisation.sf.net (May 25, 2011)

Chapter 9

Conclusions

General-purpose integer linear programming solvers has seen only incremental, if very impressive [60, 64, 56], improvements in the past two decades. Many original "alternative" approaches have been developed, but a number of those suffer from two major flaws: firstly, they are not compatible with the branch-and-cut approach employed by contemporary commercial general-purpose solvers and, hence, are only rarely implemented. Secondly, they are neither generally applicable, nor designed specifically to tackle classes of instances that are difficult to solve using the branch-and-cut approach, such as those in Scheduling and Timetabling. It may hence be worth investigating further alternatives.

This dissertation has presented an approach to designing methods, which could be implemented in existing branch-and-cut solvers, but which improve their currently mediocre performance on a number of classes of instances. It attempts to find and exploit "useful structures". Three such useful structures have been described, which are both pervasive in random models and widespread in Scheduling and Timetabling applications. Solvers exploiting two of these structures [88, 19] constitute the state of the art in two particular areas in Scheduling and Timetabling. Nevertheless, there are a number of questions left open.

9.1 Open Questions

The ultimate question is, indeed, what effect would the exploitation of useful structures have on the performance of a state-of-the-art solver, such as the next release of IBM/ILOG CPLEX. This thesis has made only the first steps towards answering it. In doing so, a number of further questions have appeared.

In Mathematics and Theoretical Computer Science:

- What structures can be made useful? Table 2.2 lists a number of structures. Some of them, notably basis reduction methods, seem "useful", but are they?

- Is there a (precise) sense in which the (primal, improvement, branching, separation) sub-problems are easier than the integer linear programming proper, despite the results of Schulz [375]? Perhaps some "approximation ratio in expectation" in the spirit of smooth analysis?

- Can one efficiently compute decompositions of hypergraphs or matroids, whose expected width measures in some model of random graphs were sublinear? Notably, can one obtain booleanwidth decompositions [13, 85, 43] in polynomial-time? See Section 3.3.1.

- Empirically, strong branching performs particularly well. Can its performance be bounded? Based on the growing body of analyses of the use of inexact solutions in exact methods [362], it seems to.

- Is there a polynomial-time algorithm for finding the closure of TANGLED MIN-MAX REMOVAL and the closure of ABS-VAL REMOVAL, MIN-MAX REMOVAL, and TANGLED MIN-MAX REMOVAL? See Definition 6.9 in Section 6.4.

- What are the convergence properties of multiple-splitting schemes in augmented Lagrangian methods? Notably, does Conjecture 7.22 hold?

- Can one optimise over the super-exponentially large instances of semidefinite programming obtained using the known operators [293, 278, 221] from instances of linear programming in time polynomial in the size of the original instances of linear programming? See Section 2.5.3.

- What is the number of $m \times n, \{-1, 0, 1\}$ matrices, which are totally unimodular and distinct up to the permutations of rows and columns? Notably, can one provide a closed-form solution to the recursion given by Proposition 4.3, following constructions (C1–C6)?

- What is the expected size of the largest submatrix in a constraint matrix of a random integer program, which corresponds to the instance of propositional satisfiability with the neighbourhood structure that Lovász local lemma [319] guarantees it to be feasible?

- What is the expected size of the largest totally unimodular submatrix in an $m \times n$ matrix, whose entries are independently equiprobably drawn from the set $\{0, \pm 1\}$? Notably, does Conjecture 4.14 hold?

- Can one design algorithms for the extraction of a large totally unimodular submatrix with non-trivial approximation guarantees, based on the characterisation of Seymour [378, 374]? See Section 4.2.1.

In the development of integer programming solvers:

- Can tree decompositions be used in a "useful structure"?
- Pataki et al. [341] provide an exquisite analysis of the complexity of branch and bound, using basis reduction. Hence, can basis reduction be made into a "useful structure"?
- Can one introduce specialised column-generation or interior-point methods for solving linear programming relaxations obtained by discretisation, such as those of Section 8? The Dantzig-Wolfe decomposition could, perhaps, divide the problem into sub-problems pertaining to each type of box (8.19) and a master problem enforcing the set-packing constraint (8.16). To what extent can combinatorial algorithms [336] speed up the solution of discretised (time-index and space-indexed) relaxations?
- In many minimisation problems in combinatorial optimisation, the optimum is rather a small number k and it is easy to produce feasible solutions with value $k + 1$ as well as to infer integer infeasibility of solutions with value $k - 1$, which could perhaps be exploited.
- For problems where the optimal value of the objective function is higher, it could be possible to apply branching on the components of the objective function. What would be the performance of such heuristics?
- What is the performance of branching heuristics derived from colouring the conflict graph, such as: (i) Least constraint colour to most constrained node (ii) "Where local search for k-colouring gets stuck" (iii) "Where iterated local search gets stuck most often" (iv) Shortest path sampled in a Zykovian merge-separate tree (v) Shortest path in a Zykovian merge-separate tree?
- What is the empirical performance of the algorithms for recognition of totally unimodular matrices, based on the characterisation of Seymour [378, 374]. See Section 4.2.1.

Appendix A

The Preservation of Structure

Do only what only you can do.

Edsger Wybe Dijkstra

In 2008–2009, I spent considerable amounts of time working along the following lines: An integer programming solver should exploit structure present in the instance. Some of the most successful means of exploitation use reformulation. Unfortunately, the extraction of the structure is as hard as solving the original problem, and finding the best reformulation of an algebraic model seems undecidable with respect to many criteria. Hence, let the user describe the structure and suggest some possible reformulations. This line of work, inspired by Parkes [340] and Gondzio [116], has resulted in a parser of an optimisation programming language with support for semi-automated reformulations. An overview is presented for the record.

An Overview The progress in the development of integer linear programming (ILP) solvers makes it possible to solve more and more complex problems. With the complexity of the problem, the number of possible formulations ("models") grows combinatorially: Shall we introduce this aggregate variable? How do we cope with the trade-off between the large number and limited strength of constraints? There seem to be, however, little or no tools to aid the modeller.

There is a long history of work on formalisation of mathematical modelling, including the seminal papers of [183]. See the survey of [209]. Many concepts presented by Geoffrion, such as algebraic models ("model schemata") and integration with databases, are now a common place. Widely used modelling languages, including OPL [400], AMPL [167], and ZIMPL [268] do not, however, have support for model management or semi-automated model tuning, and only ZIMPL comes with an open-source parser.

Plosar We present a programming language for optimisation with support for semi-automated reformulations (abbreviated "Plosar"). The language is not created from scratch, but extends the concept of separation of model and data and the familiar syntax of Van Hentenryck [400]. The extension consists of the keywords `optional`, `pickone`, and `or`. Using the new keywords, it is possible to formulate modelling alternatives.

We present an open-source parser, which allows for the generation of instances in the standard ".lp" format. Unlike many previous open-source parsers built upon Bison or Yacc, the parser uses Boost Spirit, which allows for better error reporting and easier debugging. Further applications of the parser include the translation of algebraic models to AMS LaTeX and C++ source code using ILOG Concert libraries. Although trivial in terms of mathematical content, we believe these tools could be of interest to the community.

We also present a tool for "semi-automated discovery of reformulations". It allows for the search space of modelling alternatives to be searched exhaustively, with the evaluation of a model based on a number of data sets ("a benchmark") and various statistical measures. This could certainly be extended to make use of local search or experiment design.

Appendix B

The Exploitation of Unknown Structures

When we write programs that "learn", it turns out we do and they don't.

Alan J. Perlis

In 2007–2008, I spent considerable amounts of time working along the following lines: An integer programming solver is a bag of tricks. Unfortunately, we do not know what trick works when. Let us use machine learning. This work, in part inspired by a suggestion of Nemhauser [323], has not been finished and has been superseded by Kılınç-Karzan et al. [264], Hutter et al. [247, 245, 246], and the staff of Fair Isaac Corporation. An overview is presented for the record.

Auto-Tuning: An Overview The exploitation of unknown structures is closely related with the idea of "auto-tuning". There seem to be four or more commonly used meanings of the word "auto-tuning":

- Pure "off-line" learning: What settings have the best mean of a performance indicator on a given set of instances?
- "Off-line" learning of static rules using "instance attributes": What settings have the best mean of a performance indicator on a given set of instances and a given set of attributes of each instance and attributes of the instance to solve?
- Pure "on-line" learning: What settings have the best expected performance, with respect to a fixed indicator, based on their performance so far on this particular instance?
- Hybrid learning based on "instance attributes" and "node attributes": What settings have the best expected performance, with respect to a

fixed indicator, based on the performance of a given set of instances with a given set of attributes of each instance and "the progress so far" on the particular instance with its attributes?

It seems clear that pure "off-line" learning can be accomplised by an algorithm running in time given by the product of the number of parameter settings, the number of instances, and the per-instance run-time. This is clearly in class $P^{\mathcal{NP}}$. "Off-line" learning of static rules using "instance attributes" for various notions of a rule and conditions on its sparsity can be shown to be in $\mathcal{NP}^{\mathcal{NP}}$. Pure "on-line" learning can be shown to be in $PSPACE^{\mathcal{NP}}$. Hybrid learning seems to be yet more difficult.

A considerable amount of work has focused on "off-line" learning of static rules using "instance attributes". One may want to learn rules such as:

- use branching rules based on inference, when the objective function is constant or seems artificial
- use branching on general disjunctions, if there are more than 25 % of general integer variables
- use strong branching, if Gaussian elimination performs well on the constraint matrix A

Such "rules" have often been discussed [122, 48, 8, e.g.] yet until recently, the integer programming community has not focused on automating their inference.

An Overview of the Unfinished Work Appropriate instance attributes seem to be:

- Ratio of rows to columns in the constraint matrix A
- Density of non-zeros in the constraint matrix A
- Percentage of binary variables
- Percentage of integral variables with variable upper bounds
- Percentage of non-integral variables
- Percentage of k-sparse ($k = 1, 2$) columns in A

- Percentage of columns constrained in one direction only in A
- Number and percentage of dense columns in A
- Average density of the top 10 densest columns in A
- Median number of variables per constraint
- Density of non-zeros in $A^T A$
- Number and percentage of dense columns in $A^T A$
- Average density of the top 10 densest columns in $A^T A$
- Percentage of constraints of type T (cover, packing, partition)
- Percentage of two-variable constraints of type T (cover, packing, partition)
- Number and percentage of dense rows
- Average density of the top 10 densest rows
- Ratio of the number of non-zeros in the original matrix to the number of non-zeros in the pre-solved matrix
- Rank of the quality of solution obtained by primal heuristic A at the root node
- Number of cuts of type T applied at the root node

Suitable node attributes seem to include:

- Depth of the node
- Number of nodes visited
- Number of nodes left
- Ratio of number of nodes left to number of nodes visited
- Rank of heuristic A in the counts of successful runs of primal heuristics so far
- Median rank of the run-times of successful runs of primal heuristic A so far

- Median rank of the run-times of runs of primal heuristic A so far
- Number of cuts of type A applied at so far

The "off-line" rules can be used to obtain initial estimates in an upper confidence bound policy.

Appendix C

The Computational Results

Table C.1: Dimensions of graphs induced by reversible clique partitions obtained from DIMACS instances (G), with (Q') and without (Q) pre-processing. Empty spaces indicate graphs that are trivial to colour.

Instance	Original Graph G		Rev. Cliq. Part. Q		Rev. Cliq. Part. Q'	
	Vert.	Edges	Vert.	Edges	Vert.	Edges
1-FullIns_3	30	100	29	89		
1-FullIns_4	93	593	92	561	25	85
1-FullIns_5	282	3247	281	3152	61	358
1-Insertions_4	67	232	67	232	60	208
1-Insertions_5	202	1227	202	1227	202	1227
1-Insertions_6	607	6337	607	6337	600	6301
2-FullIns_3	52	201	51	186		
2-FullIns_4	212	1621	211	1566	16	65
2-FullIns_5	852	12201	851	11986	93	582
2-Insertions_3	37	72	37	72		
2-Insertions_4	149	541	149	541	149	541
2-Insertions_5	597	3936	597	3936	597	3936
3-FullIns_3	80	346	79	327	17	65
3-FullIns_4	405	3524	404	3440	22	114
3-FullIns_5	2030	33751	2029	33342	94	768
3-Insertions_3	56	110	56	110		
3-Insertions_4	281	1046	281	1046	281	1046
3-Insertions_5	1406	9695	1406	9695	1395	9642
4-FullIns_3	114	541	113	518		
4-FullIns_4	690	6650	689	6531		
4-FullIns_5	4146	77305	4145	76610	195	1769
4-Insertions_3	79	156	79	156		
4-Insertions_4	475	1795	475	1795	475	1795
5-FullIns_3	154	792	153	765	39	229
5-FullIns_4	1085	11395	1084	11235	121	1037
abb313GPIA	1557	53356	1557	53356	853	16093

Table C.1: Dimensions of graphs induced by reversible clique partitions obtained from DIMACS instances. (Continued.)

Instance	Original Graph G		Rev. Cliq. Part. Q		Rev. Cliq. Part. Q'	
	Vert.	Edges	Vert.	Edges	Vert.	Edges
anna	138	493	125	437		
ash331GPIA	662	4181	662	4181	661	4180
ash608GPIA	1216	7844	1216	7844	1215	7843
ash958GPIA	1916	12506	1916	12506	1915	12505
david	87	406	74	322		
DSJC1000.1	1000	49629	1000	49629	1000	49629
DSJC1000.5	1000	249826	1000	249826	1000	249826
DSJC1000.9	1000	449449	1000	449449	1000	449449
DSJC125.1	125	736	125	736	125	736
DSJC125.5	125	3891	125	3891	125	3891
DSJC125.9	125	6961	125	6961	125	6961
DSJC250.1	250	3218	250	3218	250	3218
DSJC250.5	250	15668	250	15668	250	15668
DSJC250.9	250	27897	250	27897	250	27897
DSJC500.1	500	12458	500	12458	500	12458
DSJC500.5	500	62624	500	62624	500	62624
DSJC500.9	500	112437	500	112437	500	112437
DSJR500.1	500	3555	480	3341		
DSJR500.1c	500	121275	500	121275	281	38166
DSJR500.5	500	58862	497	58218	483	56618
ear	190	4793	185	4758	172	4636
fpsol2.i.1	496	11654	427	5108	107	2454
fpsol2.i.2	451	8691	395	5657	154	2705
fpsol2.i.3	425	8688	369	5658	153	2665
games120	120	638	119	629		
hec	81	1363	81	1363	75	1277
homer	561	1628	503	1376		
huck	74	301	54	179		
inithx.i.1	864	18707	732	11140		
inithx.i.2	645	13979	539	9317	50	544
inithx.i.3	621	13969	521	9427	49	474
jean	80	254	67	177		
latin_square_10	900	307350	900	307350	900	307350
le450_15a	450	8168	450	8168	407	7802
le450_15b	450	8169	450	8169	410	7824
le450_15c	450	16680	450	16680	450	16680

Table C.1: Dimensions of graphs induced by reversible clique partitions obtained from DIMACS instances. (Continued.)

Instance	Original Graph G		Rev. Cliq. Part. Q		Rev. Cliq. Part. Q'	
	Vert.	Edges	Vert.	Edges	Vert.	Edges
le450_15d	450	16750	450	16750	450	16750
le450_25a	450	8260	450	8260	264	5840
le450_25b	450	8263	450	8263	294	6240
le450_25c	450	17343	450	17343	435	17096
le450_25d	450	17425	450	17425	433	17106
le450_5a	450	5714	450	5714	450	5714
le450_5b	450	5734	450	5734	450	5734
le450_5c	450	9803	450	9803	450	9803
le450_5d	450	9757	450	9757	450	9757
miles1000	128	3216	123	3049		
miles1500	128	5198	104	3486		
miles250	128	387	117	341		
miles500	128	1170	115	1065		
miles750	128	2113	122	2011		
mug100_1	100	166	84	118		
mug100_25	100	166	83	115		
mug88_1	88	146	75	107		
mug88_25	88	146	72	98		
mulsol.i.1	197	3925	166	2274		
mulsol.i.2	188	3885	158	2458	35	337
mulsol.i.3	184	3916	155	2504	35	336
mulsol.i.4	185	3946	155	2504	36	360
mulsol.i.5	186	3973	157	2549	36	356
myciel2						
myciel3	11	20	11	20	11	20
myciel4	23	71	23	71	23	71
myciel5	47	236	47	236	47	236
myciel6	95	755	95	755	95	755
myciel7	191	2360	191	2360	191	2360
qg.order100	10000	990000	10000	990000	10000	990000
qg.order30	900	26100	900	26100	900	26100
qg.order40	1600	62400	1600	62400	1600	62400
qg.order60	3600	212400	3600	212400	3600	212400
queen10_10	100	1470	100	1470	100	1470
queen11_11	121	1980	121	1980	121	1980
queen12_12	144	2596	144	2596	144	2596

Table C.1: Dimensions of graphs induced by reversible clique partitions obtained from DIMACS instances. (Continued.)

Instance	Original Graph G		Rev. Cliq. Part. Q		Rev. Cliq. Part. Q'	
	Vert.	Edges	Vert.	Edges	Vert.	Edges
queen13_13	169	3328	169	3328	169	3328
queen14_14	196	4186	196	4186	196	4186
queen15_15	225	5180	225	5180	225	5180
queen16_16	256	6320	256	6320	256	6320
queen5_5	25	160	25	160	25	160
queen6_6	36	290	36	290	36	290
queen7_7	49	476	49	476	49	476
queen8_12	96	1368	96	1368	96	1368
queen8_8	64	728	64	728	64	728
queen9_9	81	1056	81	1056	81	1056
school1	385	19095	376	18937	353	18799
school1_nsh	352	14612	344	14486	322	14343
wap01a	2368	110871	1594	73666	1594	73666
wap02a	2464	111742	1594	72498	1594	72498
wap03a	4730	286722	3716	224640	3716	224640
wap04a	5231	294902	3814	221704	3814	221704
wap05a	905	43081	749	35116	746	35102
wap06a	947	43571	741	34012	735	33760
wap07a	1809	103368	1611	91746	1609	91698
wap08a	1870	104176	1628	91140	1627	91122
will199GPIA	701	6772	701	6772	660	5836
zeroin.i.1	211	4100	182	2131		
zeroin.i.2	211	3541	188	2187		
zeroin.i.3	206	3540	183	2186		

Table C.2: The dimensions of test instances: numbers of events, occupancy measured as the number of events divided by the number of available time-place slots, and dimensions of the constraint matrices produced by formulations of Udine Course Timetabling (variables × constraints, non-zeros in constaint matrix).

Instance	Ev.	Occ.	Standard	(Non-zero)	New	(Non-zero)
rand01	100	70%	15415 × 3194	469.35k	5398 × 4176	188.34k
rand02	100	70	15415 × 3197	508.63k	5398 × 4179	188.38k
rand03	100	70	15415 × 3197	522.44k	5398 × 4179	199.47k
rand04	200	70	60835 × 6447	2.03M	21002 × 8444	794.63k
rand05	200	70	60830 × 6416	1.94M	20696 × 8381	754.97k
rand06	200	70	60830 × 6417	2.16M	20696 × 8382	814.10k
rand07	300	70	136270 × 9799	4.29M	48174 × 12907	1.76M
rand08	300	70	136260 × 9729	4.19M	47262 × 12773	1.69M
rand09	300	70	136255 × 9698	4.46M	46806 × 12710	1.74M
rand11	100	80	12935 × 3296	356.88k	5097 × 4406	159.66k
rand12	100	80	12925 × 3233	380.59k	4835 × 4279	160.43k
rand13	200	80	50835 × 6402	1.71M	17652 × 8399	664.51k
rand14	200	80	50840 × 6427	1.56M	17908 × 8456	623.57k
rand15	200	80	50830 × 6371	1.49M	17396 × 8336	606.71k
rand16	300	80	113755 × 9627	3.92M	39231 × 12639	1.49M
rand17	300	80	113770 × 9726	3.64M	40374 × 12834	1.48M
rand18	300	80	113760 × 9650	3.66M	39612 × 12694	1.46M
udine1	207	86	50350 × 4297	963.38k	11756 × 5393	280.62k
udine2	223	93	54440 × 5626	1.30M	13452 × 6889	378.48k
udine3	252	97	66940 × 7883	2.20M	16036 × 9252	579.15k
udine4	250	100	64200 × 12060	3.70M	15505 × 13678	915.37k

Table C.3: The performance of the standard and the proposed (New) formulation of vertex colouring, measured in run times of CPLEX and numbers of iterations performed with no built-in symmetry breaking (-0). The last column lists ratios of CPLEX run times.

Instance	Std-0	(Its.)	New-0	(Its.)	$\frac{\text{Std-0}}{\text{New-0}}$
rand01	2.85s	1635	0.90s	931	3.16
rand02	2.99s	1666	0.94s	1106	3.18
rand03	9.92s	5792	1.05s	1045	9.45
rand04	99.48s	26317	5.18s	2802	19.20
rand05	73.72s	19802	33.49s	17467	2.20
rand06	83.78s	22537	40.35s	19836	2.08
rand07	216.08s	35821	86.44s	25541	2.50
rand08	59.70s	10760	43.45s	13342	1.37
rand09	127.19s	22155	98.32s	25782	1.29
rand11	3.80s	1761	1.51s	1194	2.52
rand12	4.55s	2005	2.31s	1377	1.97
rand13	95.67s	22851	47.94s	18957	2.00
rand14	45.25s	10544	6.64s	2629	6.81
rand15	30.77s	6799	6.89s	2685	4.47
rand16	114.32s	11603	275.44s	51518	0.42
rand17	251.15s	33185	144.93s	36949	1.73
rand18	160.25s	21686	138.04s	34461	1.16
udine1	23.23s	8082	4.45s	3370	5.22
udine2	14.51s	4749	10.04s	4826	1.45
udine3	83.41s	16807	17.25s	11698	4.84
udine4	144.49s	30655	145.99s	30655	0.99

Table C.4: The performance of two formulations of Udine Course Timetabling, differing only in the formulation of the underlying graph colouring component: run times of CPLEX or gaps remaining after 1 hour of solving and numbers of iterations performed with no built-in symmetry breaking (-0). The last column lists ratios of CPLEX run times, where optimality was reached within 1 hour using both formulations.

Instance	Std-0	(Its.)	New-0	(Its.)	$\frac{\text{Std-0}}{\text{New-0}}$
rand01	385.59s	180854	84.42s	43737	4.57
rand02	290.09s	71537	72.42s	34296	4.01
rand03	443.95s	148961	59.99s	23310	7.40
rand04	gap 0.24%	419910	1242.50s	210104	
rand05	gap 4.15%	360868	1194.71s	250148	
rand06	gap 8.33%	299998	1257.72s	247075	
rand07	gap 89.71%	234087	gap 90.11%	242978	
rand08	gap 99.85%	237243	gap 99.90%	312158	
rand09	gap 93.97%	199619	gap 95.44%	263820	
rand10	285.91s	66842	70.17s	27416	4.07
rand11	211.71s	68244	61.32s	31738	3.45
rand12	337.31s	129788	84.16s	48401	4.01
rand13	gap 0.24%	431148	884.60s	175513	
rand14	gap 6.47%	322073	1356.97s	320129	
rand15	gap 1.74%	303518	1166.50s	280722	
rand16	gap 66.44%	175766	gap 67.19%	417706	
rand17	gap 94.15%	239576	gap 94.06%	293519	
rand18	gap 90.57%	251822	gap 49.34%	345817	
udine1	1175.40s	166539	237.12s	104221	4.96
udine2	gap 100.00%	639068	gap 100.00%	3318838	
udine3	gap 99.31%	367505	gap 59.59%	2000062	
udine4	gap 99.69%	220364	gap infinite	962856	

Table C.5: The performance of two formulations of Udine Course Timetabling, differing only in the formulation of the underlying graph colouring component, and the effects of disabling (+0) the built-in symmetry breaking in CPLEX, or setting it to very aggressive (+3): run times of CPLEX or gaps remaining after 1 hour of solving.

Instance	Std+0	New+0	Std+3	New+3	$\frac{\text{Std+3}}{\text{New-0}}$
rand01	385.59s	84.42s	165.52s	76.45s	1.96
rand02	290.09s	72.42s	343.33s	65.51s	4.74
rand03	443.95s	59.99s	298.52s	72.06s	4.98
rand04	gap 0.24%	1242.50s	gap 0.24%	1356.63s	
rand05	gap 4.15%	1194.71s	gap 4.15%	1107.12s	
rand06	gap 8.33%	1257.72s	gap 8.33%	1162.52s	
rand07	gap 89.71%	gap 90.11%	gap 89.71%	gap 90.11%	
rand08	gap 99.85%	gap 99.90%	gap 99.85%	gap 99.90%	
rand09	gap 93.97%	gap 95.44%	gap 93.97%	gap 95.44%	
rand10	285.91s	70.17s	321.51s	81.12s	4.58
rand11	211.71s	61.32s	207.41s	56.79s	3.38
rand12	337.31s	84.16s	253.75s	84.64s	3.02
rand13	gap 0.24%	884.60s	gap 1.85%	795.50s	
rand14	gap 6.47%	1356.97s	gap 6.47%	1197.39s	
rand15	gap 1.74%	1166.50s	gap 30.43%	1051.74s	
rand16	gap 66.44%	gap 67.19%	gap 66.44%	gap 67.19%	
rand17	gap 94.15%	gap 94.06%	gap 94.15%	gap 94.06%	
rand18	gap 90.57%	gap 49.34%	gap 90.57%	gap 92.25%	
udine1	1175.40s	237.12s	1247.33s	142.84s	5.26
udine2	gap 100.00%	gap 100.00%	gap 100.00%	gap 100.00%	
udine3	gap 99.31%	gap 59.59%	gap 99.33%	gap 70.04%	
udine4	gap 99.69%	gap infinite	gap infinite	gap infinite	

Table C.6: Results for instances from Track 3 (comp) of the International Timetabling Competition 2007.

Graph	LB	Runtime	LB^m	Runtime	Rounded
comp01.course	4.00	4 s	5.00	0 s	7
comp02.course	5.98	2 s	6.00	7 s	12
comp03.course	6.94	1 s	7.00	7 s	14
comp04.course	4.98	1 s	5.00	3 s	12
comp05.course	8.00	1 s	7.99	3 s	14
comp06.course	5.99	3 s	6.00	8 s	14
comp07.course	6.00	5 s	6.55	21 s	17
comp08.course	6.99	2 s	6.98	7 s	11
comp09.course	4.99	1 s	5.00	5 s	10
comp10.course	6.00	3 s	6.39	11 s	16
comp11.course	5.00	1 s	6.00	0 s	8
comp12.course	9.91	4 s	9.96	15 s	18
comp13.course	5.98	1 s	6.00	7 s	8
comp14.course	6.00	2 s	6.00	10 s	14
comp15.course	6.94	1 s	7.00	7 s	15
comp16.course	6.00	4 s	5.99	9 s	15
comp17.course	5.98	3 s	6.00	14 s	12
comp18.course	4.99	1 s	5.22	1 s	8
comp19.course	6.00	1 s	6.00	5 s	11
comp20.course	6.00	4 s	6.37	14 s	13
comp21.course	8.00	2 s	8.00	14 s	13

Table C.7: For Knesser graphs $K(n,2)$ and forbidden intersection graphs $F(n,\gamma)$, where the size of the largest colour class in an optimal colouring is C, lower bounds LB^m and optima χ^m for $(C-m)$-bounded colouring are shown. For $m=0$, no bounds were applied.

Graph	LB^0	χ^0	LB^{-1}	χ^{-1}	LB^{-2}	χ^{-2}	LB^{-3}	χ^{-3}
$K(5,2)$	2.50	3	3.33	4	5.00	5	10.00	1
$K(6,2)$	3.00	4	3.75	4	5.00	5	7.50	8
$K(7,2)$	3.50	5	4.20	5	5.25	6	7.00	7
$K(8,2)$	4.67	6	5.60	6	7.00	7	9.33	10
$FI(6,0.50)$	2.00	2	2.03	3	2.06	3	2.13	3
$FI(6,0.67)$	6.40	7	7.11	8	8.00	8	9.14	10
$FI(6,0.83)$	2.00	2	2.03	3	2.06	3	2.13	3
$FI(6,1.00)$	2.00	2	2.00	2	2.06	3	2.13	3

Table C.8: The performance of various solvers on 3D Pigeon Hole Problem instances encoded in the Chen/Padberg formulation. "–" denotes that optimality of the incumbent solution has not been proven within an hour.

	Time (s)		
	Gurobi 4.0	CPLEX 12.2.0	SCIP 2.0.1 + CLP
Pigeon-01	< 1	< 1	< 1
Pigeon-02	< 1	< 1	< 1
Pigeon-03	< 1	< 1	< 1
Pigeon-04	< 1	< 1	< 1
Pigeon-05	< 1	< 1	3.3
Pigeon-06	< 1	< 1	37.9
Pigeon-07	1.5	3.6	779.3
Pigeon-08	7.4	25.6	–
Pigeon-09	88.6	398.4	–
Pigeon-10	1381.4	–	–
Pigeon-11	–	–	–
Pigeon-12	–	–	–

Table C.9: The performance of Gurobi 4.0 on 3D Pigeon Hole Problem instances encoded in the Chen/Padberg and the space-indexed formulations. "–" denotes that no integer solution has been found.

	Time (s)	
	Chen/Padberg	Space-indexed
Pigeon-01	< 1	< 1
Pigeon-02	< 1	< 1
Pigeon-03	< 1	< 1
Pigeon-04	< 1	< 1
Pigeon-05	< 1	< 1
Pigeon-06	< 1	< 1
Pigeon-07	1.5	< 1
Pigeon-08	7.4	< 1
Pigeon-09	88.6	< 1
Pigeon-10	1381.4	< 1
Pigeon-100	–	< 1
Pigeon-1000	–	1.0
Pigeon-10000	–	1.8
Pigeon-100000	–	4.2
Pigeon-1000000	–	45.1
Pigeon-10000000	–	664.0
Pigeon-100000000	–	–

Table C.10: The performance of the space-indexed model on the Van-loading and Van-loading-X instances after one hour. Each row represents 10 pseudo-randomly generated instances.

Dataset	Box Types	Boxes (Mean)	Gap (Mean)
Van-loading-01	3	78	0.10%
Van-loading-02	5	92	0.31%
Van-loading-03	8	81	0.42%
Van-loading-04	10	73	0.32%
Van-loading-05	12	73	0.09%
Van-loading-06	15	70	0.42%
Van-loading-07	20	68	0.50%
Van-loading-08	30	71	0.99%
Van-loading-09	40	69	1.26%
Van-loading-10	50	73	1.10%
Van-loading-X-01	3	133	0.16%
Van-loading-X-02	5	155	0.16%
Van-loading-X-03	8	132	0.56%
Van-loading-X-04	10	119	0.87%
Van-loading-X-05	12	123	0.29%
Van-loading-X-06	15	115	0.36%
Van-loading-X-07	20	116	0.30%
Van-loading-X-08	30	122	0.29%
Van-loading-X-09	40	119	0.18%
Van-loading-X-10	50	121	0.28%

Figure C.1: The best solutions obtained within an hour per solver per instance from the SA dataset. Each square represents 1 pseudo-randomly generated instance. The number is the volume utilisation (in percent) with the tight upper bound of 100.

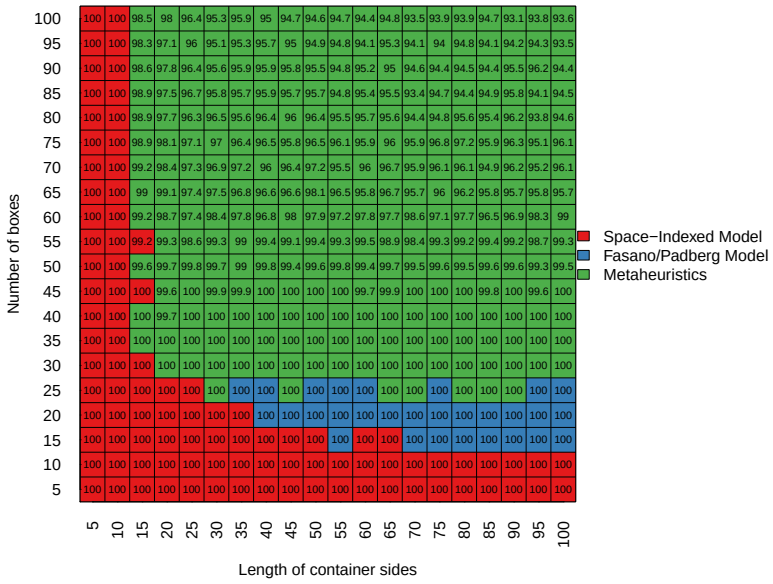
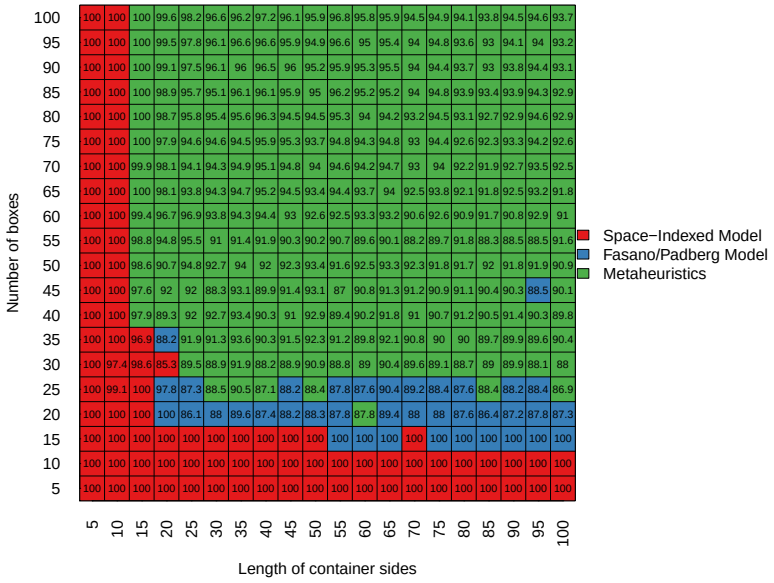


Figure C.2: The best solutions obtained within an hour per solver per instance from the SAX dataset. Each square represents 1 pseudo-randomly generated instance. The number displayed is $100(1 - s/b)$ for solution with value s and upper bound b.



Bibliography

[1] K. Aardal, C. A. J. Hurkens, and A. K. Lenstra, *Solving a system of linear Diophantine equations with lower and upper bounds on the variables*, Math. Oper. Res. **25**, no. 3 (2000), pp. 427–442. [41, 43]

[2] K. Aardal, R. Weismantel, and L. A. Wolsey, *Non-standard approaches to integer programming*, Discrete Appl. Math. **123**, no. 1-3 (2002), pp. 5–74. Workshop on Discrete Optimization, DO'99 (Piscataway, NJ). [42]

[3] R. Aboudi, Å. Hallefjord, R. Helming, and K. Jørnsten, *A note on the pivot and complement heuristic for 0-1 programming problems*, Oper. Res. Lett. **8**, no. 1 (1989), pp. 21–23. [21]

[4] R. Aboudi and K. Jørnsten, *Tabu search for general zero-one integer programs using the pivot and complement heuristic*, ORSA J. Computing **6**, no. 1 (1994), pp. 82–93. [20, 21]

[5] D. Abramson, H. Dang, and M. Krishnamoorthy, *A comparison of two methods for solving 0-1 integer programs using a general purpose simulated annealing algorithm*, Ann. Oper. Res. **63** (1996), pp. 129–150. [19]

[6] D. Achlioptas and A. Naor, *The two possible values of the chromatic number of a random graph*, Ann. of Math. (2) **162**, no. 3 (2005), pp. 1335–1351. [94]

[7] T. Achterberg, *Conflict analysis in mixed integer programming*, Discrete Optim. **4**, no. 1 (2007), pp. 4–20. [21, 28]

[8] T. Achterberg, *Constraint Integer Programming*, PhD thesis, Technische Universität Berlin, Berlin, 2007. [5, 19, 28, 29, 30, 138]

[9] T. Achterberg, *SCIP: solving constraint integer programs*, Math. Program. Comput. **1**, no. 1 (2009), pp. 1–41. [16, 19, 29, 30, 129]

[10] T. Achterberg and T. Berthold, *Improving the feasibility pump*, Discrete Optim. **4** (2007), pp. 77–86. [21, 29, 30]

[11] T. Achterberg, T. Koch, and A. Martin, *Branching rules revisited*, Op. Res. Lett. **33** (2005), pp. 42–54. ZIB Technical Report TR-04-13. [28, 29, 30]

[12] T. Achterberg and C. Raack, *The Mcf-separator: detecting and exploiting multi-commodity flow structures in MIPs*, Math. Programming Comput. **2**, no. 2 (2010), pp. 125–165. [8, 40, 69]

[13] I. Adler, B.-M. Bui-Xuan, Y. Rabinovich, G. Renault, J. Telle, and M. Vatshelle, *On the boolean-width of a graph: Structure and applications*, in Graph Theoretic Concepts in Computer Science, D. Thilikos, ed., Lecture Notes in Computer Science 6410, Springer Berlin / Heidelberg, 2010, pp. 159–170. [48, 49, 132]

[14] K. H. Ahn, *Algorithms for Identification of Embedded Networks and Specialized Simplex Methods for Embedded-Network Linear Programs*, PhD thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, 1984. [63]

[15] A. V. Aho, D. S. Johnson, R. M. Karp, S. R. Kosaraju, C. C. McGeoch, C. H. Papadimitriou, and P. Pevzner, *Emerging opportunities for theoretical computer science*, SIGACT News **28** (1997), pp. 65–74. [39]

[16] A. Ali, H.-S. Han, and J. Kennington, *Use of hidden network structure in the set partitioning problem*, in Integer Programming and Combinatorial Optimization, E. Balas and J. Clausen, eds., Lecture Notes in Computer Science 920, Springer Berlin / Heidelberg, 1995, pp. 172–184. [8, 40, 63]

[17] A. I. Ali and H. Thiagarajan, *A network relaxation based enumeration algorithm for set partitioning*, European J. Oper. Res. **38**, no. 1 (1989), pp. 76–85. [40, 63]

[18] S. D. Allen, E. K. Burke, and G. Kendall, *A hybrid placement strategy for the three-dimensional strip packing problem*, European Journal of Operational Research **209**, no. 3 (2011), pp. 219 – 227. [130]

[19] S. D. Allen, E. K. Burke, and J. Mareček, *A space-indexed formulation of packing boxes into a larger box*, Oper. Res. Lett. **to appear** (2011). [131]

[20] N. Alon, *Personal communication, June 12, 2009. Electronic mail.* [v]

[21] N. Alon, *Spectral techniques in graph algorithms*, in LATIN'98: Theoretical Informatics, C. Lucchesi and A. Moura, eds., Lecture Notes in Computer Science 1380, Springer Berlin / Heidelberg, 1998, pp. 206–215. [33]

[22] N. Alon and N. Kahale, *A spectral technique for coloring random 3-colorable graphs*, SIAM J. Comput. **26**, no. 6 (1997), pp. 1733–1748. [85, 86]

[23] N. Alon and J. H. Spencer, *The probabilistic method*, Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons Inc., Hoboken, NJ, third ed., 2008. [v]

[24] G. Appa and B. Kotnyek, *A bidirected generalization of network matrices*, Networks **47**, no. 4 (2006), pp. 185–198. [61]

[25] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Travelling Salesman Problem: A Computational Study*, Princeton UP, Princeton, NJ, 2007. [25, 29, 41]

[26] S. Arora and B. Barak, *Computational complexity*, Cambridge University Press, Cambridge, 2009. A modern approach. [3, 15]

[27] A. Atamtürk and M. W. P. Savelsbergh, *Integer-programming software systems*, Ann. Oper. Res. **140** (2005), pp. 67–124. [19, 28]

[28] J. E. Atkins, E. G. Boman, and B. Hendrickson, *A spectral algorithm for seriation and the consecutive ones problem*, SIAM J. Comput. **28**, no. 1 (1999), pp. 297–310. [69]

[29] C. Aykanat, A. Pinar, and Ü. V. Çatalyürek, *Permuting sparse rectangular matrices into block-diagonal form*, SIAM J. Sci. Comput. **25**, no. 6 (2004), pp. 1860–1879. [68]

[30] E. Balas, *An additive algorithm for solving linear programs with zero-one variables*, Operations Res. **13** (1965), pp. 517–546. [19, 28]

[31] E. Balas, *Ranking the facets of the octahedron*, Discrete Math. **2**, no. 2 (1972), pp. 1–15. [22]

[32] E. Balas, S. Ceria, M. Dawande, F. Margot, and G. Pataki, *Octane: A new heuristic for pure 0-1 programs*, Oper. Res. **49**, no. 2 (2001), pp. 207–225. [22]

[33] E. Balas and C. H. Martin, *Pivot and complement – a heuristic for* $0-1$ *programming*, Manage. Sci. **26**, no. 1 (1980), pp. 86–96. [21]

[34] E. Balas, S. Schmieta, and C. Wallace, *Pivot and shift – a mixed integer programming heuristic*, Discrete Optim. **1**, no. 1 (2004), pp. 3–12. [21]

[35] V. C. Barbosa, C. A. G. Assis, and J. O. do Nascimento, *Two novel evolutionary formulations of the graph coloring problem*, J. Comb. Optim. **8**, no. 1 (2004), pp. 41–63. [95]

[36] J. Barnat and P. Moravec, *Parallel algorithms for finding sccs in implicitly given graphs*, in Proceedings of the 11th international workshop, FMICS 2006 and 5th international workshop, PDMC conference on Formal methods: Applications and technology, FMICS'06/PDMC'06, Berlin, Heidelberg, 2007, Springer-Verlag, pp. 316–330. [89]

[37] J. J. Bartholdi, III, *A good submatrix is hard to find*, Oper. Res. Lett. **1**, no. 5 (1981/82), pp. 190–193. [61]

[38] R. Battiti and G. Tecchiolli, *Local search with memory: benchmarking RTS*, OR Spectrum **17** (1995), pp. 67–86. [19]

[39] E. M. L. Beale and J. J. H. Forrest, *Global optimization using special ordered sets*, Math. Programming **10** (1976), pp. 52–69. [5, 6]

[40] J. Bean and A. Hadj-Alouane, *A dual genetic algorithm for bounded integer programs*, Tech. Report 92-53, The Univ. of Michigan Department of Indust. & Oper. Engng, 1992. [25]

[41] D. E. Bell, *A theorem concerning the integer lattice*, Studies in Appl. Math. **56**, no. 2 (1976/77), pp. 187–188. [42]

[42] R. Bellman and K. Fan, *On systems of linear inequalities in Hermitian matrix variables*, in Proc. Sympos. Pure Math., Vol. VII, Amer. Math. Soc., Providence, R.I., 1963, pp. 1–11. [32]

[43] R. Belmonte and M. Vatshelle, *Graph classes with structured neighborhoods and algorithmic applications*, in Graph Theoretic Concepts in Computer Science, J. Kratochvíl, ed., Lecture Notes in Computer Science 6410, Springer Berlin / Heidelberg, 2011, pp. 159–170. [132]

[44] M. Benichou, J.-M. Gauthier, P. Girodet, G. Hentges, G. Ribière, and O. Vincent, *Experiments in mixed-integer linear programming*, Math. Programming **1**, no. 1 (1971), pp. 76–94. [29, 30]

[45] M. Bergner, A. Caprara, F. Furini, M. Lübbecke, E. Malaguti, and E. Traversi, *Partial convexification of general MIPs by dantzig-wolfe reformulation*, in Proceedings of Integer Programming and Combinatorial Optimization, 15th International IPCO Conference, Yorktown Heights, New York, USA, June 15–17, 2011, Lecture Notes in Computer Science, Springer, 2011. [8, 40, 43, 65, 69]

[46] Y. Berstein and S. Onn, *The Graver complexity of integer programming*, Ann. Comb. **13**, no. 3 (2009), pp. 289–296. [41]

[47] L. Bertacco, M. Fischetti, and A. Lodi, *A feasibility pump heuristic for general mixed-integer problems*, Discrete Optim. **4**, no. 1 (2008), pp. 63–76. [21]

[48] T. Berthold, *Primal heuristics for mixed integer programs*, master's thesis, Technische Universität Berlin, 2006. [19, 20, 21, 22, 25, 138]

[49] T. Berthold, *RENS – Relaxation enforced neighborhood search*, tech. report, Zuse Institute, Berlin, 2007. ZIB Technical Report TR-07-28. [20]

[50] D. Bertsimas, C.-P. Teo, and R. Vohra, *On dependent randomized rounding algorithms*, in Proceedings of Integer Programming and Combinatorial Optimization, 5th International IPCO Conference, Vancouver, British Columbia, Canada, June 3-5, 1996, W. H. Cunningham, S. T. McCormick, and M. Queyranne, eds., Lecture Notes in Computer Science 1084, Springer, 1996, pp. 330–344. [19, 22]

[51] D. Bertsimas, C.-P. Teo, and R. Vohra, *New randomized rounding algorithms*, Oper. Res. Lett. **24**, no. 3 (1999), pp. 105–114. [19, 22]

[52] D. Bertsimas and R. V. Vohra, *Rounding algorithms for covering problems*, Math. Programming **80** (1998), pp. 63–89. [22]

[53] D. Bienstock and N. Ozbay, *Tree-width and the Sherali-Adams operator*, Discrete Optim. **1**, no. 1 (2004), pp. 13–21. [53]

[54] D. Bienstock and M. Zuckerberg, *Solving lp relaxations of large-scale precedence constrained problems*, in Integer Programming and Combinatorial Optimization, F. Eisenbrand and F. Shepherd, eds., Lecture Notes in Computer Science 6080, Springer Berlin / Heidelberg, 2010, pp. 1–14. [119]

[55] L. J. Billera and A. Sarangarajan, *All* 0-1 *polytopes are traveling salesman polytopes*, Combinatorica **16**, no. 2 (1996), pp. 175–188. [41]

[56] R. Bixby and E. Rothberg, *Progress in computational mixed integer programming—a look back from the other side of the tipping point*, Ann. Oper. Res. **149** (2007), pp. 37–41. [75, 131]

[57] R. E. Bixby, *Personal communication, September 25, 2009. Combinatorial Optimization at Work II, Berlin.* [75]

[58] R. E. Bixby, *Kuratowski's and Wagner's theorems for matroids*, J. Combinatorial Theory Ser. B **22**, no. 1 (1977), pp. 31–53. [59]

[59] R. E. Bixby, *Hidden structure in linear programs*, in Proceedings of the 1st Symposium on Computer-Assisted Analysis and Model Simplification, Boulder, CO, 1980, H. J. Greenberg and J. S. Maybee, eds., Academic Press, 1981. [43, 76]

[60] R. E. Bixby, *Solving real-world linear programs: A decade and more of progress*, Oper. Res. **50**, no. 1 (2002), pp. 3–15. [5, 131]

[61] R. E. Bixby, S. Ceria, C. M. McZeal, and M. W. P. Savelsbergh, *An updated mixed integer programming library: MIPLIB 3.0*, Optima **58** (1998), pp. 12–15. [5]

[62] R. E. Bixby and W. H. Cunningham, *Converting linear programs to network problems*, Math. Oper. Res. **5**, no. 3 (1980), pp. 321–357. [60, 61]

[63] R. E. Bixby, M. Fenelon, Z. Gu, E. E. Rothberg, and R. Wunderling, *MIP: Theory and practice – closing the gap*, in Proceedings of the 19th IFIP TC7 Conference on System Modelling and Optimization, Deventer, The Netherlands, 2000, Kluwer, pp. 19–50. [5]

[64] R. E. Bixby, M. Fenelon, Z. Gu, E. E. Rothberg, and R. Wunderling, *Mixed-integer programming: A progress report*, in The Sharpest Cut: The Impact of Manfred Padberg and His Work, M. Grötschel, ed., SIAM, Philadelphia, PA, 2004. [5, 131]

[65] R. E. Bixby and R. Fourer, *Finding embedded network rows in linear programs. II. Augmentation heuristics*, 1987. [62, 63]

[66] R. E. Bixby and R. Fourer, *Finding embedded network rows in linear programs. I. Extraction heuristics*, Management Sci. **34**, no. 3 (1988), pp. 342–376. [43, 62, 63]

[67] J. Blömer, R. Karp, and E. Welzl, *The rank of sparse random matrices over finite fields*, Random Structures Algorithms **10**, no. 4 (1997), pp. 407–419. [53]

[68] H. L. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput. **25**, no. 6 (1996), pp. 1305–1317. [56]

[69] H. L. Bodlaender, *A partial k-arboretum of graphs with bounded treewidth*, Theoret. Comput. Sci. **209**, no. 1-2 (1998), pp. 1–45. [48]

[70] H. L. Bodlaender and F. V. Fomin, *Equitable colorings of bounded treewidth graphs*, Theoret. Comput. Sci. **349**, no. 1 (2005), pp. 22–30. [97]

[71] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks, *Approximating treewidth, pathwidth, frontsize, and shortest elimination tree*, J. Algorithms **18**, no. 2 (1995), pp. 238–255. [51]

[72] H. L. Bodlaender and A. M. C. A. Koster, *Treewidth computations. I. Upper bounds*, Inform. and Comput. **208**, no. 3 (2010), pp. 259–275. [50, 51]

[73] B. Bollobás, *Random graphs*, Cambridge Studies in Advanced Mathematics 73, Cambridge University Press, Cambridge, second ed., 2001. [87]

[74] I. Bomze, F. Frommlet, and M. Locatelli, *Copositivity cuts for improving sdp bounds on the clique number*, Mathematical Programming **124** (2010), pp. 13–32. [107]

[75] P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot, *A feasibility pump for mixed integer nonlinear programs*, Math. Programming (2008), p. To appear. [21]

[76] A. Bonutti, F. De Cesco, L. Di Gaspero, and A. Schaerf, *Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results*, Ann. Oper. Res. (2010), pp. 1–12. [115]

[77] R. Borndörfer, C. E. Ferreira, and A. Martin, *Matrix decomposition by branch-and-cut*, tech. report, Zuse Institute, Berlin, 1997. ZIB Technical Report SC 97-14. [67, 68]

[78] R. Borndörfer, C. E. Ferreira, and A. Martin, *Decomposing matrices into blocks*, SIAM J. Optim. **9**, no. 1 (1999), pp. 236–269. [67, 68]

[79] H. Bosse, M. Grötschel, and M. Henk, *Polynomial inequalities representing polyhedra*, Math. Program. **103**, no. 1, Ser. A (2005), pp. 35–44. [41, 43]

[80] G. G. Brown, R. D. McBride, and R. K. Wood, *Extracting embedded generalized networks from linear programming problems*, Math. Programming **32**, no. 1 (1985), pp. 11–31. [62]

[81] G. G. Brown and M. P. Olson, *Dynamic factorization in large-scale optimization*, Math. Programming **64**, no. 1, Ser. A (1994), pp. 17–51. [76]

[82] G. G. Brown and W. G. Wright, *Automatic identification of embedded structure in large-scale optimization models*, in Proceedings of the 1st Symposium on Computer-Assisted Analysis and Model Simplification, Boulder, CO, 1980, H. J. Greenberg and J. S. Maybee, eds., Academic Press, 1981. [62]

[83] G. G. Brown and W. G. Wright, *Automatic identification of embedded network rows in large-scale optimization models*, Math. Programming **29**, no. 1 (1984), pp. 41–56. [62]

[84] T. N. Bui and C. Jones, *Finding good approximate vertex and edge partitions is NP-hard*, Inform. Process. Lett. **42**, no. 3 (1992), pp. 153–159. [67]

[85] B. Bui-Xuan, J. Telle, and M. Vatshelle, *Boolean-width of graphs*, in Parameterized and Exact Computation, J. Chen and F. Fomin, eds., Lecture Notes in Computer Science 5917, Springer Berlin / Heidelberg, 2009, pp. 61–74. [48, 49, 132]

[86] S. Burer and D. Vandenbussche, *Solving lift-and-project relaxations of binary integer programs*, SIAM J. Optim. **16**, no. 3 (2006), pp. 726–750. [110]

[87] R. E. Burkard, M. Kocher, and R. Rudolf, *Rounding strategies for mixed integer programs arising from chemical production planning*, Yugoslav J. of Operations Research **8** (1998), pp. 9–23. [19]

[88] E. K. Burke, J. Mareček, A. J. Parkes, and H. Rudová, *Decomposition, reformulation, and diving in university course timetabling*, Comput. Oper. Res. **37**, no. 1 (2010), pp. 582–597. [90, 91, 131]

[89] E. K. Burke, J. Mareček, A. J. Parkes, and H. Rudová, *A supernodal formulation of vertex colouring with applications in course timetabling*, Ann. Oper. Res. **179**, no. 1 (2010), pp. 105–130. [95]

[90] E. K. Burke, J. Mareček, A. J. Parkes, and H. Rudová, *A branch-and-cut procedure for the udine course timetabling problem*, Ann. Oper. Res. (2011), p. To appear. [115]

[91] J. V. Burke and J. J. More, *On the identification of active constraints*, SIAM J. Numer. Anal. **25**, no. 5 (1988), pp. 1197–1211. [29]

[92] S. R. Buss and J. Krajíček, *An application of Boolean complexity to separation problems in bounded arithmetic*, Proc. London Math. Soc. (3) **69**, no. 1 (1994), pp. 1–21. [23]

[93] P. Camion, *Characterization of totally unimodular matrices*, Proc. Amer. Math. Soc. **16** (1965), pp. 1068–1073. [57, 58]

[94] M. Campêlo, V. A. Campos, and R. C. Corrêa, *On the asymmetric representatives formulation for the vertex coloring problem*, Discrete Appl. Math. **156**, no. 7 (2008), pp. 1097–1111. [95]

[95] A. Caprara, *Properties of some ILP formulations of a class of partitioning problems*, Discrete Appl. Math. **87**, no. 1-3 (1998), pp. 11–23. [94]

[96] A. Caprara and M. Fischetti, $\{0, \frac{1}{2}\}$-*Chvátal-Gomory cuts*, Math. Programming **74**, no. 3, Ser. A (1996), pp. 221–235. [31, 32]

[97] Ü. V. Çatalyürek, *Hypergraph models for sparse matrix partitioning and reordering*, PhD thesis, Bilkent University, 1999. [51]

[98] Ü. V. Çatalyürek, C. Aykanat, and B. Uçar, *On two-dimensional sparse matrix partitioning: models, methods, and a recipe*, SIAM J. Sci. Comput. **32**, no. 2 (2010), pp. 656–683. [51]

[99] A. Chabrier, E. Danna, C. L. Pape, and L. Perron, *Solving a network design problem*, Ann. Oper. Res. **130** (2004), pp. 217–239. [25]

[100] R. Chandrasekaran, *Total unimodularity of matrices*, SIAM J. Appl. Math. **17** (1969), pp. 1032–1034. [58]

[101] M. Charikar, *On semidefinite programming relaxations for graph coloring and vertex cover*, in SODA '02: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, 2002, Society for Industrial and Applied Mathematics, pp. 616–620. [116]

[102] C. S. Chen, S. M. Lee, and Q. S. Shen, *An analytical model for the container loading problem*, European J. Oper. Res. **80**, no. 1 (1995), pp. 68–76. [120]

[103] J. Cheriyan and K. Mehlhorn, *Algorithms for dense graphs and networks on the random access computer*, Algorithmica **15**, no. 6 (1996), pp. 521–549. [89]

[104] M. Chiarandini, I. Dumitrescu, and T. Stützle, *Stochastic local search algorithms for the graph colouring problem*, in Handbook of Approximation Algorithms and Metaheuristics, T. F. Gonzalez, ed., Computer & Information Science Series, Chapman & Hall/CRC, Boca Raton, FL, USA, 2007, pp. 63.1–63.17. [102]

[105] M. Chlebík and J. Chlebíková, *Hardness of approximation for orthogonal rectangle packing and covering problems*, J. Discrete Algorithms **7**, no. 3 (2009), pp. 291–305. [120]

[106] P. M. Christophel, *An improved heuristic for the mops mixed integer programming solver*, master's thesis, Universiät Padeborn, 2005. [19, 20]

[107] P. M. Christophel, L. Suhl, and U. H. Suhl, *Finding feasible solutions to hard mixed-integer programming problems using hybrid heuristics*, in Operations Research Proceedings 2005, H.-D. Haasis, H. Kopfer, and J. Schönberger, eds., Operations Research Proceedings 2005, Springer Berlin Heidelberg, 2006, pp. 355–360. [20]

[108] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas, *The strong perfect graph theorem*, Ann. of Math. (2) **164**, no. 1 (2006), pp. 51–229. [103]

[109] V. Chvátal, *Edmonds polytopes and a hierarchy of combinatorial problems*, Discrete Math. **4** (1973), pp. 305–337. [31]

[110] V. Chvátal, *Linear Programming*, W. H. Freeman, New York, NY, 1983. [5]

[111] V. Chvátal, *Resolution search*, Discrete Appl. Math. **73**, no. 1 (1997), pp. 81–99. [30]

[112] K. L. Clarkson, *Las vegas algorithms for linear and integer programming when the dimension is small*, J. ACM **42** (1995), pp. 488–499. [42, 43]

[113] A. Coja-Oghlan, *The Lovász number of random graphs*, Combin. Probab. Comput. **14**, no. 4 (2005), pp. 439–465. [94]

[114] A. Coja-Oghlan and A. Taraz, *Exact and approximative algorithms for coloring $G(n, p)$*, Random Structures Algorithms **24**, no. 3 (2004), pp. 259–278. [84]

[115] P. Coll, J. Marenco, I. Méndez-Díaz, and P. Zabala, *Facets of the graph coloring polytope*, Ann. Oper. Res. **116** (2002), pp. 79–90. [94, 95]

[116] M. Colombo, A. Grothey, J. Hogg, K. Woodsend, and J. Gondzio, *A structure-conveying modelling language for mathematical and stochastic programming*, Math. Program. Comput. **1**, no. 4 (2009), pp. 223–247. [135]

[117] S. Cook and J. Krajíček, *Consequences of the provability of* $\mathbf{NP} \subseteq \mathbf{P/poly}$, J. Symbolic Logic **72**, no. 4 (2007), pp. 1353–1371. [26]

[118] S. A. Cook, *The complexity of theorem-proving procedures*, in Proceedings of the third annual ACM symposium on Theory of computing, STOC '71, New York, NY, USA, 1971, ACM, pp. 151–158. [15]

[119] S. A. Cook and R. A. Reckhow, *The relative efficiency of propositional proof systems*, J. Symbolic Logic **44**, no. 1 (1979), pp. 36–50. [26]

[120] W. Cook, T. Rutherford, H. E. Scarf, and D. Shallcross, *An implementation of the generalized basis reduction algorithm for integer programming*, ORSA J. Comput. **5**, no. 2 (1993), pp. 206–212. [41, 43]

[121] W. Cook and P. Seymour, *Tour merging via branch-decomposition*, INFORMS J. Comput. **15**, no. 3 (2003), pp. 233–248. [51, 85]

[122] W. J. Cook, R. Fukasawa, and M. Goycoolea, *Effectiveness of different cut selection rules*, in MIP2006, 2006. [138]

[123] G. Cote and M. A. Laughton, *Large-scale mixed integer programming: Benders-type heuristics*, European J. Oper. Res. **16** (1984), pp. 327–333. [21]

[124] T. G. Crainic, M. Gendreau, and J. M. Farvolden, *A simplex-based tabu search method for capacitated network design*, INFORMS J. on Computing **12**, no. 3 (2000), pp. 223–236. [20]

[125] H. Crowder, E. L. Johnson, and M. Padberg, *Solving large-scale zero-one linear programming problems*, Oper. Res. **31**, no. 5 (1983), pp. 803–834. [16]

[126] W. H. Cunningham and J. Edmonds, *A combinatorial decomposition theory*, Canad. J. Math. **32**, no. 3 (1980), pp. 734–765. [103]

[127] W. H. Cunningham and J. A. Geelen, *On integer programming and the branch-width of the constraint matrix*, in Integer Programming and Combinatorial Optimization (Ithaca, NY), Lecture Notes in Comput. Sci., Springer, Berlin, 2007, pp. 158–166. [47, 51, 52]

[128] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 1969 24th national conference, ACM '69, New York, NY, USA, 1969, ACM, pp. 157–172. [40]

[129] C. D'AMBROSIO, *Application-oriented Mixed Integer Non-Linear Programming*, PhD thesis, Università degli studi di Bologna, 2009. [21]

[130] F. DAMMEYER AND S. VOSS, *Dynamic tabu list management using the reverse elimination method*, Ann. Oper. Res. **41** (1993), pp. 29–46. [20]

[131] E. DANNA, *Integrating local search techniques into mixed integer programming*, 4OR **2**, no. 4 (2004), pp. 321–324. [25]

[132] E. DANNA AND A. LODI, *Using infeasible nodes to select branching variables.* ISMP 2009 – The 20th International Symposium of Mathematical Programming, August 2009. Chicago, US. [16]

[133] E. DANNA, E. E. ROTHBERG, AND C. L. PAPE, *Exploring relaxation induced neighborhoods to improve MIP solutions*, Math. Programming **102**, no. 1 (2005), pp. 71–90. [25, 30]

[134] G. B. DANTZIG AND M. N. THAPA, *Linear programming I: Introduction*, Springer, New York, NY, 1997. [5]

[135] G. B. DANTZIG AND P. WOLFE, *Decomposition principle for linear programs*, Oper. Res. **8**, no. 1 (1960), pp. 101–111. [65]

[136] T. A. DE QUEIROZ, F. K. MIYAZAWA, Y. WAKABAYASHI, AND E. C. XAVIER, *Algorithms for 3d guillotine cutting problems: Unbounded knapsack, cutting stock and strip packing*, Comput. Oper. Res. **39**, no. 2 (2012), pp. 200–212. [125]

[137] G. DOBSON, *Worst-case analysis of greedy heuristics for integer programming with nonnegative data*, Math. Oper. Res. **7**, no. 4 (1982), pp. 515–531. [22]

[138] R. G. DOWNEY AND M. R. FELLOWS, *Fixed-parameter tractability and completeness. I. Basic results*, SIAM J. Comput. **24**, no. 4 (1995), pp. 873–921. [44]

[139] I. S. DUFF AND J. K. REID, *The multifrontal solution of indefinite sparse symmetric linear*, ACM Trans. Math. Softw. **9**, no. 3 (1983), pp. 302–325. [103]

[140] I. DUKANOVIC AND F. RENDL, *Combinatorial tricks and Lovasz theta function applied to graph coloring*, in Proc. PATAT 2004, 2004, p. 479. [107]

[141] A. L. DULMAGE AND N. S. MENDELSOHN, *Two algorithms for bipartite graphs*, J. Soc. Indust. Appl. Math. **11** (1963), pp. 183–194. [68]

[142] R. E. ECHOLS AND L. COOPER, *Solution of integer linear programming problems by direct search*, J. ACM **15**, no. 1 (1968), pp. 75–84. [19, 21]

[143] J. ECKSTEIN, *Splitting methods for monotone operators with applications to parallel optimization*, PhD thesis, Massachusetts Institute of Technology. Dept. of Civil Engineering, Cambridge, MA, 1989. [110]

[144] J. ECKSTEIN AND D. P. BERTSEKAS, *On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators*, Math. Programming **55**, no. 3, Ser. A (1992), pp. 293–318. [110]

[145] J. ECKSTEIN AND M. NEDIAK, *Pivot, cut, and dive: a heuristic for 0-1 mixed integer programming*, J. Heuristics **13**, no. 5 (2007), pp. 471–503. [22]

[146] F. EISENBRAND AND A. S. SCHULZ, *Bounds on the Chvátal rank of polytopes in the* 0/1*-cube*, Combinatorica **23**, no. 2 (2003), pp. 245–261. [31]

[147] S. C. EISENSTAT, H. C. ELMAN, M. H. SCHULTZ, AND A. H. SHERMAN, *The (new) Yale Sparse Matrix Package*, in Elliptic problem solvers, II (Monterey, Calif., 1983), Academic Press, Orlando, FL, 1984, pp. 45–52. [103]

[148] P. ERDŐS, *Problem 9*, in Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963), M. Fiedler, ed., Publ. House Czechoslovak Acad. Sci., Prague, 1964, p. 159. [97]

[149] P. ERDŐS AND L. LOVÁSZ, *Problems and results on* 3*-chromatic hypergraphs and some related questions*, in Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vol. II, North-Holland, Amsterdam, 1975, pp. 609–627. Colloq. Math. Soc. János Bolyai, Vol. 10. [42]

[150] J. R. EVANS, *Structural analysis of local search heuristics in combinatorial optimization*, Comput. Oper. Res. **14**, no. 6 (1987), pp. 465–477. [25]

[151] S. EVEN, A. ITAI, AND A. SHAMIR, *On the complexity of timetable and multicommodity flow problems*, SIAM J. Comput. **5**, no. 4 (1976), pp. 691–703. [66]

[152] B. H. FAALAND AND F. S. HILLIER, *Interior path methods for heuristic integer programming procedures*, Oper. Res. **27**, no. 6 (1979), pp. 1069–1087. [19, 21]

[153] G. FASANO, *Cargo analytical integration in space engineering: A three-dimensional packing model*, in Operational Research in Industry, T. A. Ciriani, S. Gliozzi, E. L. Johnson, and R. Tadei, eds., Purdue University Press, 1999, pp. 232–246. [122]

[154] C. E. FERREIRA, A. MARTIN, AND R. WEISMANTEL, *Solving multiple knapsack problems by cutting planes*, SIAM J. Optim. **6**, no. 3 (1996), pp. 858–877. [69]

[155] M. FIEDLER, *A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory*, Czechoslovak Math. J. **25(100)**, no. 4 (1975), pp. 619–633. [51, 69, 85]

[156] R. M. FIGUEIREDO, M. LABBÉ, AND C. C. DE SOUZA, *An exact approach to the problem of extracting an embedded network matrix*, Comput. Oper. Res. **38**, no. 11 (2011), pp. 1483 – 1492. [62]

[157] M. FISCHETTI, F. GLOVER, AND A. LODI, *The feasibility pump*, Math. Programming **104**, no. 1 (2005), pp. 91–104. [21]

[158] M. FISCHETTI AND A. LODI, *Local branching*, Math. Programming **98**, no. 1 (2003), pp. 23–47. [25]

[159] M. FISCHETTI AND A. LODI, *Repairing MIP infeasibility through local branching*, Comput. Oper. Res. **35**, no. 5 (2008), pp. 1436–1445. [21, 25]

[160] M. FISCHETTI, A. LODI, AND D. SALVAGNIN, *Just mip it!*, in Matheuristics, V. Maniezzo, T. Stützle, and S. Voß, eds., Annals of Information Systems 10, Springer US, 2010, pp. 39–70. [86]

[161] M. FISCHETTI, C. POLO, AND M. SCANTAMBURLO, *A local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem*, Networks **44**, no. 2 (2004), pp. 61–72. [25]

[162] L. FLEISCHER, B. HENDRICKSON, AND A. PINAR, *On identifying strongly connected components in parallel*, in Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing, IPDPS '00, London, UK, UK, 2000, Springer-Verlag, pp. 505–511. [89]

[163] R. Fletcher, *Practical methods of optimization*, A Wiley-Interscience Publication, John Wiley & Sons Ltd., Chichester, second ed., 1987. [3, 117]

[164] J. Flum and M. Grohe, *Parameterized complexity theory*, Texts in Theoretical Computer Science. An EATCS Series, Springer-Verlag, Berlin, 2006. [44]

[165] J. J. Forrest and R. Lougee-Heimer, *CBC User Guide*, IBM Research and COIN-OR, 2005. [5]

[166] J. J. H. Forrest, J. P. H. Hirst, and J. A. Tomlin, *Practical solution of large mixed integer programming problems with UMPIRE*, Management Sci. **20** (1973/74), pp. 736–773. [28, 29, 30]

[167] R. Fourer, D. M. Gay, and B. W. Kernighan, *A modeling language for mathematical programming*, Manage. Sci. **36**, no. 5 (1990), pp. 519–554. [135]

[168] P. Frankl and V. Rödl, *Forbidden intersections*, Trans. Amer. Math. Soc. **300**, no. 1 (1987), pp. 259–286. [115]

[169] A. P. French, A. C. Robinson, and J. M. Wilson, *Using a hybrid genetic-algorithm/branch and bound approach to solve feasibility and optimization integer programming problems*, J. Heuristics **7**, no. 6 (2001), pp. 551–564. [25]

[170] A. Frieze and C. McDiarmid, *Algorithmic theory of random graphs*, Random Structures Algorithms **10**, no. 1-2 (1997), pp. 5–42. Average-case analysis of algorithms (Dagstuhl, 1995). [102]

[171] A. Fügenschuh and A. Martin, *Computational integer programming and cutting planes*, in Discrete Optimization, G. N. K. Aardal and R. Weismantel, eds., Handbooks in Operations Research and Management Science 12, Elsevier, 2005, pp. 69 – 121. [28]

[172] H. N. Gabow, *Path-based depth-first search for strong and biconnected components*, Inform. Process. Lett. **74**, no. 3-4 (2000), pp. 107–114. [89]

[173] T. Gallai, *Transitiv orientierbare Graphen*, Acta Math. Acad. Sci. Hungar **18** (1967), pp. 25–66. [103]

[174] G. Gamrath, *Generic branch-cut-and-price*, master's thesis, Technische Universität Berlin, 2010. [65]

[175] Y. Gao, *On the threshold of having a linear treewidth in random graphs*, in Computing and combinatorics, Lecture Notes in Comput. Sci. 4112, Springer, Berlin, 2006, pp. 226–234. [56]

[176] Y. Gao, *Treewidth of Erdös-Rényi random graphs, random intersection graphs, and scale-free random graphs*, CoRR **abs/0907.5481** (2009). [56]

[177] Y. Gao and J. Culberson, *On the treewidth of NK landscapes*, in Genetic and Evolutionary Computation Conference, 2003, pp. 948–954. [56]

[178] L. D. Gaspero and A. Schaerf, *Neighborhood portfolio approach for local search applied to timetabling problems*, J. Math. Model. Algorithms **5**, no. 1 (2006), pp. 65–89. [114]

[179] J.-M. Gauthier and G. Ribière, *Experiments in mixed-integer linear programming using pseudo-costs*, Math. Programming **12**, no. 1 (1977), pp. 26–47. [29, 30]

[180] B. Gavish and H. Pirkul, *Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality*, Math. Programming **31**, no. 1 (1985), pp. 63–89. [19]

[181] M. Gendreau, M. Iori, G. Laporte, and S. Martello, *A tabu search algorithm for a routing and container loading problem*, Transportation Science **40**, no. 3 (2006), pp. 342–350. [120, 129]

[182] A. M. Geoffrion, *Integer programming by implicit enumeration and Balas' method*, SIAM Rev. **9** (1967), pp. 178–190. [19, 28]

[183] A. M. Geoffrion, *The formal aspects of structured modeling*, Oper. Res. **37**, no. 1 (1989), pp. 30–51. [135]

[184] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal. **10** (1973), pp. 345–363. Collection of articles dedicated to the memory of George E. Forsythe. [68]

[185] A. George and D. R. McIntyre, *On the application of the minimum degree algorithm to finite element systems*, SIAM J. Numer. Anal. **15**, no. 1 (1978), pp. 90–112. [103]

[186] S. Ghosh, *DINS, a MIP improvement heuristic*, in Integer Programming and Combinatorial Optimization, M. Fischetti and D. P. Williamson, eds., Lecture Notes in Computer Science 4513, Springer, 2007, pp. 310–323. [25]

[187] S. Ghosh and R. Hayward, *Pivot and gomory cut: A 0-1 mixed integer programming heuristic*, tech. report, University of Alberta, 2006. [22]

[188] A. Ghouila-Houri, *Caractérisation des matrices totalement unimodulaires*, C. R. Acad. Sci. Paris **254** (1962), pp. 1192–1194. [58]

[189] A. Gilpin and T. Sandholm, *Information-theoretic approaches to branching in search*, Discrete Opt. **In Press** (2010), pp. –. [28, 29]

[190] W. Glankwamdee and J. Linderoth, *Lookahead branching for mixed integer programming*, Tech. Report 06T-004, Industrial and Systems Engineering, Lehigh University, 2006. [29]

[191] F. Glover, *A multiphase-dual algorithm for the zero-one integer programming problem*, Oper. Res. **13**, no. 6 (1965), p. 879. [22]

[192] F. Glover, *Creating network structure in lps*, in Proceedings of the 1st Symposium on Computer-Assisted Analysis and Model Simplification, Boulder, CO, 1980, H. J. Greenberg and J. S. Maybee, eds., Academic Press, 1981. [76]

[193] F. Glover, *Future paths for integer programming and links to artificial intelligence*, Comput. Oper. Res. **13**, no. 5 (1986), pp. 533–549. [20]

[194] F. Glover, *Adaptive memory projection methods for integer programming*, in Metaheuristic Optimization via Memory and Evolution, C. Rego and B. Alidaee, eds., Dordrecht, 2005, Kluwer Academic, pp. 425–440. [20]

[195] F. Glover, *Parametric tabu-search for mixed integer programs*, Comput. Oper. Res. **33**, no. 9 (2006), pp. 2449–2494. [20]

[196] F. Glover, *Infeasible/feasible search trajectories and directional rounding in integer programming*, J. Heuristics **13**, no. 6 (2007), pp. 505–541. [21]

[197] F. Glover, *Inequalities and target objectives for metaheuristic search – part i: Mixed binary optimization*, in Advances in Metaheuristics for Hard Optimization, P. Siarry and Z. Michalewicz, eds., Natural Computing Series, Springer, 2008, pp. 439–474. [20]

[198] F. GLOVER AND D. KLINGMAN, *The simplex SON algorithm for LP/embedded network problems*, Math. Programming Stud. **15** (1981), pp. 148–176. Network models and associated applications. [62]

[199] F. GLOVER AND M. LAGUNA, *General purpose heuristics for integer programming – Part I.*, J. Heuristics **2**, no. 4 (1997), pp. 343–358. [20, 21]

[200] F. GLOVER AND M. LAGUNA, *General purpose heuristics for integer programming – part ii*, J. Heuristics **3** (1997), pp. 161–179. [20, 21]

[201] A. V. GOLDBERG AND A. MARCHETTI-SPACCAMELA, *On finding the exact solution of a zero-one knapsack problem*, in Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, 1984, Washington, D.C., USA, ACM, 1984, pp. 359–368. [22]

[202] D. GOLDFARB AND S. MA, *Fast multiple splitting algorithms for convex optimization*, SIAM J Optim (2010). [110, 111]

[203] M. C. GOLUMBIC, *The complexity of comparability graph recognition and coloring*, Computing **18**, no. 3 (1977), pp. 199–208. [103]

[204] R. E. GOMORY, *Outline of an algorithm for integer solutions to linear programs*, Bull. Amer. Math. Soc. **64** (1958), pp. 275–278. [31]

[205] R. E. GOMORY, *Solving linear programming problems in integers*, in Proc. Sympos. Appl. Math., Vol. 10, American Mathematical Society, Providence, R.I., 1960, pp. 211–215. [31]

[206] R. E. GOMORY, *An all-integer programming algorithm*, in Industrial Scheduling, Prentice-Hall, Englewood Cliffs, N.J., 1963, pp. 193–206. [21]

[207] E. S. GOTTLIEB AND M. R. RAO, *The generalized assignment problem: valid inequalities and facets*, Math. Programming **46**, no. 1, (Ser. A) (1990), pp. 31–52. [69]

[208] G. W. GRAVES AND R. D. MCBRIDE, *The factorization approach to large-scale linear programming*, Math. Programming **10**, no. 1 (1976), pp. 91–110. [76]

[209] H. J. GREENBERG, *A bibliography for the development of an intelligent mathematical programming system*, Ann. Oper. Res. **65** (1996), pp. 55–90. [135]

[210] G. R. GRIMMETT AND C. J. H. MCDIARMID, *On colouring random graphs*, Math. Proc. Cambridge Philos. Soc. **77** (1975), pp. 313–324. [102]

[211] M. GRÖTSCHEL, *Personal communication, October 9, 2009. Combinatorial Optimization at Work II, Berlin.* [39]

[212] M. GRÖTSCHEL AND M. HENK, *The representation of polyhedra by polynomial inequalities*, Discrete Comput. Geom. **29**, no. 4 (2003), pp. 485–504. [41, 43]

[213] M. GRÖTSCHEL AND L. LOVÁSZ, *Combinatorial optimization*, in Handbook of combinatorics, Vol. 1, 2, Elsevier, Amsterdam, 1995, pp. 1541–1597. [18]

[214] M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER, *Geometric algorithms and combinatorial optimization*, Algorithms and Combinatorics: Study and Research Texts 2, Springer-Verlag, Berlin, 1988. [32]

[215] C. GUÉRET, C. PRINS, AND M. SEVAUX, *Applications of optimization with Xpress-MP*, Dash Optimization, Blisworth, UK, 2002. Translation from the French original Programmation linéaire published by Editions Eyrolles in Paris in 2000 by Susanne Heipcke. [5, 30]

[216] N. GÜLPINAR, G. GUTIN, G. MITRA, AND I. MAROS, *Detecting embedded networks in LP using GUB structures and independent set algorithms*, Comput. Optim. Appl. **15**, no. 3 (2000), pp. 235–247. [61]

[217] N. GÜLPINAR, G. GUTIN, G. MITRA, AND A. ZVEROVITCH, *Extracting pure network submatrices in linear programs using signed graphs*, Discrete Appl. Math. **137**, no. 3 (2004), pp. 359–372. [61, 63]

[218] N. GÜLPINAR, G. MITRA, AND I. MAROS, *Detecting embedded pure network structures in LP problems*, Top **6**, no. 1 (1998), pp. 67–95. [62]

[219] N. GÜLPINAR, G. MITRA, AND I. MAROS, *Creating advanced bases for large scale linear programs exploiting embedded network structure*, Comput. Optim. Appl. **21**, no. 1 (2002), pp. 71–93. [63]

[220] J. GUO AND R. NIEDERMEIER, *Invitation to data reduction and problem kernelization*, SIGACT News **38** (2007), pp. 31–45. [10, 44]

[221] N. GVOZDENOVIĆ, M. LAURENT, AND F. VALLENTIN, *Block-diagonal semidefinite programming hierarchies for 0/1 programming*, Oper. Res. Lett. **37**, no. 1 (2009), pp. 27–31. [33, 34, 41, 43, 132]

[222] S. GWON CHANG AND D. WAN TCHA, *A heuristic for multiple choice programming*, Comput. Oper. Res. **12**, no. 1 (1985), pp. 25–37. [21]

[223] M. HABIB AND M. C. MAURER, *On the X-join decomposition for undirected graphs*, Discrete Appl. Math. **1**, no. 3 (1979), pp. 201–207. [103]

[224] N. HALKO, P.-G. MARTINSSON, AND J. A. TROPP, *Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions*, SIAM Rev. **53**, no. 2 (2011), pp. 217–288. [117]

[225] M. M. HALLDÓRSSON AND G. KORTSARZ, *Multicoloring: Problems and techniques*, in Mathematical Foundations of Computer Science 2004, 29th International Symposium, MFCS 2004, Prague, Czech Republic, August 22-27, 2004, Proceedings, J. Fiala, V. Koubek, and J. Kratochvíl, eds., Lecture Notes in Computer Science 3153, Springer, 2004, pp. 25–41. [102]

[226] H. W. HAMACHER, *Consecutive ones decomposition of integer matrices and applications*, Electronic Notes in Discrete Mathematics **27** (2006), pp. 47 – 48. ODSA 2006 - Conference on Optimal Discrete Structures and Algorithms. [41, 43]

[227] S. HANAFI AND F. GLOVER, *Resolution search and dynamic branch-and-bound*, J. Comb. Optim. **6**, no. 4 (2002), pp. 401–423. [30]

[228] P. HANSEN, M. LABBÉ, AND D. SCHINDL, *Set covering and packing formulations of graph coloring: algorithms and first polyhedral results*, Discrete Optim. **6**, no. 2 (2009), pp. 135–147. [95]

[229] P. HANSEN, N. MLADENOVIĆ, AND D. UROSEVIĆ, *Variable neighborhood search and local branching*, Comput. Oper. Res. **33**, no. 10 (1985), pp. 3034–3045. [25]

[230] F. HARARY, *The number of linear, directed, rooted, and connected graphs*, Trans. Amer. Math. Soc. **78** (1955), pp. 445–463. [70]

[231] F. HARARY, *A matrix criterion for structural balance*, Naval Res. Logist. **7**, no. 2 (1960), pp. 195–199. [61]

[232] F. Harary and E. Palmer, *On the number of forests*, Mat. Časopis Sloven. Akad. Vied **19** (1969), pp. 110–112. [70]

[233] J. K. Hartman, *On rearranging matrices to block angular form (Management Sci. **18**(1971/72), 98–108) by Roman L. Weil and Paul C. Kettler*, Management Sci. **19**, no. 3 (1972), pp. 352–353. [68]

[234] U.-U. Haus, M. Köppe, and R. Weismantel, *The integral basis method for integer programming*, Math. Methods Oper. Res. **53**, no. 3 (2001), pp. 353–361. [21]

[235] U.-U. Haus, M. Köppe, and R. Weismantel, *A primal all-integer algorithm based on irreducible solutions*, Math. Program. **96**, no. 2, Ser. B (2003), pp. 205–246. Algebraic and geometric methods in discrete optimization. [21, 41]

[236] C. Helmberg, *Fixing variables in semidefinite relaxations*, SIAM J. Matrix Anal. Appl. **21**, no. 3 (2000), pp. 952–969. [20]

[237] L. Henrique Sacchi and V. Amaral Armentano, *A computational study of parametric tabu search for 0-1 mixed integer programs*, Comput. Oper. Res. **38**, no. 2 (2011), pp. 464–473. [20]

[238] F. S. Hillier, *Efficient heuristic procedures for integer linear programming with an interior*, Oper. Res. **17**, no. 4 (1969), pp. 600–637. [19, 21]

[239] E. A. Hirsch and D. Itsykson, *On optimal heuristic randomized semidecision procedures, with application to proof complexity*, in 27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010), J.-Y. Marion and T. Schwentick, eds., Leibniz International Proceedings in Informatics (LIPIcs) 5, Dagstuhl, Germany, 2010, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 453–464. [27]

[240] P. Hliněný, S. il Oum, D. Seese, and G. Gottlob, *Width parameters beyond tree-width and their applications*, Comput. J. **51**, no. 3 (2008), pp. 326–362. [55]

[241] P. Hliněný and S.-i. Oum, *Finding branch-decompositions and rank-decompositions*, SIAM J. Comput. **38**, no. 3 (2008), pp. 1012–1032. [50]

[242] A. J. Hoffman and J. B. Kruskal, *Integral boundary points of convex polyhedra*, in Linear inequalities and related systems, Annals of Mathematics Studies, no. 38, Princeton University Press, Princeton, N. J., 1956, pp. 223–246. [58]

[243] Z. Hua and F. Huang, *An effective genetic algorithm approach to large scale mixed integer programming problems*, Appl. Math. Comput. **174**, no. 2 (2006), pp. 897–909. [21]

[244] S. Huston, J. Puchinger, and P. J. Stuckey, *The core concept for 0/1 integer programming*, in Proceedings of the Fourteenth Australasian Theory Symposium, CATS 2008, J. Harland and P. Manyem, eds., 2008, pp. 39–47. [18, 39]

[245] F. Hutter, H. Hoos, and K. Leyton-Brown, *Automated configuration of mixed integer programming solvers*, in Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, A. Lodi, M. Milano, and P. Toth, eds., Lecture Notes in Computer Science 6140, Springer Berlin / Heidelberg, 2010, pp. 186–202. [8, 137]

[246] F. Hutter, H. Hoos, and K. Leyton-Brown, *Tradeoffs in the empirical evaluation of competing algorithm designs*, Ann. Math. Artif. Intel. (2010), pp. 1–25. [8, 137]

[247] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, *Paramils: an automatic algorithm configuration framework*, J. Artif. Int. Res. **36** (2009), pp. 267–306. [8, 137]

[248] T. Ibaraki, T. Ohashi, and H. Mine, *A heuristic algorithm for mixed integer programming problems*, Mathematical Programming Study **2** (1974), pp. 115–136. [21, 25]

[249] ILOG, *ILOG CPLEX Advanced Programming Techniques*, ILOG S. A., Incline Village, NV, 2006. [5]

[250] Ö. Johansson, *Clique-decomposition, NLC-decomposition, and modular decomposition—relationships and results for random graphs*, in Proceedings of the Twenty-ninth Southeastern International Conference on Combinatorics, Graph Theory and Computing (Boca Raton, FL, 1998), 1998, pp. 39–60. [56]

[251] , *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*, Boston, MA, 1996, American Mathematical Society. [113]

[252] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, *How easy is local search?*, J. Comput. System Sci. **37**, no. 1 (1988), pp. 79–100. 26th IEEE Conference on Foundations of Computer Science (Portland, OR, 1985). [23]

[253] E. L. Johnson, G. L. Nemhauser, and M. W. P. Savelsbergh, *Progress in linear programming-based algorithms for integer programming: An exposition.*, INFORMS J. Computing **12**, no. 1 (2000), pp. 2–23. [5]

[254] F. Juhász, *The asymptotic behaviour of Lovász' θ function for random graphs*, Combinatorica **2**, no. 2 (1982), pp. 153–155. [84, 94]

[255] L. Junqueira, R. Morabito, and D. S. Yamashita, *Three-dimensional container loading models with cargo stability and load bearing constraints*, Comput. Oper. Res. **39**, no. 1 (2012), pp. 74–85. [125]

[256] R. Kannan, *Minkowski's convex body theorem and integer programming*, Math. Oper. Res. **12**, no. 3 (1987), pp. 415–440. [41]

[257] R. Kannan and C. L. Monma, *On the computational complexity of integer programming problems*, in Optimization and operations research (Proc. Workshop, Univ. Bonn, Bonn, 1977), Lecture Notes in Econom. and Math. Systems 157, Springer, Berlin, 1978, pp. 161–172. [15]

[258] D. Karger, R. Motwani, and M. Sudan, *Approximate graph coloring by semidefinite programming*, J. ACM **45**, no. 2 (1998), pp. 246–265. [113, 116]

[259] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of computer computations (Proc. Sympos., IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y., 1972), Plenum, New York, 1972, pp. 85–103. [14, 15]

[260] G. Karypis and V. Kumar, *Analysis of multilevel graph partitioning*, in Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM), New York, NY, USA, 1995, ACM. [51]

[261] J. D. Kelly, *Smooth-and-dive accelerator: A pre-milp primal heuristic applied to scheduling*, Comput. Chem. Eng. **27**, no. 6 (2003), pp. 827–832. [21]

[262] J. P. KELLY, B. L. GOLDEN, AND A. A. ASSAD, *Large-scale controlled rounding using tabu search with strategic oscillation*, Ann. Oper. Res. **41**, no. 1-4 (1993), pp. 69–84. [20]

[263] D. KEMPE AND F. MCSHERRY, *A decentralized algorithm for spectral analysis*, J. Comput. System Sci. **74**, no. 1 (2008), pp. 70–83. [85]

[264] F. KILINÇ KARZAN, G. L. NEMHAUSER, AND M. W. P. SAVELSBERGH, *Information-based branching schemes for binary linear mixed integer problems*, Math. Program. Comput. **1**, no. 4 (2009), pp. 249–293. [8, 29, 137]

[265] I. M. KIMMO NIEMINEN, SAMPO RUUTH, *Genetic algorithm for finding a good first integer solution for milp*, Tech. Report 2003-4, Imperial College London Department of Computing, 2003. [25]

[266] T. KLOKS, *Treewidth*, Lecture Notes in Computer Science 842, Springer-Verlag, Berlin, 1994. [56]

[267] D. E. KNUTH, *The sandwich theorem*, Electron. J. Combin. **1** (1994), p. Article 1. [107]

[268] T. KOCH, *Rapid Mathematical Programming*, PhD thesis, Technische Universität Berlin, Berlin, 2004. ZIB Technical Report TR-04-58. [114, 135]

[269] A. M. C. A. KOSTER, A. ZYMOLKA, AND M. KUTSCHKA, *Algorithms to separate* $\{0, \frac{1}{2}\}$*-Chvátal-Gomory cuts*, Algorithmica **55**, no. 2 (2009), pp. 375–391. [32]

[270] B. KOTNYEK, *A generalisation of totally unimodular and network matrices*, PhD thesis, London School of Economics and Political Science, London, UK, 2002. [61]

[271] L. B. KOVÁCS, *A computer program system for the solution of pure linear discrete problems*, in Progress in operations research, Vols. I, II (Proc. Sixth Hungarian Conf., Eger, 1974), North-Holland, Amsterdam, 1976, pp. 573–592. Colloq. Math. Soc. János Bolyai, Vol. 12. [19]

[272] J. KRAJÍČEK AND P. PUDLÁK, *Propositional proof systems, the consistency of first order theories and the complexity of computations*, J. Symbolic Logic **54**, no. 3 (1989), pp. 1063–1079. [26]

[273] L. KUČERA, *The greedy coloring is a bad probabilistic algorithm*, J. Algorithms **12**, no. 4 (1991), pp. 674–684. [84]

[274] J. C. LAGARIAS, *The computational complexity of simultaneous Diophantine approximation problems*, in 23rd Annual Symposium on Foundations of Computer Science (Chicago, Ill., 1982), IEEE, New York, 1982, pp. 32–39. [14, 15, 18]

[275] A. LAND AND S. POWELL, *Computer codes for problems of integer programming*, Ann. Discrete Math. **5** (1979), pp. 221–269. Discrete optimization (Proc. Adv. Res. Inst. Discrete Optimization and Systems Appl., Banff, Alta., 1977), II. [28]

[276] A. LAND AND S. POWELL, *A survey of the operational use of ILP models*, Ann. Oper. Res. **149** (2007), pp. 147–156. [28]

[277] A. H. LAND AND A. G. DOIG, *An automatic method of solving discrete programming problems*, Econometrica **28** (1960), pp. 497–520. [16]

[278] J. B. LASSERRE, *An explicit exact SDP relaxation for nonlinear 0-1 programs*, in Integer programming and combinatorial optimization (Utrecht, 2001), Lecture Notes in Comput. Sci. 2081, Springer, Berlin, 2001, pp. 293–303. [33, 34, 41, 43, 132]

[279] E. L. LAWLER AND M. D. BELL, *A method for solving discrete optimization problems*, Oper. Res. **14**, no. 6 (1966), pp. 1098–1112. [19]

[280] C. LEE, J. LEE, AND S.-I. OUM, *Rank-width of random graphs*, J. Graph Theory (2011), p. to appear. [55]

[281] J. LEE, *All-different polytopes*, J. Comb. Optim. **6**, no. 3 (2002), pp. 335–352. [95]

[282] H. W. LENSTRA, JR., *Integer programming with a fixed number of variables*, Math. Oper. Res. **8**, no. 4 (1983), pp. 538–548. [15, 41, 43, 44]

[283] P. LIBERATORE, *On the complexity of choosing the branching literal in DPLL*, Artificial Intelligence **116**, no. 1-2 (2000), pp. 315–326. [27]

[284] J. T. LINDEROTH AND M. W. P. SAVELSBERGH, *A computational study of search strategies for mixed integer programming*, INFORMS J. Computing **11**, no. 2 (1999), pp. 173–187. [28, 29]

[285] J. W. H. LIU, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl. **11**, no. 1 (1990), pp. 134–172. [51]

[286] J. LÖFBERG, *Yalmip : A toolbox for modeling and optimization in MATLAB*, in Proc. of CACSD, Taipei, Taiwan, 2004. [107]

[287] A. LØKKETANGEN, *Heuristics for 0/1 mixed integer programming*, in Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende, eds., Oxford University Press, Oxford, UK, 2002. [19]

[288] A. LØKKETANGEN AND F. GLOVER, *Tabu search for zero-one mixed integer programming with advanced level strategies and learning*, Int. J. Oper. Quant. Manag. **1**, no. 2 (1995), pp. 89–108. [20]

[289] A. LØKKETANGEN AND F. GLOVER, *Solving zero-one mixed integer programming problems using tabu search*, European J. Oper. Res. **106**, no. 2-3 (1998), pp. 624–658. [20]

[290] A. LØKKETANGEN, K. JÖRNSTEN, AND S. STORØY, *Tabu search within a pivot and complement framework*, Int. Trans. Oper. Res. **1**, no. 3 (1994), pp. 305–316. [21]

[291] R. LOULOU AND E. MICHAELIDES, *New greedy-like heuristics for the multidimensional 0-1 knapsack problem.*, Oper. Res. **27**, no. 6 (1979), p. 1101. [19]

[292] L. LOVÁSZ, *Kneser's conjecture, chromatic number, and homotopy*, J. Combin. Theory Ser. A **25**, no. 3 (1978), pp. 319–324. [115]

[293] L. LOVÁSZ AND A. SCHRIJVER, *Cones of matrices and set-functions and* 0-1 *optimization*, SIAM J. Optim. **1**, no. 2 (1991), pp. 166–190. [33, 34, 41, 43, 132]

[294] M. LÜBBECKE, *Personal communication, August 27th, 2010.* [120, 129]

[295] G. LUEKER, *Two NP-complete problems in non-negative integer programming*, Tech. Report Report No. 178, Department of Computer Science, Princeton University, Princeton, N.J., 1975. [14, 15]

[296] Y.-C. LUO, M. GUIGNARD, AND C.-H. CHEN, *A hybrid approach for integer programming combining genetic algorithms, linear programming and ordinal optimization*, J. Intelligent Manufacturing **12**, no. 5 (2001), pp. 509–519. [25]

[297] A. MAHAJAN AND T. RALPHS, *On the complexity of selecting disjunctions in integer programming*, SIAM J. Optim. **20**, no. 5 (2010), pp. 2181–2198. [27, 28]

[298] J. MALICK, J. POVH, F. RENDL, AND A. WIEGELE, *Regularization methods for semidefinite programming*, SIAM J. Optim. **20**, no. 1 (2009), pp. 336–356. [110]

[299] J. W. MAMER AND R. D. MCBRIDE, *A decomposition-based pricing procedure for large-scale linear programs: An application to the linear multicommodity flow problem*, Management Science **46**, no. 5 (2000), pp. 693–709. [63, 69]

[300] H. MARCHAND, *A polyhedral study of the mixed knapsack set and its use to solve mixed integer programs*, PhD thesis, Université Catholique de Louvain, Louvain-la-Neuve, BE, 1997. [69]

[301] H. MARCHAND AND L. A. WOLSEY, *The* 0-1 *knapsack problem with a single continuous variable*, Math. Program. **85**, no. 1, Ser. A (1999), pp. 15–33. [69]

[302] S. MARGULIES, J. MA, AND I. HICKS, *The cunninghamgeelen method in practice: Branch-decompositions and integer programming*, tech. report, Rice University, 2011. [40, 43, 47, 50, 51, 52]

[303] R. E. MARSTEN AND T. L. MORIN, *Optimal solutions found for Senju and Toyoda's 0/1 integer programming problems*, Manage. Sci. **23** (1977), pp. 1364–1365. [20]

[304] M. J. MARTIN GRÖTSCHEL AND G. REINELT, *A cutting plane algorithm for the linear ordering problem*, Oper. Res. **32**, no. 6 (1984), pp. 1195–1220. [16]

[305] D. MARX, *Parameterized graph separation problems*, Theoret. Comput. Sci. **351**, no. 3 (2006), pp. 394–406. [67]

[306] D. MATULA AND L. KUČERA, *An expose-and-merge algorithm and the chromatic number of a random graph*, in Random graphs '87 (Poznań, 1987), Wiley, Chichester, 1990, pp. 175–187. [87, 100, 102]

[307] R. D. MCBRIDE, *Progress made in solving the multicommodity flow problem*, SIAM J. Optim. **8**, no. 4 (1998), pp. 947–955. [69]

[308] R. D. MCBRIDE AND J. W. MAMER, *Solving multicommodity flow problems with a primal embedded network simplex algorithm*, INFORMS J. Comput. **9**, no. 2 (1997), pp. 154–163. [69]

[309] F. MCSHERRY, *Spectral partitioning of random graphs*, in 42nd IEEE Symposium on Foundations of Computer Science (Las Vegas, NV, 2001), IEEE Computer Soc., Los Alamitos, CA, 2001, pp. 529–537. [85, 86, 88]

[310] A. MEHROTRA AND M. A. TRICK, *A column generation approach for graph coloring*, INFORMS J. Comput. **8**, no. 4 (1996), pp. 344–354. [95]

[311] I. MÉNDEZ-DÍAZ AND P. ZABALA, *A cutting plane algorithm for graph coloring*, Discrete Appl. Math. **156**, no. 2 (2008). [94, 95]

[312] F. MEYER AUF DER HEIDE, *Lower bounds for solving linear Diophantine equations on random access machines*, J. Assoc. Comput. Mach. **32**, no. 4 (1985), pp. 929–937. [9, 15]

[313] B. MÍŠEK, *On the number of classes of strongly equivalent incidence matrices*, Časopis Pěst. Mat. **89** (1964), pp. 211–218. [70]

[314] B. MÍŠEK, *Pólya's fundamental formula and incidence matrices*, in Theory of Graphs and its Applications (Proc. Sympos. Smolenice, 1963), Publ. House Czechoslovak Acad. Sci., Prague, 1964, pp. 137–141. [70]

[315] G. MITRA, *Investigation of some branch and bound strategies for the solution of mixed integer linear programs*, Math. Programming **4** (1973), pp. 155–170. [28]

[316] R. H. MÖHRING AND F. J. RADERMACHER, *Substitution decomposition for discrete structures and connections with combinatorial optimization*, in Algebraic and combinatorial methods in operations research, North-Holland Math. Stud. 95, North-Holland, Amsterdam, 1984, pp. 257–355. [103]

[317] R. J. MORAGA, G. W. DEPUY, AND G. E. WHITEHOUSE, *Meta-raps approach for the 0-1 multidimensional knapsack problem*, Computers Ind. Eng. **48**, no. 1 (2005), pp. 83 – 96. Selected Papers from the 31st. International Conference on Computers and Industrial Engineering. [19]

[318] E. MORENO, D. ESPINOZA, AND M. GOYCOOLEA, *Large-scale multi-period precedence constrained knapsack problem: A mining application*, Electronic Notes in Discrete Mathematics **36** (2010), pp. 407 – 414. ISCO 2010 - International Symposium on Combinatorial Optimization. [119]

[319] R. A. MOSER AND G. TARDOS, *A constructive proof of the general Lovász local lemma*, J. ACM **57**, no. 2 (2010), pp. Art. 11, 15. [42, 43, 132]

[320] J. H. MULLER AND J. SPINRAD, *Incremental modular decomposition*, J. ACM **36**, no. 1 (1989), pp. 1–19. [103, 104]

[321] A. MUSITELLI, *Recognition of generalised network matrices*, PhD thesis, Ecole Polytechnique Federale de Lausanne, Lausanne, CH, 2007. [61]

[322] A. MUSITELLI, *Recognizing binet matrices*, Math. Program. **124**, no. 1-2, Ser. B (2010), pp. 349–381. [61]

[323] G. L. NEMHAUSER, *Personal communication, August 2, 2007. Mixed integer programming workshop, Montreal, CA.* [137]

[324] G. L. NEMHAUSER AND L. A. WOLSEY, *Integer and combinatorial optimization*, John Wiley & Sons, New York, NY, 1988. [3]

[325] A. S. NEMIROVSKY AND D. B. YUDIN, *Problem complexity and method efficiency in optimization*, A Wiley-Interscience Publication, John Wiley & Sons Inc., New York, 1983. Translated from the Russian and with a preface by E. R. Dawson, Wiley-Interscience Series in Discrete Mathematics. [111]

[326] T. NETO AND J. P. PEDROSO, *Grasp for linear integer programming*, in Proceedings of the Fourth Metaheuristics International Conference, J. P. Sousa and M. G. C. Resende, eds., Porto, Portugal, 2001, pp. 377–385. [19]

[327] T. NETO AND J. P. PEDROSO, *Grasp for linear integer programming*, in METAHEURISTICS: Computer Decision-Making, J. P. Sousa and M. G. C. Resende, eds., Combinatorial Optimization Book Series, Kluwer Academic Publishers, 2003, pp. 545–574. [19]

[328] K. NIEMINEN, *Development of mixed integer programming methods*, master's thesis, Helsinki University of Technology, 2002. [25]

[329] V. NWANA, K. DARBY-DOWMAN, AND G. MITRA, *A co-operative parallel heuristic for mixed zero-one linear programming: combining simulated annealing with branch and bound*, European J. Oper. Res. **164**, no. 1 (2005), pp. 12–23. [19]

[330] M. OOSTEN, J. H. G. C. RUTTEN, AND F. C. R. SPIEKSMA, *Disconnecting graphs by removing vertices: a polyhedral approach*, Statist. Neerlandica **61**, no. 1 (2007), pp. 35–60. [68]

[331] J. B. ORLIN, A. P. PUNNEN, AND A. S. SCHULZ, *Approximate local search in combinatorial optimization*, SIAM J. Comput. **33**, no. 5 (2004), pp. 1201–1214. [24]

[332] S.-I. OUM, *Rank-width and vertex-minors*, J. Combin. Theory Ser. B **95**, no. 1 (2005), pp. 79–100. [48, 49]

[333] S.-I. OUM, *Rank-width is less than or equal to branch-width*, J. Graph Theory **57**, no. 3 (2008), pp. 239–244. [54]

[334] M. PADBERG, *Packing small boxes into a big box*, Math. Methods Oper. Res. **52**, no. 1 (2000), pp. 1–21. [122]

[335] M. PADBERG AND G. RINALDI, *A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems*, SIAM Rev. **33**, no. 1 (1991), pp. 60–100. [16]

[336] Y. PAN AND L. SHI, *On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems*, Math. Program. **110**, no. 3, Ser. A (2007), pp. 543–559. [130, 133]

[337] C. H. PAPADIMITRIOU, *On the complexity of integer programming*, J. Assoc. Comput. Mach. **28**, no. 4 (1981), pp. 765–768. [15, 22, 44]

[338] C. H. PAPADIMITRIOU, *Worst-case and probabilistic analysis of a geometric location problem*, SIAM J. Comput. **10**, no. 3 (1981), pp. 542–557. [130]

[339] C. H. PAPADIMITRIOU, *On the complexity of unique solutions*, J. Assoc. Comput. Mach. **31**, no. 2 (1984), pp. 392–400. [128]

[340] A. J. PARKES, *Personal communication.* [135]

[341] G. PATAKI, M. TURAL, AND E. B. WONG, *Basis reduction and the complexity of branch-and-bound*, in Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '10, Philadelphia, PA, USA, 2010, Society for Industrial and Applied Mathematics, pp. 1254–1261. [41, 43, 133]

[342] J. PATEL AND J. W. CHINNECK, *Active-constraint variable ordering for faster feasibility of mixed integer linear programs*, Math. Programming **110**, no. 3, Ser. A (2007), pp. 445–474. [20, 28, 29]

[343] J. P. PEDROSO, *An evolutionary solver for linear integer programming*, BSIS Technical Report 98-7, Riken Brain Science Institute, Wako-shi, Saitama, Japan, 1998. [19]

[344] J. P. PEDROSO, *An evolutionary solver for pure integer linear programming*, Working Paper 5/99, Centro de Investigação Operacional da Universidade de Lisboa, Centro de Investigação Operacional, Faculdade de Ciências da Universidade de Lisboa, 1749-016 Lisboa, Portugal, 1999. [19]

[345] J. P. PEDROSO, *An evolutionary solver for pure integer linear programming*, Int. Trans. Oper. Res. **9**, no. 3 (2002), pp. 337–352. [19]

[346] J. P. PEDROSO, *Tabu search for linear integer programming.* MIC'2003 – the Fifth Meta-heuristics International Conference, August 2003. Kyoto, Japan. [19]

[347] J. P. PEDROSO, *Hybrid enumeration strategies for mixed integer programming*, Tech. Report DCC-2004-08, LIACC, Universidade do Porto, 2004. [19]

[348] J. P. PEDROSO, *Tabu search for mixed integer programming*, Tech. Report DCC-2004-02, LIACC, Universidade do Porto, 2004. [19]

[349] J. F. PIERCE, *Application of combinatorial programming to a class of all-zero-one integer programming problems*, Manage. Sci. **15**, no. 3 (1968), p. 191. [19]

[350] H. PIRKUL, *A heuristic solution procedure for the multiconstraint zero-one knapsack problem*, Naval Res. Logist. **34**, no. 2 (1987), pp. 161–172. [30]

[351] D. PISINGER, *An expanding-core algorithm for the exact 0-1 knapsack problem*, European J. Oper. Res. **87**, no. 1 (1995), pp. 175–187. [18]

[352] T. PITASSI AND N. SEGERLIND, *Exponential lower bounds and integrality gaps for tree-like lovász-schrijver procedures*, in SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms, Philadelphia, PA, USA, 2009, Society for Industrial and Applied Mathematics, pp. 355–364. [33]

[353] L. PITSOULIS, K. PAPALAMPROU, G. APPA, AND B. KOTNYEK, *On the representability of totally unimodular matrices on bidirected graphs*, Discrete Math. **309**, no. 16 (2009), pp. 5024–5042. [61]

[354] G. PÓLYA, *Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen.*, Acta Math. (Uppsala) **68** (1937), pp. 145–254. [70]

[355] J. POVH, F. RENDL, AND A. WIEGELE, *A boundary point method to solve semidefinite programs*, Computing **78**, no. 3 (2006), pp. 277–286. [110]

[356] J. PUCHINGER, *Combining metaheuristics and integer programming for solving cutting and packing problems*, PhD thesis, Vienna University of Technology Institute of Computer Graphics and Algorithms, 2006. [20]

[357] J. PUCHINGER AND G. R. RAIDL, *Relaxation guided variable neighborhood search*, in Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search, Tenerife, Spain, 2005. [20]

[358] J. PUCHINGER AND G. R. RAIDL, *Bringing order into the neighborhoods: Relaxation guided variable neighborhood search*, J. Heuristics (in press), pp. 1–16. [20]

[359] P. PUDLÁK, *The lengths of proofs*, in Handbook of proof theory, Stud. Logic Found. Math. 137, North-Holland, Amsterdam, 1998, pp. 547–637. [26]

[360] P. RAGHAVAN AND C. D. TOMPSON, *Randomized rounding: a technique for provably good algorithms and algorithmic proofs*, Combinatorica **7**, no. 4 (1987), pp. 365–374. [19]

[361] K. RANA, *A decomposition technique for mixed integer programming problems*, Comput. Oper. Res. **19**, no. 6 (1992), pp. 505–519. [21]

[362] P. RICHTÁRIK, *Approximate level method*, CORE Discussion Papers 2008083, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), Dec. 2008. [132]

[363] N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. IV. Tree-width and well-quasi-ordering*, J. Combin. Theory Ser. B **48**, no. 2 (1990), pp. 227–254. [47]

[364] N. ROBERTSON AND P. D. SEYMOUR, *Graph minors. X. Obstructions to tree-decomposition*, J. Combin. Theory Ser. B **52**, no. 2 (1991), pp. 153–190. [47, 48, 49, 54]

[365] C. RÖSSNER AND J.-P. SEIFERT, *Approximating good simultaneous diophantine approximations is almost NP-hard*, in Mathematical Foundations of Computer Science 1996, W. Penczek and A. Szalas, eds., Lecture Notes in Computer Science 1113, Springer, 1996, pp. 494–505. [17]

[366] E. ROTHBERG AND B. HENDRICKSON, *Sparse matrix ordering methods for interior point linear programming*, INFORMS J. Comput. **10**, no. 1 (1998), pp. 107–113. [68, 69]

[367] E. E. ROTHBERG, *An evolutionary algorithm for polishing mixed integer programming solutions*, INFORMS J. Computing **19**, no. 4 (2007), pp. 534–541. [25]

[368] G. SABIDUSSI, *Graph derivatives*, Math. Z. **76** (1961), pp. 385–401. [103]

[369] Z. SADOWSKI, *On an optimal propositional proof system and the structure of easy subsets of TAUT*, Theoret. Comput. Sci. **288**, no. 1 (2002), pp. 181–193. Complexity and logic (Vienna, 1998). [27]

[370] R. M. SALTZMAN AND F. S. HILLIER, *A heuristic ceiling point algorithm for general integer linear programming*, Manage. Sci. **38**, no. 2 (1992), pp. 263–283. [20]

[371] D. SALVAGNIN, *Constraint Programming Techniques for Mixed Integer Linear Programs*, PhD thesis, Università degli studi di Padova, 2009. [16]

[372] F. SANTOS AND B. STURMFELS, *Higher Lawrence configurations*, J. Combin. Theory Ser. A **103**, no. 1 (2003), pp. 151–164. [41]

[373] H. E. SCARF, *An observation on the structure of production sets with indivisibilities*, Proc. Nat. Acad. Sci. U.S.A. **74**, no. 9 (1977), pp. 3637–3641. [42]

[374] A. SCHRIJVER, *Theory of linear and integer programming*, Wiley-Interscience Series in Discrete Mathematics, John Wiley & Sons Ltd., Chichester, 1986. A Wiley-Interscience Publication. [15, 58, 60, 63, 133]

[375] A. S. SCHULZ, *On the relative complexity of 15 problems related to 0/1-integer programming*, in Research trends in combinatorial optimization, Springer, Berlin, 2009, pp. 399–428. [24, 31, 32, 132]

[376] D. G. SEESE, *Entscheidbarkeits- und Interpretierbarkeitsfragen Monadischer Theorien zweiter Stufe gewisser Klassen von Graphen*, PhD thesis, Humboldt-Universität zu Berlin, Berlin, 1976. [47]

[377] S. SENJU AND Y. TOYODA, *An approach to linear programming with 0-1 variables*, Manage. Sci. **15**, no. 4 (1968), pp. 196–207. [20]

[378] P. D. SEYMOUR, *Decomposition of regular matroids*, J. Combin. Theory Ser. B **28**, no. 3 (1980), pp. 305–359. [58, 60, 133]

[379] P. D. SEYMOUR, *Applications of the regular matroid decomposition*, in Matroid theory (Szeged, 1982), Colloq. Math. Soc. János Bolyai 40, North-Holland, Amsterdam, 1985, pp. 345–357. [58, 59]

[380] P. D. SEYMOUR AND R. THOMAS, *Call routing and the ratcatcher*, Combinatorica **14**, no. 2 (1994), pp. 217–241. [49]

[381] C. E. SHANNON, *Game playing machines*, J. Franklin Inst. **260** (1955), pp. 447–453. [53]

[382] L. SHAPLEY, *On committees*, in New Methods of Thought and Procedure, Springer, Berlin, 1967, pp. 246–270. [103]

[383] A. A. SHNEIDER, *Classification analysis of heuristic algorithms for graph coloring*, Cybernetics and Systems Analysis **20** (1984), pp. 484–492. [102]

[384] J. P. SOUSA AND L. A. WOLSEY, *A time indexed formulation of non-preemptive single machine scheduling problems*, Mathematical Programming **54** (1992), pp. 353–367. [130]

[385] A. SRINIVASAN, *An extension of the Lovász local lemma, and its applications to integer programming*, SIAM J. Comput. **36**, no. 3 (2006), pp. 609–634. [19, 22, 42]

[386] D. V. STEWARD, *On an approach to techniques for the analysis of the structure of large systems of equations*, SIAM Rev. **4** (1962), pp. 321–342. [68]

[387] G. W. STEWART, *On the early history of the singular value decomposition*, SIAM Rev. **35**, no. 4 (1993), pp. 551–566. [111]

[388] R. TARJAN, *Depth-first search and linear graph algorithms*, SIAM J. Comput. **1**, no. 2 (1972), pp. 146–160. [89]

[389] R. E. TARJAN, *Decomposition by clique separators*, Discrete Math. **55**, no. 2 (1985), pp. 221–232. [48]

[390] J. R. TEBBOTH, *A Computational Study of Dantzig-Wolfe Decomposition*, PhD thesis, University of Buckingham, 2001. [69]

[391] R. R. THOMAS, *A geometric Buchberger algorithm for integer programming*, Math. Oper. Res. **20**, no. 4 (1995), pp. 864–884. [41]

[392] C. THOMASSEN, *Embeddings and minors*, in Handbook of combinatorics, Vol. 1, 2, Elsevier, Amsterdam, 1995, pp. 301–349. [48]

[393] Y. TOYODA, *A simplified algorithm for obtaining approximate solutions to zero-one programming problems*, Managem. Sci. **21**, no. 12 (1975), pp. 1417–1427. [20]

[394] L. N. TREFETHEN AND D. BAU, III, *Numerical linear algebra*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997. [119]

[395] W. T. TUTTE, *A homotopy theorem for matroids. I, II*, Trans. Amer. Math. Soc. **88** (1958), pp. 144–174. [59]

[396] W. T. TUTTE, *An algorithm for determining whether a given binary matroid is graphic*, Proc. Amer. Math. Soc. **11** (1960), pp. 905–917. [61]

[397] E. ULUSAL, *Integer Programming Models for the Branchwidth Problem*, PhD thesis, Texas A&M University, Austin, TX, 2008. [50]

[398] R. URBANIAK, R. WEISMANTEL, AND G. M. ZIEGLER, *A variant of the Buchberger algorithm for integer programming*, SIAM J. Discrete Math. **10**, no. 1 (1997), pp. 96–108. [41]

[399] J. M. van den Akker, *LP-based solution methods for single-machine scheduling problems*, Technische Universiteit Eindhoven, Eindhoven, 1994. Dissertation. [130]

[400] P. Van Hentenryck, *OPL: A Modeling Language for Combinatorial Optimization*, MIT Press, Cambridge, MA, 1999. [10, 135, 136]

[401] S. P. M. van Hoesel and B. Marchal, *Finding good tree decompositions by local search*, Electronic Notes in Discrete Mathematics **32** (2009), pp. 43–50. [51]

[402] M. Vasquez and J.-K. Hao, *A hybrid approach for the 0-1 multidimensional knapsack problem*, in Proceedings of the 17th international joint conference on Artificial intelligence - Volume 1, San Francisco, CA, USA, 2001, Morgan Kaufmann Publishers Inc., pp. 328–333. [20]

[403] V. Vassilev and K. Genova, *An algorithm of internal feasible directions for linear integer programming*, European J. Oper. Res. **52**, no. 2 (1991), pp. 203–214. [21]

[404] A. F. Veinott, Jr. and G. B. Dantzig, *Integral extreme points*, SIAM Rev. **10** (1968), pp. 371–372. [57]

[405] Y. Vimont, S. Boussier, and M. Vasquez, *Reduced costs propagation in an efficient implicit enumeration for the 0-1 multidimensional knapsack problem*, J. Comb. Optim. **15**, no. 2 (2008), pp. 165–178. [20]

[406] J. P. Walser, *Integer optimization by local search*, Lecture Notes in Computer Science 1637, Springer-Verlag, Berlin, 1999. A domain-independent approach, With a foreword by Henry Kautz, Lecture Notes in Artificial Intelligence. [20, 23, 42]

[407] D. M. Warme, *Spanning trees in hypergraphs with applications to Steiner trees*, PhD thesis, University of Virginia, Charlottesville, VA, USA, 1998. [64]

[408] R. L. Weil and P. C. Kettler, *Rearranging matrices to block-angular form for decomposition (and other) algorithms*, Management Sci. **18** (1971/72), pp. 98–108. [68]

[409] Z. Wen, D. Goldfarb, and W. Yin, *Alternating direction augmented lagrangian methods for semidefinite programming*, Math Program Comput (2010). [110, 111]

[410] R. Wessäly, S. Orlowski, A. Zymolka, A. M. C. A. Koster, and C. Gruber, *Demand-wise Shared Protection revisited: A new model for survivable network design*, in Proceedings of INOC 2005, Lisbon, Portugal, 2005, International Network Optimization Conference, pp. 100–105. [25]

[411] C. Wilbaut and S. Hanafi, *New convergent heuristics for 0-1 mixed integer programming*, European J. Oper. Res. (2008). [22]

[412] H. P. Williams, *Personal communication, June 30, 2009. Workshop in the honour of Professor Williams' career and retirement, London.* [57]

[413] H. P. Williams and H. Yan, *Representations of the all_different predicate of constraint satisfaction in integer programming*, INFORMS J. Comput. **13**, no. 2 (2001), pp. 96–103. [95]

[414] , *Handbook of semidefinite programming*, International Series in Operations Research & Management Science, 27, Boston, MA, 2000, Kluwer Academic Publishers. [32]

[415] L. A. Wolsey, *Integer programming*, Wiley, New York, NY, 1998. [3]

[416] R. Wunderling, *Personal communication, July 27, 2010. Mixed integer programming workshop, Atlanta, GA.* [81]

[417] R. Yokoyama, Y. Hasegawa, and K. Ito, *A milp decomposition approach to large scale optimization in structural design of energy supply systems*, Energy Conversion and Management **43**, no. 6 (2002), pp. 771–790. [21]

[418] P. Zabala and I. Méndez-Díaz, *A branch-and-cut algorithm for graph coloring*, Discrete Appl. Math. **154**, no. 5 (2006), pp. 826–847. [94, 95]

[419] S. H. Zanakis, *Heuristic 0-1 linear programming: An experimental comparison of three methods*, Manage. Sci. **24**, no. 1 (1977), pp. 91–104. [19]

[420] A. Zanette, M. Fischetti, and E. Balas, *Can pure cutting plane algorithms work?*, in Integer programming and combinatorial optimization, Lecture Notes in Comput. Sci. 5035, Springer, Berlin, 2008, pp. 416–434. [40, 43]

[421] E. Zemel, *Probabilistic analysis of geometric location problems*, SIAM J. Algebraic Discrete Methods **6**, no. 2 (1985), pp. 189–200. [130]

[422] X.-Y. Zhao, D. Sun, and K.-C. Toh, *A Newton-CG augmented Lagrangian method for semidefinite programming*, SIAM J. Optim. **20**, no. 4 (2010), pp. 1737–1765. [110]

[423] D. Zuckerman, *Linear degree extractors and the inapproximability of max clique and chromatic number*, Theory Comput. **3** (2007), pp. 103–128. [82]

[424] A. A. Zykov, *On some properties of linear complexes*, Mat. Sbornik N.S. **24(66)** (1949), pp. 163–188. [102]