Carnegie Mellon University

Tepper School of Business

Doctoral Dissertation

New Techniques for Discrete Optimization

David Bergman

March, 2013

Submitted to the Tepper School of Business in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Algorithms, Combinatorics, and Optimization

Dissertation Committee:

J.N. Hooker (co-chair)

Willem-Jan van Hoeve (co-chair)

R. Ravi

Tuomas Sandholm

Acknowledgements

Perhaps the most rewarding part of completing this dissertation is reflecting back on the support that I have received during the course of my doctoral work.

I would like to begin by thanking my advisors, John Hooker and Willem-Jan van Hoeve. I will carry the advice you have given me throughout my career. It has been a great pleasure to work with such high-caliber researchers and I thank you for all of the time and support you have given me throughout my candidacy.

I would also like to thank the many other professors at Carnegie Mellon who have played instrumental roles in my success. R. Ravi and Tuomas Sandholm for serving on my thesis committee and offering advice for my research. In addition, I would like to extend a thank you to all of the faculty members in the Operations Research Department at the Tepper School of Business; Egon Balas, Gérard Cornuéjols, Fatma Kilinç-Karzan, François Margot, Javier Peña, and to Michael Trick, thank you for all of the helpful career advice and suggestions for future work. Furthermore, I would like to express my gratitude to Alan Scheller-Wolf for providing me with much needed guidance as I enter the next stage in my career. Finally, I would like to thank the two co-chairs of my committee, John Hooker and Willem-Jan van Hoeve, R.Ravi, and Louis-Martin Rousseau for helping me in my career search by writing letters of recommendation on my behalf.

It has been a great pleasure to learn and develop my research agenda with my fellow Operations Research and Algorithm, Combinatorics, and Optimization students. Yogesh Awate, Negar Soheili Azad, Deepak Bal, Amitabh Basu, Patrick Bennett, Andre Cire, Stylianos Despotakis, Lisa Espig, Samid Hoda, Jeremy Karp, Aleksandr Kazachkov, Brian Kell, Selvaprabu Nadarajah, Viswanath Nagarajan, Afshin Nikzad, Benjamin Peterson,

Andrea Qualizza, Amin Sayedi, Marla Slusky, Christian Tjandraamadja, Babis Tsourakakis, Nan Xiong and Sercan Yildiz, thanks to all of you! I would also like to thank the students that entered the Ph.D. program at Tepper with me in 2008. Ishani Aggarwal, Alp Aklay, Steven Baker, Dario Cestau, Henry Chong, Kevin Chung, Xin Fang, Majid Gholamzadeh, Qihang Lin, Marco Molinaro, Sivie Naimer, Maxime Roy, Batchimeg Sambalaibat, David Schreindorfer, Nazli Turan, Hao Xue, and Ran Zhao, it was a pleasure working alongside you throughout the years.

A special thanks to the Tepper Ph.D. Student Services Assistant Director, Lawrence Rapp. Tepper is lucky to have you; you always go above and beyond to make sure that students are taken care of and can produce the best work possible.

I have had many friends during my years in Pittsburgh that helped me along the way. To my friends who served in the Graduate Student Assembly with me, Will and Emily Boney, Grace Heckmann, Patrick Foley, Ashlie Henery, Jason Imbrogno, Amelia Kriss, Ruth Poproski, Erica Sampson, Carolyn Norwood, Andy Schultz, and Nancy Stiger, it was great getting to know you and serving the University with you. And thank you to Suzie Laurich-Mcintyre and Kaycee Palko for constant support and dedication to the Graduate Student Assembly.

To Andre Cire, thank you for being such a great friend and co-author. From the classes we have taken, to traveling the world together, conference by conference, to spinning classes, to working long hours at Tepper; I have had a great time getting to know you and working with you. I consider the majority of the work in this thesis a result of our collaboration and I look forward to working with you in the future.

To Jason Imbrogno, Rachel Goffman, and Rachel Levy, thanks for being such great friends. Having your support and friendship outside of coursework and research has helped me more than you know.

And to the Zafris family, Ryan, Wendy, Finnegan, and Preston, thank you for giving me a home away from home. The good times we have shared together and the way that you open your home to me is something I will never forget.

To Andrew and Beverly, thank you for being such a wonderful part of my life. You are always there for my family and I am honored to call you my aunt and uncle.

Daniel and Mandy Buckley, words are not enough to describe my gratitude to you. I have learned so much from you, reached out to you for help in times of need, and have shared the best of times with you. Thank you for always being there for me and exemplifying what it means to be a best friend.

I would like to especially thank my girlfriend Lauren. You have helped me concentrate on writing this dissertation (more than you even know), and without your love and support I would not have been able to complete it! You always make me very happy and I truly appreciate everything you do for me!

And last but not least, a thanks to my family. Jacob and Jeff, better brothers do not exist. Jacob, the advice that you have given me and continue to offer me has proven to be indispensable. Having you as a role model has truly shaped my life and without you I wouldn't have been able to grow into the person I am today. Jeff, living with you in Pittsburgh at the beginning of my Ph.D. made my first two years here a blast. Whenever I need anything you drop whatever you are doing to help me out and I thank you for being such a great brother. I am proud of both of you and will never take having such great brothers for granted. Jenn, thank you for not only being so great to my brothers but also for helping me out whenever I need anything. I could not ask for a better sister-in-law.

And finally, to my parents, it is not just the support and guidance that you give me that makes you the best parents in the world, it is your undying devotion to me that sets you apart. With every mistake, you lift me up. With every accomplishment, you are there to celebrate with me. Thank you for everything, and I dedicate this dissertation to you.

Preface

The work in this dissertation is the result of research done in collaboration with my two co-chairs, John Hooker and Willem-Jan van Hoeve, a fellow Ph.D. student Andre Cire, and Tallys Yunes.

The work presented in Chapters 2-5 is the expansion of ideas developed during the writing of my second summer paper at Carnegie Mellon University. Part of the work presented here can be found in [12, 9, 10].

The material in Chapter 6 is the result of the expansion of work presented in my first summer paper at Carnegie Mellon University. A preliminary version of the work can be found in [11].

Abstract

Objective function bounds are critical to solving discrete optimization problems. In this thesis, we explore several new techniques for obtaining bounds, and also describe a new branch-and-bound algorithm for discrete optimization problems.

The first area explored is the application of decision diagrams (DDs) to binary optimization problems. In recent years, DDs have been applied to various problems in operations research. These include sequential pattern data mining, cut generation, product configuration, and post-optimality analysis in integer programming, to name a few. Through these applications and others, DDs have proven to be a useful tool for a variety of tasks. Unfortunately, DDs may grow exponentially large, prohibiting their use.

To overcome this difficulty, we introduce the notion of limited-width approximate decision diagrams for discrete optimization problems. By limiting the width of a decision diagram, the size of the data structure can be controlled to the level of accuracy desired.

Discussed in this dissertation is the use of approximate DDs as problem relaxations and restrictions of the feasible set. We introduce top-down compilation algorithms for approximate DDs and discuss how they can be used to generate both upper and lower bounds on the optimal value for any separable objective function.

We then discuss how relaxed and restricted DDs can be used together to create a DD-based branch-and-bound algorithm. The algorithm differs substantially from traditional branch-and-bound algorithms on this class of problems in several important ways. First, relaxed DDs provide a discrete relaxation, as opposed to a continuous relaxation (for example a linear programming relaxation), which is typically employed. In addition, subproblems are generated by branching on several partial solutions taken at once, thereby eliminating

certain symmetry from the search. We discuss the application of the algorithm to the classical maximum independent set problem. Computational results show that the algorithm is competitive with state-of-the-art integer programming technology.

The next area explored is the idea of obtaining valid inequalities for a 0-1 model from a constraint programming formulation of the problem. In particular, we formulate a graph coloring problem as a system of all-different constraints. By analyzing the polyhedral structure of all-different systems, we obtain facet-defining inequalities that can be mapped to valid cuts in the classical 0-1 model of the problem.

We employ a common strategy for generating problem-specific cuts: the identification of facet-defining cuts for special types of induced subgraphs, such as odd-holes, webs, and paths. We identify cuts that bound the objective function as well as cuts that exclude infeasible solutions.

One structure that we focus on is cyclic structures and show that the cuts we obtain are stronger than previously known cuts. For example, when an existing separation algorithm identifies odd-hole cuts, we can supply stronger cuts with no additional calculation. In addition, we generalize odd-hole cuts to odd-cycle cuts that are stronger than any collection of odd-hole cuts. We also identify cuts associated with intersecting systems, for which there are no previously known 0-1 cuts, to the best of our knowledge.

Contents

Acknowledgements **3**

Preface **7**

Abstract **9**

List of Figures **15**

List of Tables **19**

1 Introduction **21**

2 Relaxation Decision Diagrams **29**

2.1 Introduction 29

2.2 Previous Work 30

2.3 Binary Decision Diagrams 31

2.4 BDD Representation of Independent Sets 33

2.5 Exact and Relaxed BDDs 34

2.6 Exact BDD Compilation 35

2.7 Relaxed BDDs 38

2.8 Merging Heuristics 42

2.9 Variable Ordering 43

2.10 Computational Experiments 50

2.11 Conclusions 63

3 Tightening Bounds from Relaxed Decision Diagrams 65

3.1 Introduction 65
3.2 Preliminaries 67
3.3 Top-Down MDD Compilation 69
3.4 Value Enumeration 73
3.5 Application to Set Covering 74
3.6 Computational Results 78
3.7 Conclusion 83

4 Restriction Decision Diagrams 85

4.1 Introduction 85
4.2 Binary Decision Diagrams 87
4.3 Exact BDDs 89
4.4 Restricted BDDs 94
4.5 Applications 96
4.6 Computational Experiments 103
4.7 Conclusion 112

5 Decision Diagram-Based Branch and Bound 115

5.1 Introduction 115
5.2 Binary Optimization Problems 117
5.3 Binary Decision Diagrams 119
5.4 Exact BDDs 126
5.5 Approximate BDDs 134
5.6 Branching via BDDs 142
5.7 Branch and Bound 149
5.8 Computational Results 152
5.9 Conclusions and Future Work 159

6 Finite-Domain Cuts for Graph Coloring 161

6.1 Introduction 161
6.2 The Problem 163

6.3 Previous Work 165
6.4 Mapping into 0-1 Space 166
6.5 General Properties of the Polytope 168
6.6 Cycles 175
6.7 Combs 189
6.8 Webs 196
6.9 Paths 200
6.10 Intersecting Systems 205
6.11 Computational Results 211
6.12 Conclusion 216

7 Conclusions **219**

References **223**

List of Figures

2.1 An exact and relaxed BDD for the family of independent sets 32

2.2 Comparing Exact BDD for a Path Graph with 2 Different Orderings 44

2.3 Bound quality vs. graph density for each merging heuristic, using the `random` instance set with `MPD` ordering and maximum BDD width 10 51

2.4 Bound quality vs. graph density for different variable ordering heuristics . . . 52

2.5 Relaxation bound vs. maximum BDD width for **dimacs** instance `p-hat_300-1` 53

2.6 Bound quality vs. graph density for **random** instances, showing results for LP only, LP plus cutting planes, and BDDs with maximum width 100, 1000, and 10000 54

2.7 Bound quality vs. graph density for **dimacs** instances, showing results for LP only, LP plus cutting planes, and BDDs with maximum width 100, 1000, and 10000 55

2.8 Bound quality for an LP relaxation (no cuts) vs. width 100 BDDs for **random** instances 55

2.9 Bound quality for an LP relaxation with cuts vs. width 1000 BDDs for **random** instances 55

2.10 Bound quality for an LP relaxation with cuts vs. width 10000 BDDs for **random** instances 56

2.11 Bound quality for an LP relaxation (no cuts) vs. width 100 BDDs for **dimacs** instances 56

2.12 Bound quality for an LP relaxation with cuts vs. width 1000 BDDs for **dimacs** instances . . . 57
2.13 Bound quality for an LP relaxation with cuts vs. width 10000 BDDs for **dimacs** instances . . . 57

3.1 Exact MDD for the feasible set of a set covering instance . . . 69
3.2 Relaxed MDD construction for the set covering problem . . . 76
3.3 Solution times for the set covering problem for different widths and varying number of rounds of filtering . . . 80
3.4 Comparing integer programming and value enumeration for instances with $n = 250$ . . . 81
3.5 Comparing number of instances solved via integer programming and value enumeration for instances with $n = 500$ . . . 82
3.6 Comparing lover bounds via integer programming and value enumeration for instances with $n = 500$ . . . 83

4.1 Reduced BDD for a binary optimization problem . . . 89
4.2 Restriction BDD for a binary optimization problem . . . 95
4.3 Exact and restricted BDD for a set covering problem . . . 99
4.4 Exact and restricted BDD for a set packing problem . . . 102
4.5 Restricted BDD performance versus the maximum allotted width for a set covering instance . . . 106
4.6 Average gap difference for set covering instances (varying bandwidth) . . . 109
4.7 Average gap difference for set covering instances (varying matrix density) . . . 109
4.8 Average gap difference for set packing instances (varying bandwidth) . . . 111
4.9 Average gap difference for set packing instances (varying matrix density) . . . 113

5.1 Reduced BDD for a binary optimization problem . . . 121
5.2 BDD representation using standard and 0-BDDs . . . 123
5.3 BDD representation using standard and 1-BDDs . . . 124
5.4 BDD representation using standard and 0/1-BDDs . . . 126
5.5 Exact BDD compilation . . . 130

5.6 Width-3 relaxed BDD 136
5.7 Width-3 restricted BDD for the knapsack problem 141
5.8 Width-3 relaxed BDD for the knapsack problem (with labels) 143
5.9 Relaxed BDD for the family of independent sets in a star graph 146
5.10 Number of instances solved versus time for BDD branch-and-bound and CPLEX 154
5.11 End gap comparison for BDD branch-and-bound and CPLEX 155

6.1 Cycle graph 175
6.2 Cycle graph with non-uniformly sized intersections 184
6.3 Comb graph 189
6.4 Web graph 197
6.5 Path graph 201
6.6 Intersecting system graph 206

List of Tables

2.1 Bound quality and computation times for LP and BDD relaxations, using **random** instances . . . 58

2.2 Bound quality and computation times for LP and BDD relaxations, using **dimacs** instances . . . 59

2.3 Bound comparison for the `dimacs` instances for an LP relaxation and Relaxed BDDs . . . 59

4.1 CPLEX parameters for comparison with restricted BDDs . . . 103

4.2 Average gap difference for set covering instances (varying bandwidth) . . . 108

4.3 Average gap difference for set covering instances (varying matrix density) . . 110

4.4 Average gap difference for set packing instances (varying bandwidth) . . . 111

4.5 Average gap difference for set packing instances (varying matrix density) . . 112

5.1 Comparison of BDD-Based branch-and-bound with IP solver CPLEX . . . 156

6.1 Cycle graph chromatic number lower bounds (0-1 model) . . . 212

6.2 Cycle graph chromatic number lower bounds (finite-domain model) . . . 214

6.3 Web graph chromatic number lower bounds . . . 215

6.4 Chromatic number lower bounds, finite-domain cuts versus 0-1 odd-hole cuts 218

Chapter 1

Introduction

The focus of the work presented in this dissertation is the investigation of new solution approaches to discrete optimization problems. This class of problems is applicable across a wide range of applications, including business analytics, process improvement, and health care operations, to name a few. With the exponential explosion of data and computational power, businesses, more than ever, are seeking to apply operations research techniques to discrete optimization problems that they encounter to better the way they operate. As the problems they seek to solve grow larger and more complex, it is crucial to improve on existing technologies on this class of optimization problems.

The first technique investigated here is the use of decision diagrams (DDs) to represent the feasible set of a problem. Binary Decision Diagrams (BDDs) [2, 50, 16], a specific type of decision diagram, provide compact graphical representations of Boolean functions, and have traditionally been used for circuit design and formal verification [44, 50]. More recently, however, BDDs and their generalization Multivalued Decision Diagrams (MDDs) [46] have been used in operations research for a variety of purposes, including cut generation [6], vertex enumeration [8], and post-optimality analysis [36, 37]. Decision diagrams have also been successfully applied to other problems, including for example sequential pattern mining and genetic programming [51, 68].

In the context of discrete optimization, DDs can be used to represent the set of feasible solutions to the problem. Unfortunately, the size of the DD can grow exponentially large.

In fact, as a corollary to a result in [65], there exists 0-1 problems for which the DD that exactly represents the set of feasible solutions must be exponentially large, regardless of the variable ordering chosen. As such, we investigate the use of *approximate* DDs to represent an *approximation* of the feasible set.

Relaxation MDDs were introduced in [3] as a replacement for the domain store relaxation, i.e., the Cartesian product of the variable domains, that is typically used in Constraint Programming (CP). MDDs provide a richer data structure that can capture a tighter relaxation of the feasible set of solutions, as compared with the domain store relaxation. In order to make this approach scalable, MDD relaxations of limited size are applied. Various methods for compiling these discrete relaxations are provided in [38]. The methods described in that paper focus on iterative splitting and edge filtering algorithms that are used to tighten the relaxations. Similar to classical domain propagation, MDD propagation algorithms have been developed for individual (global) constraints, including inequality constraints, equality constraints, *alldifferent* constraints and *among* constraints [38, 39] and has also been applied to disjunctive scheduling [19].

In this dissertation, however, we suggest using approximate DDs for the purpose of representing either an over-approximation or an under-approximation of the feasible set. The former (assuming a maximization problem) can then be used to prove upper-bounds on the optimal value, while the latter can be used to find good feasible solutions and/or prove lower-bounds.

In Chapter 2 we explore a technique that can be used to generate *relaxed* DDs that represents a relaxation of the feasible set. The technique described is a modification of a top-down algorithm that can be used to exactly represent the feasible set, which is also discussed here. We note that using DDs for problem relaxations is quite different than the traditional relaxations used in this domain. Typical methods of providing relaxations are continuous, like linear programming or semi-defining programming relaxations. Here we suggest using a *discrete* relaxation. As a test case, we apply the proposed method to the maximum independent set problem. We find that BDDs can deliver significantly tighter bounds than those obtained by state-of-the-art integer programming software, which solves an LP relaxation augmented by cutting planes. The BDD bounds are also obtained in far

less computation time.

The ordering of the variables plays an important role in not only the size of exact BDDs, but also in the bound obtained by relaxed BDDs. For example, there are 0-1 problems with n variables and a single equality constraint for which there exists orderings that result in a DD with width $2^{\frac{n}{2}}$, while other orderings result in DDs with width 2. It is well known that finding orderings which minimize the size of BDDs (or even improving on a given ordering) is NP-hard [22, 15]. Moreover, in preliminary computational results, we found that the ordering of the vertices is the single most important parameter in creating small width exact BDDs and in proving tight bounds via relaxed BDDs.

As such, we do a thorough investigation of the variable ordering for the application of DDs to the maximum independent set problem in this chapter. We explore particular classes of graphs (paths, cliques, interval graphs, trees), and find variable orderings that allow us to bound the largest possible size of the DD. In addition, we show that for every graph, there exist orderings for which the exact DD will have width bounded by the Fibonacci numbers.

We also explore heuristics orderings for relaxed DDs. We describe orderings specific to the maximum independent set problem, and also describe an ordering for general discrete optimization problems.

Bounds provided by relaxations are important, but methods used to tighten the relaxations are perhaps even more important. For example, cutting planes used to tighten linear programming relaxations are crucial to their application.

In Chapter 3 we discuss the problem of tightening relaxation bounds obtained by relaxed DDs. The techniques discussed there involve iteratively pulling out all solutions in the relaxed DD with objective function value equal to a given upper bound, and either finding a feasible solution (thereby proving the optimality of the bound) or proving that none exists (thereby allowing the upper-bound to be lowered). We compare the strength of bounds provided by relaxed DDs with those provided by a linear programming relaxation augmented with cutting planes on structured instances of the set covering problem. We also compare the speed with which relaxed DDs (used as a pure inference method) and integer programming solve the problem. We find that relaxed DDs are superior to conventional integer programming when the ones in the constraint matrix lie in a relatively narrow band.

That is, the matrix has relatively small bandwidth, meaning that the maximum distance between any two ones in the same row is limited.

The bandwidth of a set covering matrix can often be reduced, perhaps significantly, by reordering the columns. Thus relaxed DD can solve a given set covering problem much more rapidly than integer programming if its variables can be permuted to result in a relatively narrow bandwidth. Algorithms and heuristics for minimum bandwidth ordering are discussed in [54, 17, 21, 24, 34, 55, 60, 66].

In Chapter 4 we explore another form of approximate DDs: *restricted* DDs. As opposed to relaxed DDs, restricted DD represent an under-approximation of the feasible set. These structures can thus be used as a new method for generating heuristic solutions to binary optimization problems. We discuss how restricted DDs can be generated and show that the proposed algorithm delivers solutions of comparable quality to a state-of-the-art general-purpose optimization solver on randomly generated set covering and set packing problems.

Chapter 5 is the crux of the DD chapters. In this chapter we describe an algorithm that combines relaxed and restricted DDs into a single framework, developing a new branch-and-bound algorithm for discrete optimization problems. This algorithm differs substantially from typical branch-and-bound algorithms (perhaps the most widely used technique) for discrete optimization problem.

First, it employs relaxed DDs for upper-bounds (again, assuming a maximization problem). As discussed above, problem relaxations for discrete optimization problems are typically continuous relaxations. However, as the problems are themselves discrete, looking into discrete relaxations is a natural course of investigation.

Second, subproblems are generated by nodes in the relaxed DDs. In traditional branch-and-bound schemes, algorithms proceed by solving subproblems generated by fixing a given variable to 0 and 1. In the DD approach investigated here, the nodes produce subproblems that are defined by a pool of partial solutions. This allows for branching on multiple variable fixings taken at once and also eliminates certain symmetry from the search.

Again, as a text case, we discuss the application of the algorithm to the maximum independent set problem. Computational results show that more problems can be solved less time using the DD approach then conventionally integer programming techniques. In

addition, the algorithm is more robust in terms of the percent gap remaining after a given amount of computation time.

In Chapter 6, we change gears and look at obtaining strong cutting-planes from alternate models of 0-1 problems.

In integer programming models, a choice from several alternatives is typically encoded by a set of binary variables. For example, the job assigned to a particular worker might be represented by 0-1 variables y_{ij}, where $\sum_j y_{ij} = 1$ for each worker i, and $y_{ij} = 1$ indicates that job j is assigned to worker i. Valid inequalities can then be generated in terms of the 0-1 variables, so as to strengthen the continuous relaxation of the model.

An alternative approach is to formulate such a choice directly in terms of finite-domain variables. For example, variable x_i might indicate which job is assigned to worker i. The value of x_i need not be a number, but if we choose to denote jobs by numbers, we can analyze the convex hull of feasible solutions and write valid inequalities in terms of the variables x_i. These inequalities can then be mapped into a 0-1 model of the problem using a simple change of variable. The resulting 0-1 inequalities may be different from and more effective than known cutting planes for the 0-1 model.

This chapter explores the idea of using a finite-domain formulation of a problem as a source of new valid inequalities for the 0-1 model. We will refer to such inequalities as *finite-domain cuts*. We apply the idea to the vertex coloring problem on graphs, which has a natural finite-domain formulation in terms of *all-different* constraints. Such "global" constraints frequently appear in constraint programming models, where finite-domain variables are often used rather than 0-1 variables to encode discrete choices.

We employ a common strategy for generating problem-specific cuts: the identification of facet-defining cuts for special types of induced subgraphs, such as odd holes, webs, and paths. We identify cuts that bound the objective function (which we call *z-cuts*) as well as cuts that exclude infeasible solutions (*x-cuts*).

We find that for coloring problems, finite-domain cuts for several subgraph structures (when mapped into 0-1 space) provide tighter bounds than known 0-1 cuts for those subgraphs. Furthermore, we identify more general structures for which finite-domain cuts are substantially more effective than known cuts, or for which no known cuts exist.

When comparing finite-domain cuts with traditional 0-1 cuts, varying levels of strengthening can occur:

- Finite-domain *web cuts*, when mapped into the 0-1 model, yield tighter bounds than standard web cuts. This means, in particular, that if an existing algorithm identifies separating web cuts, we can replace them with more effective finite-domain web cuts at no additional computational cost.
- *Odd cycles* are a generalization of odd holes. We show that in the special case of odd holes, finite-domain cuts provide tighter bounds than standard odd hole and clique cuts. We can therefore replace known separating odd hole cuts with more effective cuts, at no additional cost. In the general case of odd cycles, only two finite-domain cuts for a given cycle provide a substantially tighter bound than hundreds or thousands of odd hole and clique cuts that can be generated for that cycle. We provide a polynomial-time algorithm that identifies all separating finite-domain cuts for a given odd cycle.
- By contrast, finite-domain *path cuts* do not improve existing bounds. When mapped into 0-1 space, they have no effect on the bound provided by the standard 0-1 model.
- *Intersecting systems* illustrate how a finite-domain perspective can yield facet-defining cuts for novel structures. To our knowledge, no 0-1 cuts have previously been identified for this general class of subgraphs. We also present a polynomial-time separation algorithm.

Mapping finite-domain cuts into 0-1 space has the advantage that finite-domain cuts can be combined with standard 0-1 constraints as well as previously known families of 0-1 cuts. However, bounds can also be obtained directly from the finite-domain model by solving its relaxation, which is much smaller than the 0-1 model. We investigate both approaches computationally.

In this chapter, we proceed with a problem statement and brief literature review. We then describe the mapping of finite-domain cuts into 0-1 space and prove some of its elementary properties. We next describe some general properties of the finite-domain polytope and then derive facet-defining inequalities for combs, odd cycles, webs, paths, and intersecting

systems, and study their properties when mapped into 0-1 space. In particular, we show that a family of facet-defining x-cuts gives rise to a family of facet-defining z-cuts in a canonical way, a result that is crucial for obtaining good bounds. A section on computational results compares the strength of finite-domain cuts and known 0-1 cuts on odd cycles and webs. It also demonstrates the advantages of odd cycle cuts on a set of benchmark instances. The chapter concludes with a summary and suggestions for future research.

Chapter 2

Relaxation Decision Diagrams

2.1 Introduction

Bounds on the optimal value are often indispensable for the practical solution of discrete optimization problems, as for example in branch-and-bound procedures. Such bounds are frequently obtained by solving a continuous relaxation of the problem, perhaps a linear programming (LP) relaxation of an integer programming model. In this chapter, we explore an alternative strategy of obtaining bounds from a *discrete* relaxation, namely a binary decision diagram (BDD).

Binary decision diagrams are compact graphical representations of Boolean functions [2, 50, 16]. They were originally introduced for applications in circuit design and formal verification [44, 50] but have since been used for a variety of other purposes. These include sequential pattern mining and genetic programming [51, 68].

A BDD can represent the feasible set of a 0-1 optimization problem, because the constraints can be viewed as defining a Boolean function $f(x)$ that is 1 when x is a feasible solution. Unfortunately, a BDD that exactly represents the feasible set can grow exponentially in size. We circumvent this difficulty by creating a *relaxed* BDD of limited size that represents a superset of the feasible set. The relaxation is created by merging nodes of the BDD in such a way that no feasible solutions are excluded. A bound on any additively separable objective function can now be obtained by solving a longest (or shortest) path

problem on the relaxed BDD. The idea is readily extended to general discrete (as opposed to 0-1) optimization problems by using *multi-valued decision diagrams* (MDDs).

As a test case, we apply the proposed method to the maximum independent set problem on a graph. We find that BDDs can deliver tighter bounds than those obtained by a strong LP formulation, even when the LP is augmented by cutting planes generated at the root node by a state-of-the-art mixed integer solver. In most instances, the BDD bounds are obtained in less computation time, even though we used a non-default barrier LP solver that is faster for these instances.

The chapter is organized as follows. After a brief literature review, we show how BDDs can represent 0-1 optimization problems in general and the maximum weighted independent set problem in particular. We then exhibit an efficient top-down compilation algorithm that generates exact reduced BDDs for the independent set problem, and prove its correctness. We then modify the algorithm to generate a limited-size relaxed BDD, prove its correctness, and show that it has polynomial time complexity. We also discuss variable ordering for exact and relaxed BDD compilation, as this can have a significant impact on the size of the exact BDD and the bound provided by relaxed BDDs. In addition, we describe heuristics for deciding which nodes to merge while building a relaxed BDD.

At this point we report computational results for random and benchmark instances of the maximum independent set problem. We experiment with various heuristics for ordering variables and merging nodes in the relaxed BDDs and test the quality of the bound provided by the relaxed BDDs versus the maximum BDD size. We then compare the bounds obtained from the BDDs with the LP bounds obtained by a commercial mixed integer solver. We conclude with suggestions for future work.

2.2 Previous Work

Relaxed BDDs and MDDs were introduced by [3] for the purpose of replacing the domain store used in constraint programming by a richer data structure. They found that MDDs drastically reduce the size of the search tree and allow faster solution of problems with multiple all-different constraints, which are equivalent to graph coloring problems. Similar

methods were applied to other types of constraints in [38] and [39]. The latter chapter also develops a general top-down compilation method based on state information at nodes of the MDD.

None of this work addresses the issue of obtaining bounds from relaxed BDDs. Three of us applied this idea to the set covering problem in a conference chapter [12], which reports good results for certain structured instances. In the current chapter, we present novel and improved methods for BDD compilation and relaxation. These methods are superior to continuous relaxation technology for a much wider range of instances, and require far less time.

The ordering of variables can have a significant bearing on the effectiveness of a BDD relaxation. We investigated this for the independent set problem in [9] and apply the results here.

We note that binary decision diagrams have also been applied to post-optimality analysis in discrete optimization [36, 37], cut generation in integer programming [6], and 0-1 vertex and facet enumeration [8].

Branch-and-bound methods for the independent set problem, which make essential use of relaxation bounds, are studied by [64, 67, 62], and surveyed by [63].

2.3 Binary Decision Diagrams

Given binary variables $x = (x_1, \ldots, x_n)$, a *binary decision diagram* (BDD) $B = (U, A)$ for x is a directed acyclic multigraph that encodes a set of values of x. The set U of nodes is partitioned into layers $L_1, \ldots, L_n$ corresponding to variables $x_1, \ldots, x_n$, plus a terminal layer L_{n+1}. Layers L_1 and L_{n+1} are singletons consisting of the *root* node r and the *terminal* node t, respectively. All directed arcs in A run from a node in some layer L_j to a node in some deeper layer L_k $(j < k)$. For a node $u \in L_j$, we write $\ell(u) = j$ to indicate the layer in which u lies.

Each node $u \in L_j$ has one or two out-directed arcs, a *0-arc* $a_0(u)$ and/or a *1-arc* $a_1(u)$. These correspond to setting x_j to 0 and 1, respectively. We use the notation $b_0(u)$ to indicate the node at the opposite end of arc $a_0(u)$, and similarly for $b_1(u)$. Thus, 0-arc $a_0(u)$ is

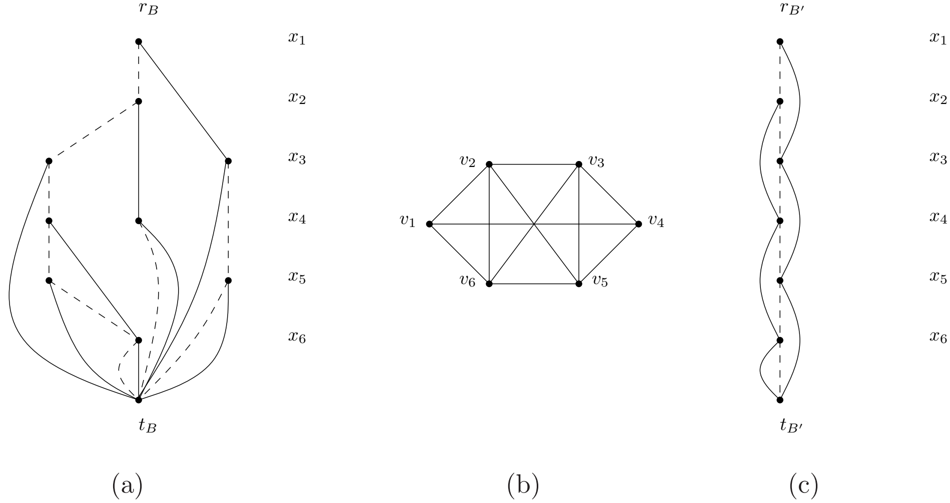


Figure 2.1: (a) Example of a BDD. (b) Instance of the independent set problem for which (a) is an exact BDD. (c) Relaxed BDD for the instance in (b).

$(u, b_0(u))$, and 1-arc $a_1(u)$ is $(u, b_1(u))$. Each arc-specified path from r to t represents the 0-1 tuple x in which $x_{\ell(u)} = 1$ for each 1-arc $a_1(u)$ on the path, and $x_j = 0$ for all other j. The entire BDD represents the set $\text{Sol}(B)$ of all tuples corresponding to r–t paths.

It is often useful to abbreviate a BDD by using *long arcs*. These arcs skip over variables whose values are represented implicitly. A long arc can indicate that all skipped variables take the value zero (resulting in a *zero-suppressed* BDD) or the value one (a *one-suppressed* BDD). More commonly, a long arc indicates that the skipped variables can take either value. One advantage of BDDs is that we can choose the type of long arc that suits the problem at hand. We use zero-suppressed BDDs [58] because there are many zero-valued arcs in BDDs for the independent set problem. Thus a long arc from layer L_j to layer L_k encodes the partial assignment $(x_j, \ldots, x_{k-1}) = (1, 0, \ldots, 0)$.

Figure 2.1(a) illustrates a BDD for variables $x = (x_1, \ldots, x_6)$. The left-most path from root node r to terminal node t represents the tuple $(x_1, \ldots, x_6) = (0, 0, 1, 0, 0, 0)$. The third arc in the path is a long arc because it skips three variables. It encodes the partial assignment $(x_3, x_4, x_5, x_6) = (1, 0, 0, 0)$. The entire BDD of Fig. 2.1(a) represents a set of 10 tuples, corresponding to the 10 r–t paths.

Given nodes $u, u' \in U$, we will say that $B_{uu'}$ is the portion of B induced by the nodes in

U that lie on some directed path from u to u'. Thus $B_{rt} = B$. Two nodes u, u' on a given layer of a BDD are *equivalent* if B_{ut} and $B_{u't}$ are the same BDD. A *reduced* BDD is one that contains no equivalent nodes. A standard result of BDD theory [16, 68] is that for a fixed variable order, there is a unique reduced BDD that represents a given set. The *width* ω_j of layer L_j is $|L_j|$, and the width $\omega(B)$ of a BDD B is $\max_j\{\omega_j\}$. The BDD of Fig. 2.1(a) is reduced and has width 2.

The feasible set of any optimization problem with binary variables $x_1, \ldots, x_n$ can be represented by an appropriate reduced BDD. The BDD can be regarded as a compact representation of a search tree for the problem. It can in principle be obtained by omitting infeasible leaf nodes from the search tree, superimposing isomorphic subtrees, and identifying all feasible leaf nodes with t. We will present below a much more efficient procedure for obtaining a reduced BDD. A slight generalization of BDDs, *multi-valued decision diagrams* (MDDs), can similarly represent the feasible set of any discrete optimization problem. MDDs allow a node to have more than two outgoing arcs and therefore accommodate discrete variables with several possible values.

2.4 BDD Representation of Independent Sets

We focus on BDD representations of the *maximum weighted independent set problem.* Given a graph $G = (V, E)$, an *independent set* is a subset of the vertex set V, such that no two vertices are connected by an edge in E. If each vertex v_j is associated with a weight w_j, the problem is to find an independent set of maximum weight. If each $w_j = 1$, we have the *maximum independent set problem.*

If we let binary variable x_j be 1 when v_j is included in the independent set, the feasible solutions of any instance of the independent set problem can be represented by a BDD on variables $x_1, \ldots, x_n$. Figure 2.1(a), for example, represents the 10 independent sets of the graph in Fig. 2.1(b).

We can remove any node u in a BDD with a single outgoing arc if it is a 0-arc $a_0(u)$. This is accomplished by replacing every 0-arc $a_0(u')$ for which $b_0(u') = u$ with a longer arc $a_0(u')$ for which $b_0(u') = b_0(u)$. We can similarly replace every such 1-arc. If the BDD represents

an instance of the independent set problem, a single outgoing arc must be a 0-arc, which means that all nodes with single outgoing arcs can be removed. Every node in the resulting BDD has exactly two outgoing arcs.

To represent the objective function in the BDD, let each 1-arc $a_1(u)$ have length equal to the weight $w_{\ell(u)}$, and each 0-arc length 0. Then the length of a path from r to t is the weight of the independent set it represents. The weighted independent set problem becomes the problem of finding a longest path in a BDD. If for all vertices v_j weight $w_j = 1$, the four longest paths in the BDD of Fig. 2.1(a) have length 2, corresponding to the maximum independent sets $\{v_1, v_3\}$, $\{v_1, v_5\}$, $\{v_2, v_4\}$, and $\{v_4, v_6\}$.

Any binary optimization problem with an additively separable objective function $\sum_j f_j(x_j)$ can be similarly represented as a longest path problem on a BDD. Zero-suppressing long edges may be used if $f_j(0) = 0$ and $f_j(1) \geq 0$ for each j. This condition is met by any independent set problem with nonnegative weights. It can be met by any binary problem if each $f_j(x_j)$ is replaced with $\bar{f}_j(\bar{x}_j)$, where $\bar{f}_j(0) = 0$ and

$$\begin{aligned} \bar{f}_j(1) &= f_j(1) - f_j(0) \text{ and } \bar{x}_j = x_j, && \text{if } f_j(1) \geq f_j(0) \\ \bar{f}_j(1) &= f_j(0) - f_j(1) \text{ and } \bar{x}_j = 1 - x_j, && \text{otherwise.} \end{aligned}$$

In addition, recent work by [43] shows how non-separable objective functions may be represented by BDDs.

2.5 Exact and Relaxed BDDs

If Sol(B) is equal to the feasible set of an optimization problem, we will say that B is an *exact BDD* for the problem. If Sol(B) is a superset of the feasible set, B is a *relaxed BDD* for the problem. We will construct *limited-width* relaxed BDDs by requiring $\omega(B)$ to be at most some pre-set maximum width W.

Figure 2.1(c) shows a relaxed BDD B' of width 1 for the independent set problem instance of Fig. 2.1(b). B' represents 21 vertex sets, including the 10 independent sets. The length of a longest path in B' is therefore an upper bound on the optimal value of the original problem instance. If, again, for all vertices v_j weight $w_j = 1$, the longest path length is 3, which provides an upper bound on the maximum cardinality 2 of an independent set.

2.6 Exact BDD Compilation

We now describe an algorithm that builds an exact reduced BDD for the independent set problem. Similar algorithms can be designed for any optimization problem on binary variables by associating a suitable state with each node [39]. Choosing the state variable can be viewed as analogous to formulating a model for the LP relaxation, because it allows the BDD to reflect the problem at hand.

Starting with the root r, the procedure constructs the BDD $B = (U, A)$ layer by layer, selecting a graph vertex for each layer and associating a state with each node. We define the state as follows. Using a slight abuse of notation, let $\text{Sol}(B)$ be the set of independent sets represented by B (rather than the corresponding set of tuples x). Thus, in particular, $\text{Sol}(B_{ru})$ is the set of independent sets defined by paths from r to u. Let the *neighborhood* $N(T)$ of a vertex set T be the set of vertices adjacent to vertices in T, where by convention $T \subseteq N(T)$. The *state* $s(u)$ of node u is the set of vertices that can be added to any of the independent sets defined by paths from r to u. Thus

$$s(u) = \{v_{\ell(u)}, \ldots, v_n\} \setminus \bigcup_{T \in \text{Sol}(B_{ru})} N(T).$$

In an exact BDD, all paths to a given node u define partial assignments to x that have the same feasible completions. So $s(u) = \{v_{\ell(u)}, \ldots, v_n\} \setminus N(T)$ for any $T \in \text{Sol}(B_{ru})$. In addition, no two nodes on the same layer of an exact reduced BDD have the same feasible completions. So we have the following:

Lemma 1 *An exact BDD for G is reduced if and only if $s(u) \neq s(u')$ for any two nodes u, u' on the same layer of the BDD.*

The exact BDD compilation is stated in Algorithm 1. We begin by creating the root r of B, which has state $s(r) = V$ because every vertex in V is part of some independent set. We then add r to a pool P of nodes that have not yet been placed on some layer. Each node $u \in P$ is stored along with its state $s(u)$ and the arcs that terminate at u.

To create layer L_j, we first select the j-th vertex v_j by means of a function `select` (step 4), which can follow a predefined order or select vertices dynamically. We let L_j contain the nodes $u \in P$ for which $v_j \in s(u)$. These are the only nodes in P that will have

both outgoing arcs $a_0(u)$ and $a_1(u)$. All of the remaining nodes in P would have only an outgoing 0-arc if placed on this layer and can therefore be skipped. The nodes in L_j are removed from P, as we need only process them once.

For each node u in L_j, we create outgoing arcs $a_0(u)$ and $a_1(u)$ as follows. Node $b_0(u)$ (i.e., the node at the opposite end of $a_0(u)$) has state $s_0 = s(u)\backslash\{v_j\}$, and node $b_1(u)$ has state $s_1 = s(u)\backslash N(\{v_j\})$. To ensure that the BDD is reduced, we check whether $s_0 = s(u')$ for some node $u' \in P$, and if so let $b_0(u) = u'$. Otherwise, we create node u_0 with $s(u_0) = s_0$, let $b_0(u) = u_0$, and insert u_0 into P. If $s_0 = \emptyset$, u_0 is the terminal node t. Arc $a_1(u)$ is treated similarly. After the last iteration, P will contain exactly one node with state $\emptyset$, and it becomes the terminal node t of B.

We now show this algorithm returns the exact BDD.

Theorem 1 *For any graph $G = (V, E)$, Algorithm 1 generates a reduced exact BDD for the independent set problem on G.*

Proof. Let $\text{Ind}(G)$ be the collection of independent sets of G. We wish to show that if B is the BDD created by Algorithm 1, $\text{Sol}(B) = \text{Ind}(G)$. We proceed by induction on $n = |V|$.

First, suppose $n = 1$, and let G consist of a single vertex v. B consists of two nodes, r and t, and two arcs $a_0(r)$ and $a_1(r)$, both directed from r to t. Therefore, $\text{Sol}(B) = \{\emptyset, v\} = \text{Ind}(G)$. Moreover, this BDD is trivially reduced.

For the induction hypothesis, suppose that Algorithm 1 creates a reduced exact BDD for any graph on fewer than n (≥ 2) vertices. Let G be a graph on n vertices. Suppose the `select` function in Step 4 returns vertices in the order $v_1, \ldots, v_n$. Let $G_0 = (V_0, E_0)$ be the subgraph of G induced by vertex set $V\backslash\{v_1\}$, and $G_1 = (V_1, E_1)$ the subgraph induced by $V\backslash N(v_1)$. Then $\text{Ind}(G) = \text{Ind}(G_0) \cup \{T \cup \{v_1\} \mid T \in \text{Ind}(G_1)\}$, since each independent set either excludes v_1 (whereupon it appears in $\text{Ind}(G_0)$) or includes v_1 (whereupon it appears as the union of $\{v_1\}$ with a set in $\text{Ind}(G_1)$).

Let B be the BDD returned by the algorithm for G. By construction, $s(b_0(r)) = V_0$ and $s(b_1(r)) = V_1$. Let B_0 be the BDD that the algorithm creates for G_0, and similarly for B_1. We observe as follows that $B_0 = B_{b_0(r)t}$ and $B_1 = B_{b_1(r)t}$. The root r_0 of B_0 has $s(r_0) = V_0$, the same state as node $b_0(r)$ in B. But the successor nodes created by the algorithm for r_0

Algorithm 1 Exact BDD Compilation

```
1: Create node r with s(r) = V
2: Let P = {r} and R = V
3: for j = 1 to n do
4:   v_j = select(R, P)
5:   R ← R\{v_j}
6:   L_j = {u ∈ P : v_j ∈ s(u)}
7:   P ← P\L_j
8:   for all u ∈ L_j do
9:     s_0 := s(u)\{v_j}, s_1 := s(u)\N(v_j)
10:    if ∃ u' ∈ P with s(u') = s_0 then
11:      a_0(u) = (u, u')
12:    else
13:      create node u_0 with s(u_0) = s_0 (u_0 = t if s_0 = ∅)
14:      a_0(u) = (u, u_0)
15:      P ← P ∪ {u_0}
16:    if ∃ u' ∈ P with s(u') = s_1 then
17:      a_1(u) = (u, u')
18:    else
19:      create node u_1 with s(u_1) = s_1 (u_1 = t if s_0 = ∅)
20:      a_1(u) = (u, u_1)
21:      P ← P ∪ {u_1}
22: Let t be the remaining node in P and set L_{n+1} = {t}
```

and $b_0(r)$ depend entirely on the state and are therefore identical in B_0 and B, respectively. Moreover, the states of the successor nodes depend entirely on the state of the parent and which branch is taken. Thus the successor nodes have the same states in B_0 as in B. If we apply this reasoning recursively, we obtain $B_0 = B_{b_0(r)t}$. A parallel argument shows that

$B_1 = B_{b_1(r)t}$. Now

$$\begin{aligned}
\text{Sol}(B) &= \text{Sol}(B_{b_0(r)t}) \cup \{T \cup \{v_1\} \mid T \in \text{Sol}(B_{b_1(r)t})\} \\
&= \text{Sol}(B_0) \cup \{T \cup \{v_1\} \mid T \in \text{Sol}(B_1)\} \\
&= \text{Ind}(G_0) \cup \{T \cup \{v_1\} \mid T \in \text{Ind}(G_1)\} \\
&= \text{Ind}(G)
\end{aligned}$$

as claimed, where the third equation is due to the inductive hypothesis. Furthermore, since all nodes with the same state are merged, Lemma 1 implies that B is reduced. □

To analyze the time complexity of Algorithm 1, we assume that the `select` function (Step 4) is "polynomial" in the sense that its running time is at worst proportional to $|V|$ or the number of BDD nodes created so far, whichever is greater.

Lemma 2 *If the `select` function is polynomial, then the time complexity of Algorithm 1 is polynomial in the size of the reduced exact BDD $B = (U, A)$ constructed by the algorithm.*

Proof. We observe that an arc of B is never rechecked again once it was created in one of the Steps 11, 14, 17, or 20. Hence, the complexity of the algorithm is dominated by the `select` function or the constructive operations required when creating the out-arcs of a node removed from the pool P. The `select` function is clearly polynomial in $|U|$ and $|V|$. The constructive operations consist of creating a new state (Step 9) and inserting or searching in the node pool (Steps 10, 15, 16, and 21), which can be implemented in $O(|V|)$. Since every node has exactly two outgoing arcs (i.e., $|A| = 2|U|$), the resulting worst-case complexity is $O(|U|\,|V|)$, and the lemma follows. □

2.7 Relaxed BDDs

Limited-width relaxed BDDs allow us to represent an over-approximation of the family of independent sets of a graph, and thus obtain an upper bound on the optimal value of the independent set problem.

We propose a novel top-down compilation method for constructing relaxed BDDs. The procedure modifies Algorithm 1 by forcing nodes to be merged when a particular layer

Algorithm 2 Node merger for obtaining a relaxed BDD.

Insert immediately after line 7 of Algorithm 1.

1: **while** $\omega_j > W$ **do**
2: $M :=$ node_select(L_j) // *where* $2 \leq |M| \leq \omega_j - W$
3: $s_{\text{new}} := \bigcup_{u \in M} s(u)$
4: $L_j \leftarrow L_j \backslash M$
5: **if** $\exists u' \in L_j$ with $s(u') = s_{\text{new}}$ **then**
6: merge(M, u')
7: **else**
8: Create node $\hat{u}$ with $s(\hat{u}) = s_{\text{new}}$
9: merge$(M, \hat{u})$
10: $L_j = L_j \cup \{\hat{u}\}$

exceeds a pre-set maximum width W. This modification is given in Algorithm 2, which is to be inserted immediately after line 7 in Algorithm 1.

The procedure is as follows. We begin by checking if $\omega_j > W$, which indicates that the width of layer L_j exceeds W. If so, we select a subset M of L_j using function `node_select` in Step 2, which ensures that $2 \leq |M| \leq \omega_j - W$. The set M represents the nodes to be merged so that the desired width is met. Various heuristics for selecting M are discussed in Section 2.8.

The state of the new node that results from the merge, s_{new}, must be such that no feasible independent set is lost in further iterations of the algorithm. As will be established by Theorem 2, it suffices to let s_{new} be the union of the states associated with the nodes in M (Step 3). Once s_{new} is created, we search for some node $u' \in L_j$ such that $s(u') = s_{new}$. If u' exists, then by Lemma 1 we are only required to direct the incoming arcs of the nodes in M to u', as presented in Algorithm 3. Otherwise, we create a new node $\hat{u}$ with $s(\hat{u}) = s_{new}$ and add it to L_j.

In each iteration of the *while* loop in Algorithm 2, we decrease the size of L_j by at least $|M| - 1$. Thus, after at most $\omega_j - W$ iterations, the layer L_j will have width no greater than W. The modified Algorithm 1 hence yields a limited-width W BDD, i.e. $\omega(B) \leq W$.

Algorithm 3 merge(M, u')

1: **for all** $u \in M$ **do**
2: **for all** arcs $a_0(w)$ with $b_0(w) = u$ **do**
3: $b_0(w) \leftarrow u'$
4: **for all** arcs $a_1(w)$ with $b_1(w) = u$ **do**
5: $b_1(w) \leftarrow u'$

The correctness of Algorithm 2 is proved by showing that every r-t path of the exact BDD remains after merging operations.

Theorem 2 *For any graph $G = (V, E)$, Algorithm 1 modified by adding Algorithm 2 after line 7 generates a relaxed BDD.*

Proof. We will use the notation B_u for the BDD consisting of all r–t paths in B that pass through u. Thus

$$\text{Sol}(B_u) = \{V_1 \cup V_2 \mid V_1 \in \text{Sol}(B_{ru}),\ V_2 \in \text{Sol}(B_{ut})\} \tag{2.1}$$

It suffices to show that each iteration of the while-loop yields a relaxed BDD if it begins with a relaxed BDD. Thus we show that if B is a relaxed (or exact) BDD, then the BDD $\hat{B}$ that results from merging the nodes in M satisfies $\text{Sol}(B) \subseteq \text{Sol}(\hat{B})$. Here M is any proper subset of L_j for an arbitrary $j \in \{2, \ldots, n-1\}$.

Let $M = \{u_1, \ldots, u_k\}$ be the nodes to be merged into $\hat{u}$. Also, let $\bar{B}$ be the BDD consisting of all r–t paths in B that do *not* include any of the nodes u_i. Then

$$\text{Sol}(B) = \text{Sol}(\bar{B}) \cup \bigcup_{i=1}^{k} \text{Sol}(B_{u_i})$$

The merge procedure has no effect on $\text{Sol}(\bar{B})$. Hence it remains to show that

$$\bigcup_{i=1}^{k} \text{Sol}(B_{u_i}) \subseteq \text{Sol}(\hat{B}_{\hat{u}})$$

But we can write

$$\begin{aligned}
\bigcup_{i=1}^{k} \mathrm{Sol}(B_{u_i}) &= \bigcup_{i=1}^{k} \{V_1 \cup V_2 \mid V_1 \in \mathrm{Sol}(B_{ru_i}),\ V_2 \in \mathrm{Sol}(B_{u_i t})\} \\
&= \left\{ V_1 \cup V_2 \;\middle|\; V_1 \in \bigcup_{i=1}^{k} \mathrm{Sol}(B_{ru_i}),\ V_2 \in \bigcup_{i=1}^{k} \mathrm{Sol}(B_{u_i t}) \right\} \\
&= \left\{ V_1 \cup V_2 \;\middle|\; V_1 \in \mathrm{Sol}(\hat{B}_{r\hat{u}}),\ V_2 \in \bigcup_{i=1}^{k} \mathrm{Sol}(B_{u_i t}) \right\} \\
&\subseteq \left\{ V_1 \cup V_2 \;\middle|\; V_1 \in \mathrm{Sol}(\hat{B}_{r\hat{u}}),\ V_2 \in \mathrm{Sol}(\hat{B}_{\hat{u}t}) \right\} \\
&= \mathrm{Sol}(\hat{B}_{\hat{u}})
\end{aligned}$$

The first and last equations are due to (2.1). The third equation is due to $\bigcup_i \mathrm{Sol}(B_{ru_i}) = \mathrm{Sol}(\hat{B}_{r\hat{u}})$, which follows from the fact that $\hat{u}$ receives precisely the paths received by the u_is before the merge. The fourth line is due to $\bigcup_i \mathrm{Sol}(B_{u_i t}) \subseteq \mathrm{Sol}(\hat{B}_{\hat{u}t})$. This follows from the facts that (a) $\mathrm{Sol}(B_{u_i t})$ contains the independent sets in the subgraph of G induced by $s(u_i)$; (b) $\mathrm{Sol}(\hat{B}_{\hat{u}t})$ contains the independent sets in the subgraph induced by $s(\hat{u})$; and (c) $s(u_i) \subseteq s(\hat{u})$ for all i. □

The time complexity of Algorithm 2 is highly dependent on the `node_select` function and on the number of nodes to be merged. Once a subset M of nodes has been chosen, taking the union of the states (Step 3) has a time complexity of $O(|M||V|)$, and Algorithm 3 has a worst-case time complexity of $O(W|M|)$ by supposing that every node in M is adjacent to as many as W nodes located in previous layers. Hence, if k is the number of nodes to be merged, the complexity of Algorithm 2 is $O(H(k) + |M||V| + W|M|)$ per iteration of the *while* loop in Step 1, where $H(k)$ is the complexity of the node selection heuristic (`node_select`) for a given k. The number of iterations depends on the size of the selected node set. For example, if $|M|$ is always 2, then at most $W - k$ iterations are required (if none of the newly defined states appeared in L_j previously). The time complexity for the complete relaxation procedure is given by the following lemma.

Lemma 3 *Let S be the time complexity of selecting the next variable (`select` function in Step 4 of Algorithm 1), and let $R(k)$ be the time complexity of Algorithm 2. The worst-case time complexity of Algorithm 1 modified with the procedure in Algorithm 2 is given by*

$O(n(S + R(nW) + W|V|))$.

Proof. If k nodes are removed from the pool in Step 6 of Algorithm 1, then the merging procedure in Algorithm 2 ensures that at most $2\min\{k, W\}$ new nodes are added back to the pool. Thus, at each iteration the pool can be increased by at most W nodes. Since n iterations in the worst case are required for the complete compilation, the pool can have at most nW nodes.

Suppose now nW nodes are removed from the pool (Step 6 of Algorithm 1) at a particular iteration. These nodes are first merged so that the maximum width W is met (Algorithm 2), and then new nodes or arcs are created according to the result of the merge. The time complexity for the first operation is $R(nW)$, which yields a new layer with at most W nodes. For the second operation, we observe as in Lemma 2 that creating a new state or searching in the pool size can be implemented in time $O(|V|)$; hence, the second operation has a worst-case time complexity of $O(W|V|)$.

This implies that the time required per iteration is $O(S + R(nW) + W|V|)$, yielding a time complexity of $O(n(S + R(nW) + W|V|))$ for the modified procedure. □

2.8 Merging Heuristics

The selection of nodes to merge in a layer that exceeds the maximum allotted width W is critical for the construction of relaxation BDDs. Different selections may yield dramatic differences on the obtained upper bounds on the optimal value, since the merging procedure adds paths corresponding to infeasible solutions to the BDD.

In this section we present a number of possible heuristics for selecting nodes. This refers to how the subsets M are chosen on line 2 in Algorithm 2. The heuristics we test are described below.

random: Randomly select a subset M of size $|L_j| - W + 1$ from L_j. This may be used a stand-alone heuristic or combined with any of the following heuristics for the purpose of generating several relaxations.

minLP: Sort nodes in L_j in increasing order of the longest path value up to those nodes and merge the first $|L_j| - W + 1$ nodes. This is based on the idea that infeasibility is introduced

into the BDD only when nodes are merged. By selecting nodes with the smallest longest path, we lose information in parts of the BDD that are unlikely to participate in the optimal solution.

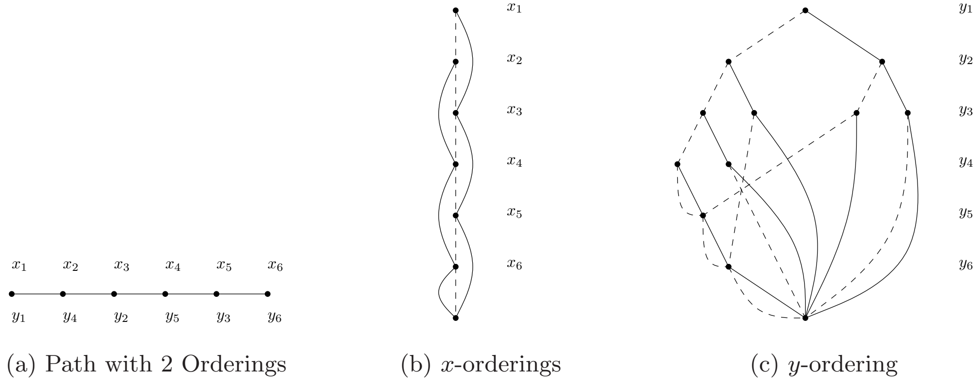
minSize: Sort nodes in L_j in decreasing order of their corresponding state sizes and merge the first 2 nodes until $|L_j| \leq W$. This heuristic merges nodes that have the largest number of vertices in their associated states. Because larger vertex sets are likely to have more vertices in common, the heuristic tends to merge nodes that represent similar regions of the solution space.

2.9 Variable Ordering

The ordering of the vertices plays an important role in not only the size of exact BDDs, but also in the bound obtained by relaxed BDDs. It is well known that finding orderings that minimize the size of BDDs (or even improving on a given ordering) is NP-hard [22, 15]. We found that the ordering of the vertices is the single most important parameter in creating small width exact BDDs and in proving tight bounds via relaxed BDDs.

Different orderings can yield exact BDDs with dramatically different widths. For example, Figure 2.2a shows a path on 6 vertices with two different orderings given by $x_1, \ldots, x_6$ and $y_1, \ldots, y_6$. In Figure 2.2b we see that the vertex ordering $x_1, \ldots, x_6$ yields an exact BDD with width 1, while in Figure 2.2c the vertex ordering $y_1, \ldots, y_6$ yields an exact BDD with width 4. This last example can be extended to a path with $2n$ vertices, yielding a BDD with a width of 2^{n-1}, while ordering the vertices according to the order that they lie on the paths yields a BDD of width 1.

In the remainder of this section we describe classes of graphs for which an appropriate ordering of the vertices leads to a bound on the width of the exact BDD. In addition, we provide a set of orderings based on maximal path decompositions that yield exact reduced BDDs in which the width of layer L_j is bounded by the $(j+1)$-st Fibonacci number for any graph. Based on this analysis, we describe various heuristic orderings for reduced BDDs, on the assumption that an ordering that results in a small-width exact reduced BDD also results in a relaxed BDD that yields a strong bound on the objective function.



(a) Path with 2 Orderings (b) x-orderings (c) y-ordering

Figure 2.2: Comparing Exact BDD for a Path Graph with 2 Different Orderings

2.9.1 Exact BDD Orderings

Here we present orderings of vertices for interval graphs, trees, and general graphs for which we can bound the width, and therefore the size, of the exact reduced BDD.

We first consider interval graphs; that is, graphs that are isomorphic to the intersection graph of a multiset of intervals on the real line. Such graphs have vertex orderings $v_1, \ldots v_n$ for which each vertex v_i is adjacent to the set of vertices $v_{a_i}, v_{a_i+1}, \ldots v_{i-1}, v_{i+1}, \ldots, v_{b_i}$ for some a_i, b_i. We call such an ordering an *interval ordering* for G. Note that paths and cliques, for example, are contained in this class of graphs.

Theorem 3 *For any interval graph, an interval ordering $v_1, \ldots, v_n$ yields an exact reduced BDD with width 1.*

Proof. Let $T_k = \{v_k, \ldots, v_n\}$. We first show by induction that for any interval ordering $v_1, \ldots, v_n$, there is a k such that $s(u) = V_k$ for all $u \in P$ throughout the entire execution of Algorithm 1.

At the start of the algorithm, $P = \emptyset, L_1 = \{r\}$ with $s(r) = T_1$. Starting from r, we have $s_0 = s(r) \setminus \{v_1\} = T_2$, and $s_1 = s(r) \setminus N(v_1) = T_{b_1+1}$. Therefore, at the end of iteration $j = 1$, P contains two nodes with states T_2 and T_{b_1+1}.

Now fix an arbitrary $j < n$ and assume for the induction hypothesis that, at the start of iteration j, each node $u \in P$ has $s(u) = T_k$. Note that $k \geq j$, since at each iteration we always eliminate v_j for each state (if it appears). Then $|L_j| = 1$, because there can be at

most one node $u' \in P$ with $v_j \in s(u')$, and the only T_k with $k \geq j$ that contains v_j is T_j. Starting from u', we have $s_0 = V_j \setminus \{v_j\} = T_{j+1}$ and $s_1 = V_j \setminus N(v_j) = T_{b_j+1}$. We therefore add at most two nodes to P at the end of iteration j, each with a state of form T_k for some k. This proves the claim.

We conclude that $s(u) = T_k$ for all $u \in P$. For any j, we have $|L_j| = 1$, because there is at most one node with $v_j \in s(u)$. Therefore, $\omega(B) = 1$. □

We now prove a width bound for trees.

Theorem 4 *For any tree with $n \geq 2$ vertices, there exists an ordering of the vertices that yields an exact reduced BDD with width at most $n/2$.*

The proof will use a lemma demonstrated by [45].

Lemma 4 *In any tree there exists a vertex v for which the connected components created upon deleting v from the tree contain at most $n/2$ vertices.*

Proof of Theorem 4. We proceed by induction on n. For $n = 2$, any tree is an interval graph, and by Theorem 3 there exists an ordering of the vertices that yields an exact reduced BDD with width 1.

For the inductive hypothesis, we suppose that for any tree T with $n' < n$ vertices, there exists an ordering of the vertices in T for which the exact reduced BDD has width at most $n'/2$. Let T be any tree with n vertices. Let v be a cut vertex satisfying the conditions of Lemma 4. Each connected component $T_i = (V_i, E_i)$, $i = 1, \ldots, k$, created upon deleting v from T is a tree. By induction, for each i, there is an ordering of the vertices in V_i for which the exact reduced BDD for T_i has width at most $\frac{|V_i|}{2} \leq \frac{n}{4}$. Let $v_1^i, \ldots, v_{|V_i|}^i$ be such an ordering and $B_i = (U_i, A_i)$ be the exact reduced BDD for T_i with this ordering.

Consider the ordering $v_1^1, \ldots, v_{|V_1|}^1, v_1^2, \ldots, v_{|V_k|}^k, v$ of the vertices in T; i.e., we order the vertices by the component orderings that yield an exact reduced BDD with width at most $\frac{n}{4}$, followed by the cut vertex v. We now show that using this ordering, the exact reduced BDD $B = (U, A)$ for T has width at most $2 \cdot \frac{n}{4} = \frac{n}{2}$, finishing the proof.

Fix $j, 1 \leq j \leq n-1$, and let v_ℓ^i be the jth vertex in the ordering for T. We claim that using this ordering, for each vertex $u \in L_j$ of B, there exists a $w \in L_\ell^i$, the ℓth layer of B_i,

for which

$$s(u) = \begin{cases} s(w) \bigcup (V_{i+1} \cup \cdots \cup V_k) \\ s(w) \bigcup (V_{i+1} \cup \cdots \cup V_k) \setminus \{v\}. \end{cases}$$

Consider the BDD B_{ru}. For every set $W \in \text{Sol}(B_{ru})$ we have that $v^i_\ell \in s(u)$ and

$$\begin{aligned} s(u) &= (v_j \cup v_{j+1} \cup \cdots \cup v_n) \setminus N(W) \\ &= \left(\left\{ v^i_\ell \cup v^i_{\ell+1} \cup \cdots \cup v^i_{|V_i|} \right\} \cup \left\{ V^{i+1} \cup \cdots \cup V^k \right\} \right) \setminus N(W) \\ &= \left(\left\{ v^i_\ell \cup v^i_{\ell+1} \cup \cdots \cup v^i_{|V_i|} \right\} \setminus N(W) \right) \cup \left\{ V^{i+1} \cup \cdots \cup V^k \right\}, \end{aligned}$$

where the last equality follows because the nodes in W are not adjacent to any vertex in the remaining components.

Now, consider the set $W[V^i] = W \cap V^i$. This must be an independent set in the graph $T[V^i]$, the graph induced by vertex set V^i. In addition, $v^i_\ell \in V^i \setminus N(W[V^i])$ is in $T[V^i]$ because this vertex also appears in $s(u)$. Therefore, there is a node $w \in L^i_\ell$ with state $s(w) = \left\{ v^i_\ell \cup v^i_{\ell+1} \cup \cdots \cup v^i_{|V_i|} \right\} \setminus N(W[V^i])$. In the entire graph T, $N(W)$ contains exactly the vertices in $T[V^i]$ that are in $N(W[V^i])$ and possibly vertex v, because v is the only vertex, besides those vertices in V^i, that can be adjacent to any vertex in V^i, as desired. □

Finally, we prove a bound for general graphs.

Theorem 5 *Given any graph, there exists an ordering of the vertices that yields an exact reduced BDD $B = (U, A)$ with width equal to at most the jth Fibonacci number Fib_{j+1}.*

Proof. Given graph $G = (V, E)$, we define a *maximal path decomposion ordering* of V as an ordered partition of the vertex set $V = V^1 \cup \cdots \cup V^k$ together with an ordering $v^i_1, \ldots, v^i_{|V^i|}$ of the vertices in each partition V^i for which

$$\begin{aligned} &\left(v^i_j, v^i_{j+1}\right) \in E && \text{for } i = 1, \ldots, k, j = 1, \ldots, |V^k| - 1 \\ &N\left(v^i_{|V^i|}\right) \subseteq V^1 \cup \cdots \cup V^i && \text{for } i = 1, \ldots, k. \end{aligned}$$

Thus each partition is covered by a path whose last vertex is independent of all vertices in the remaining partitions.

We show that for any maximal path decomposition ordering, the exact reduced BDD B will have $\omega_j \leq \text{Fib}_{j+1}$. Let P_j be the pool of nodes in Algorithm 1 before line 4 in iteration

j. We will show that $|P_j| \leq \text{Fib}_{j+1}$, which implies that $\omega_j = |L_j| \leq |P_j| \leq \text{Fib}_{j+1}$, as desired.

We first consider the case with $k = 1$; i.e., the graph contains a Hamiltonian path. Let $v_1, \ldots, v_n$ be a Hamiltonian path in G and the maximal path decomposition ordering we use to create an exact reduced BDD for G. We proceed by induction.

We first note that $P_1 = \{r\}$, so that $|P_1| = 1$. Now $L_1 = \{r\}$, and with $u = r$ in the algorithm, we have $s_0 = T_2$ and $s_1 = T_2 \setminus N(v_1)$. Since $(v_1, v_2) \in E$, these two states are different, so that $P_2 = \{u_1^2, u_2^2\}$ with $s(u_1^2) = T_2$ and $s(u_2^2) = T_2 \setminus N(v_1)$. However, $v_2 \notin s(u_2^2)$ because $(v_1, v_2) \in E$, so that $L_2 = \{u_1^2\}$. Node u_1^2 can result in the addition of at most two more nodes to P_2 in when creating P_3, one with state s_0 and one with state s_1. Therefore, $|P_3| \leq 3 \leq |P_2| + |P_1| = 2 + 1 = 3 = \text{Fib}_4$, as desired.

For the inductive hypothesis, suppose $|P_j| \leq \text{Fib}_{j+1}$ for $j \leq j'$. We seek to show that

$$|P_{j'+1}| \leq \text{Fib}_{j'+2} \tag{2.2}$$

Consider the partition of the nodes in $P_{j'}$ into $X \cup Y$, where a node $u \in P_{j'}$ is in X if there exists a node $u' \in L_{j'-1}$ for which $b_1(u') = u$; i.e., there is a 1-arc ending at u directed out of a node in $L_{j'-1}$. All other nodes are in Y. We make three observations.

1. $|Y| \leq |P_{j'-1}|$

   The nodes in $P_{j'-1}$ can be partitioned into $L_{j'-1} \cup \bar{L}_{j'-1}$. All nodes in $P_{j'}$ either arise from a 0-arc or 1-arc directed out of $L_{j'-1}$ or are copies of the nodes in $\bar{L}_{j'-1}$ (these may be combined because their associated states may coincide). Only the nodes arising from 1-arcs directed out of nodes in $L_{j'-1}$ are in X, and the remaining nodes are in Y. There are at most $|L_{j'-1}| + |\bar{L}_{j'-1}|$ nodes in Y, including at most $|L_{j'-1}|$ from 0-arcs and at most $|\bar{L}_{j'-1}|$ copies of nodes from $P_{j'-1}$. Therefore, $|Y| \leq |L_{j'-1}| + |\bar{L}_{j'-1}| = |P_{j'-1}|$.

2. $|L_{j'}| \leq |Y|$

   We have that $L_{j'} \subseteq P_{j'}$. However, any node $u \in X$ must have $v_{j'} \notin s(u)$ because $(v_{j'-1}, v_{j'}) \in E$. Therefore $L_{j'} \subseteq P_{j'}$ and $|L_{j'}| \leq |Y|$.

3. $|P_{j'+1}| \leq 2|L_{j'}| + (|P_{j'}| - |L_{j'}|) = |P_{j'}| + |L_{j'}|$

   As in $P_{j'}$, the nodes in $P_{j'+1}$ arise from a 0-arc or 1-arc directed out of $L_{j'}$ or are

copies of the nodes in $\bar{L}_{j'}$ (these may be combined because their associated states may coincide). So each node in $L_{j'}$ gives rise to at most two nodes that are inserted $P_{j'+1}$, and each node in $\bar{L}_{j'}$ contributes at most one node to $P_{j'+1}$. The inequality follows.

Putting all three observations together, we get

$$|P_{j'+1}| \leq |P_{j'}| + |L_{j'}| \leq |P_{j'}| + |Y| \leq |P_{j'}| + |P_{j'-1}|,$$

We therefore have by induction that $|P_{j'+1}| \leq \text{Fib}_{j'+1} + \text{Fib}_{j'} = \text{Fib}_{j'+2}$, proving (2.2) for $k = 1$.

Now let $k > 1$. From above, we know that $\left|P_{v^1_{|V^1|}}\right| \leq \text{Fib}_{|V^1|+1}$. We first show that

$$\left|P_{v^1_{|V^1|+1}}\right| \leq \left|P_{v^1_{|V^1|}}\right| \tag{2.3}$$

Take any node $u \in P_{v^1_{|V^1|}}$. If $v^1_{|V^1|} \notin s(u)$ then this node is reproduced in $P_{v^1_{|V^1|+1}}$. If $v^1_{|V^1|} \in s(u)$, then $s_0 = s_1$. This is because $v^1_{|V^1|}$ is independent of all nodes in the remainder of the graph. Therefore, eliminating $v^1_{|V^1|}$ or eliminating $v^1_{|V^1|} \cup N\left(v^1_{|V^1|}\right)$ from $s(u)$ corresponds to the set of vertices in G. This may coincide with the state of some node that originally did not have $v^1_{|V^1|}$ in its state, but in either case at most one new node is added to $P_{v^1_{|V^1|+1}}$. We therefore have (2.3). This in turn implies that $\left|P_{v^1_{|V^1|+1}}\right| \leq \left|P_{v^1_{|V^1|}}\right| \leq \text{Fib}_{|V^1|+1} \leq \text{Fib}_{|V^1|+2}$, as desired. In addition, since consecutive P_j's differ in size by at most a factor of 2,

$$\left|P_{v^1_{|V^1|}+2}\right| \leq 2\left|P_{v^1_{|V^1|}+1}\right| \leq 2\left|P_{v^1_{|V^1|}}\right| \leq 2\text{Fib}_{|V^1|+1} \leq \text{Fib}_{|V^1|+1},$$

as desired.

Now, since the vertices in indices $v^1_{|V^1|}+1$ and $v^1_{|V^1|}+2$ are v^2_1 and v^2_2, respectively, and their corresponding P_j's are bounded by the desired Fibonacci numbers, we can apply the reasoning from the proof of $k = 1$ to bound the sizes of the P_j's until the end of set V^2, and by induction bound the remaining P_j's. □

2.9.2 Relaxed BDD Orderings

The orderings in Section 2.9.1 inspire variable ordering heuristics for generating relaxed BDD. We outline a few that are tested below. Note that the first two orderings are *dynamic*, in that we select the j-th vertex in the order based on the first $j-1$ vertices chosen and

the partially constructed BDD. In contrast, the last ordering is *static*, in that the ordering is determined prior to building the BDD.

random: Randomly select some vertex that has yet to be chosen. This may be used a stand-alone heuristic or combined with any of the following heuristics for the purpose of generating several relaxations.

minState: Select the vertex v_j appearing in the fewest number of states in P. This minimizes the size of L_j, given the previous selection of vertices $v_1, \ldots, v_{j-1}$, since the only nodes in P that will appear in L_j are exactly those nodes containing v_j in their associated state. Doing so limits the number of merging operations that need to be performed.

MPD: As proved above, a maximal path decomposition ordering of the vertices bounds the exact BDD width by the Fibonacci numbers, which grow slower than 2^j (the worst case). Hence this ordering limits the width of all layers, therefore limiting the number of merging operations necessary to build the BDD.

random: Randomly select some vertex that has yet to be chosen. We suggest this vertex selection not only as a stand alone variable ordering heuristic, but also as a heuristic that may be mixed with any of the following heuristics for the purpose of generating several relaxations.

minState: Select the next vertex v_j as the vertex appearing in the fewest number of states in P. This selection minimizes the size of L_j, given the previous selection of vertices $v_1, \ldots, v_{j-1}$, since the only nodes in P that will appear in L_j are exactly those nodes containing v_j in their associated state. Doing so limits the number of merging operations that need to be performed.

MPD: As mentioned above, it was shown in [9] that a Maximal Path Decomposition of the vertices in a graph yields an ordering that bounds the exact BDD width by the Fibonacci numbers, which grow slower than 2^j (the worst case). Hence this ordering limits the width of all layers, therefore also limiting the number of merging operations necessary to build the BDD.

2.10 Computational Experiments

In this section, we assess empirically the quality of bounds provided by a relaxed BDD. We first investigate the impact of various parameters on the bounds. We then compare our bounds with those obtained by an LP relaxation of a clique-cover model of the problem, both with and without cutting planes. We measure the quality of a bound by its ratio with the optimal value (or best lower bound known if the problem instance is unsolved). Thus a smaller ratio indicates a better bound.

We test our procedure on two sets of instances. The first set, denoted by `random`, consists of 180 randomly generated graphs according to the Erdös-Rényi model $G(n, p)$, in which each pair of n vertices is joined by an edge with probability p. We fix $n = 200$ and generate 20 instances for each $p \in \{0.1, 0.2, \ldots, 0.9\}$. The second set of instances, denoted by `dimacs`, is composed by the complement graphs of the well-known DIMACS benchmark for the maximum clique problem, obtained from `http://cs.hbg.psu.edu/txn131/clique.html`. These graphs have between 100 and 4000 vertices and exhibit various types of structure. Furthermore, we consider the maximum cardinality optimization problem for our test bed (i.e., $w_j = 1$ for all vertices v_j).

The tests ran on an Intel Xeon E5345 with 8 GB RAM in single core mode. The BDD method was implemented in C++.

2.10.1 Merging Heuristics

We tested the three merging heuristics presented in Section 2.8 on the `random` instance set. We set a maximum width of $W = 10$ and used variable ordering heuristic `MPD`. Figure 2.3 displays the resulting bound quality.

We see that among the merging heuristics tested, **minLP** achieves by far the tightest bounds. This behavior reflects the fact that infeasibility is introduced only at those nodes selected to be merged, and it seems better to preserve the nodes with the best bounds as in **minLP**. The plot also highlights the importance of using a structured merging heuristic, because `random` yielded much weaker bounds than the other techniques tested. In light of these results, we use **minLP** as the merging heuristic for the remainder of the experiments.

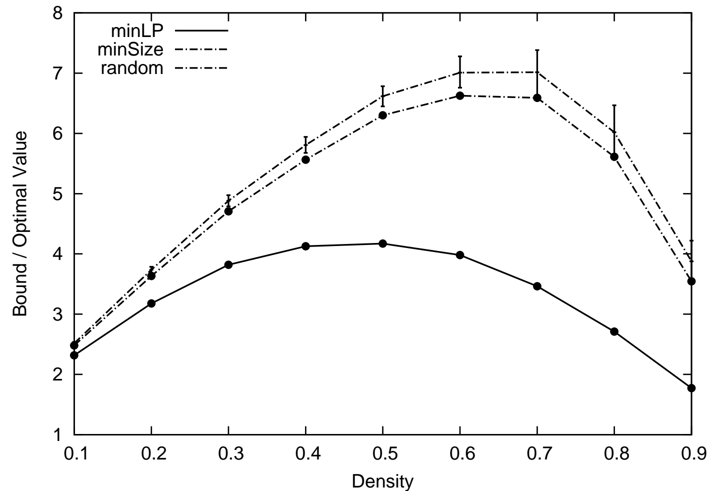


Figure 2.3: Bound quality vs. graph density for each merging heuristic, using the `random` instance set with `MPD` ordering and maximum BDD width 10. Each data point represents an average over 20 problem instances. The vertical line segments indicate the range obtained in 5 trials of the random heuristic.

2.10.2 Variable Ordering Heuristics

We tested the three variable ordering heuristics presented in Section 2.9 on the `random` instance set. The results (Fig. 2.4) indicate that the `MinState` ordering is the best of the three. This is particularly true for sparse graphs, because the number of possible node states generated by dense graphs is relatively small. We therefore use `MinState` ordering for the remainder of the experiments.

2.10.3 Bounds vs. Maximum BDD Width

The purpose of this experiment is to analyze the impact of maximum BDD width on the resulting bound. Figure 2.5 presents the results for instance `p-hat_300-1` in the `dimacs` set. The results are similar for other instances. The maximum width ranges from $W = 5$ to the value necessary to obtain the optimal value of 8. The bound approaches the optimal value almost monotonically as W increases, but the convergence is super-exponential in W.

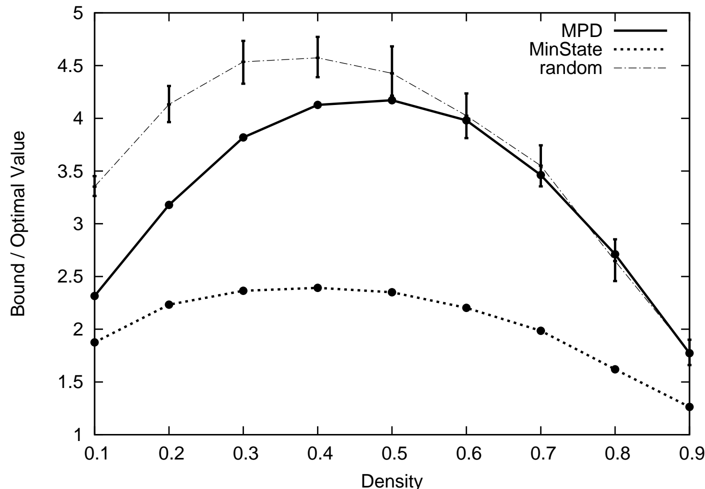


Figure 2.4: Bound quality vs. graph density for each variable ordering heuristic, using merge heuristic **minLP** and otherwise the same experimental setup as Fig. 2.3.

2.10.4 Comparison with LP Relaxation

We now address the key question of how BDD bounds compare with bounds produced by a traditional LP relaxation and cutting planes. To obtain a tight initial LP relaxation, we used a *clique cover* model [33] of the maximum independent set problem, which requires computing a clique cover before the model can be formulated. We then augmented the LP relaxation with cutting planes generated at the root node by the CPLEX MILP solver.

Given a collection $\mathcal{C} \subseteq 2^V$ of cliques whose union covers all the edges of the graph G, the clique cover formulation is

$$\begin{aligned} \max \quad & \sum_{v \in V} x_v \\ \text{s.t.} \quad & \sum_{v \in S} x_v \leq 1, \text{ for all } S \in \mathcal{C} \\ & x_v \in \{0, 1\}. \end{aligned}$$

The clique cover $\mathcal{C}$ was computed using a greedy procedure as follows. Starting with $\mathcal{C} = \emptyset$, let clique S consist of a single vertex v with the highest positive degree in G. Add to S the vertex with highest degree in $G \setminus S$ that is adjacent to all vertices in S, and repeat until no more additions are possible. At this point, add S to $\mathcal{C}$, remove from G all the edges of the

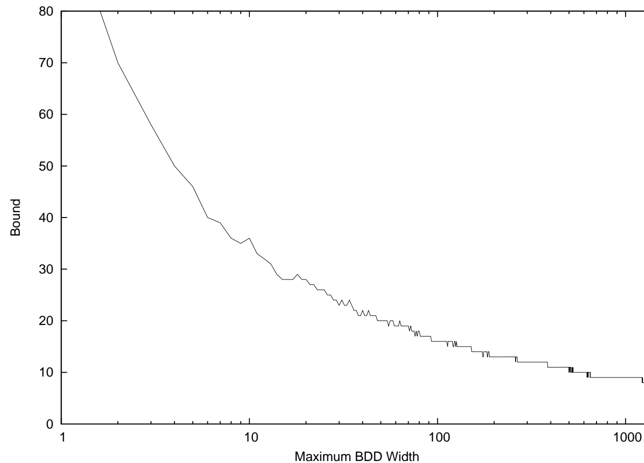


Figure 2.5: Relaxation bound vs. maximum BDD width for **dimacs** instance `p-hat_300-1`.

clique induced by S, update the vertex degrees, and repeat the overall procedure until G has no more edges.

We solved the LP relaxation with Ilog CPLEX 12.4. We used the interior point (barrier) option because we found it to be up to 10 times faster than simplex on the larger LP instances. To generate cutting planes, we ran the CPLEX MIP solver with instructions to process the root node only. We turned off presolve, because no presolve is used for the BDD method, and it had only a marginal effect on the results in any case. Default settings were used for cutting plane generation.

The results for **random** instances appear in Table 2.1 and are plotted in Fig. 2.6. The table displays geometric means, rather than averages, to reduce the effect of outliers. It uses shifted geometric means[1] for computation times. The computation times for LP include the time necessary to compute the clique cover, which is much less than the time required to solve the initial LP for **random** instances, and about the same as the LP solution time for **dimacs** instances.

The results show that BDDs with width as small as 100 provide bounds that, after taking means, are superior to LP bounds for all graph densities except 0.1. The computation time required is about the same overall—more for sparse instances, less for dense instances. The

[1] The shifted geometric mean of $v_1, \ldots, v_n$ is $g - \alpha$, where g is the geometric mean of $v_1 + \alpha, \ldots, v_n + \alpha$. We used $\alpha = 1$ second.

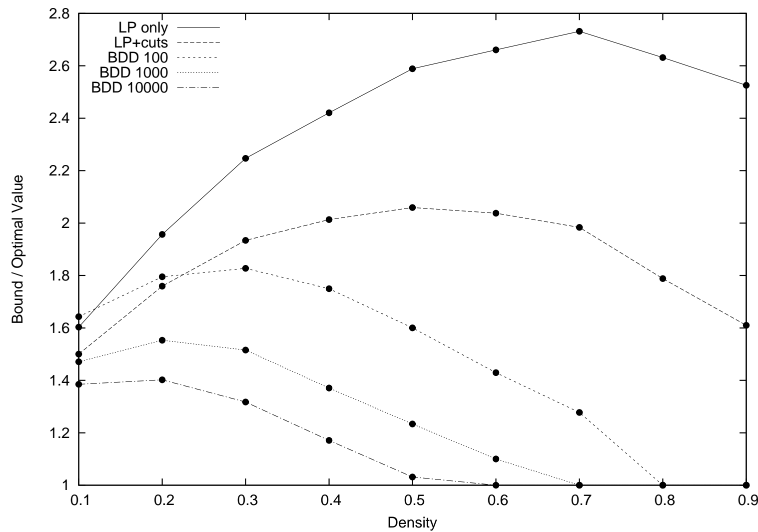


Figure 2.6: Bound quality vs. graph density for **random** instances, showing results for LP only, LP plus cutting planes, and BDDs with maximum width 100, 1000, and 10000. Each data point is the geometric mean of 20 instances.

scatterplot in Fig. 2.8 shows how the bounds compare on individual instances. The fact that almost all points lie below the diagonal indicates the superior quality of BDD bounds.

More important, however, is the comparison with the tighter bounds obtained by an LP with cutting planes, because this is the approach used in practice. BDDs of width 100 yield better bounds overall than even an LP with cuts, and they do so in less than 1% of the time. However, the mean bounds are worse for the two sparsest instance classes. By increasing the BDD width to 1000, the mean BDD bounds become superior for all densities, and they are still obtained in 5% as much time overall. Increasing the width to 10,000 yields bounds that are superior for every instance, as revealed by the scatter plot in Fig. 2.10. The time required is about a third as much as LP overall, but somewhat more for sparse instances.

The results for **dimacs** instances appear in Table 2.2 and Fig. 2.7, with scatter plots in Figs. 2.11–2.13. The instances are grouped into five density classes, with the first class corresponding to densities in the interval $[0, 0.2)$, the second class to the interval $[0.2, 0.4)$, and so forth. The table shows the average density of each class. Table 2.3 shows detailed results for each instance.

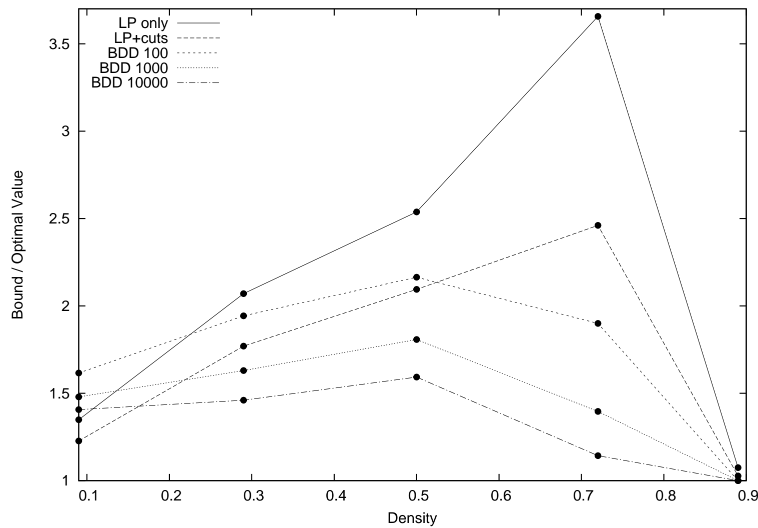


Figure 2.7: Bound quality vs. graph density for **dimacs** instances, showing results for LP only, LP plus cutting planes, and BDDs with maximum width 100, 1000, and 10000. Each data point is the geometric mean of instances in a density interval of width 0.2.

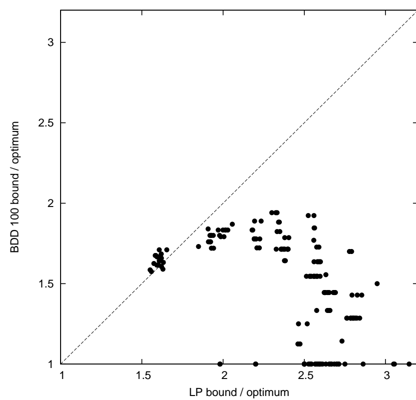


Figure 2.8: Bound quality for an LP relaxation (no cuts) vs. width 100 BDDs for **random** instances. Each data point represents one instance. The time required is about the same overall for the two types of bounds.

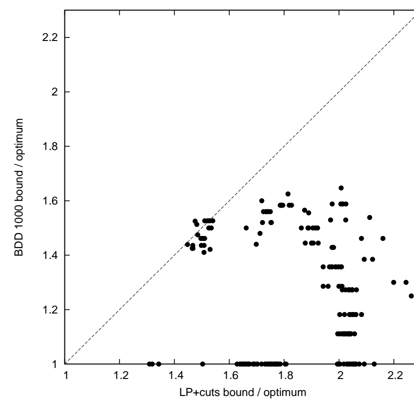


Figure 2.9: Bound quality for an LP relaxation with cuts vs. width 1000 BDDs for **random** instances. The BDD bounds are obtained in about 5% of the time required for the LP bounds.

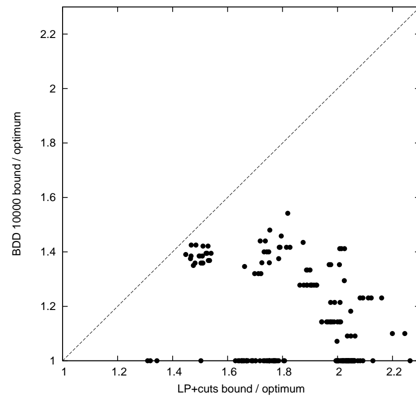


Figure 2.10: Bound quality for an LP relaxation with cuts vs. width 10000 BDDs for **random** instances. The BDD bounds are obtained in less time overall than the LP bounds, but somewhat more time for sparse instances.

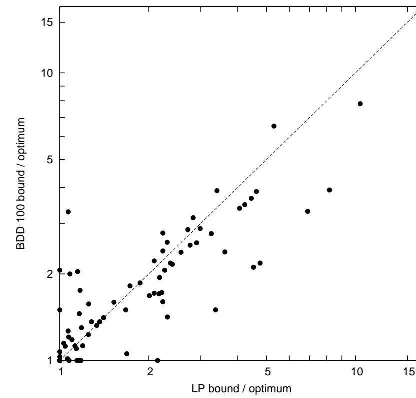


Figure 2.11: Bound quality for an LP relaxation (no cuts) vs. width 100 BDDs for **dimacs** instances. The BDD bounds are obtained in generally less time for all but the sparsest instances.

BDDs of width 100 provide somewhat better bounds than the LP without cuts, except for the sparsest instances, and the computation time is somewhat less overall. Again, however, the more important comparison is with LP augmented by cutting planes. BDDs of width 100 are no longer superior, but increasing the width to 1000 yields better mean bounds than LP for all but the sparsest class of instances. The mean time required is about 15% that required by LP. Increasing the width to 10,000 yields still better bounds and requires less time for all but the sparsest instances. However, the mean BDD bound remains worse for instances with density less than 0.2. We conclude that BDDs are generally faster when they provide better bounds, and they provide better bounds, in the mean, for all but the sparsest **dimacs** instances.

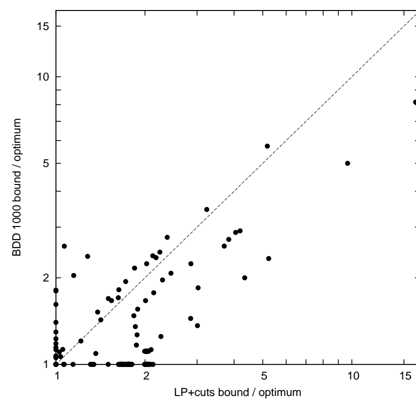


Figure 2.12: Bound quality for an LP relaxation with cuts vs. width 1000 BDDs for **dimacs** instances. The BDD bounds are obtained in about 15% as much time overall as the LP bounds.

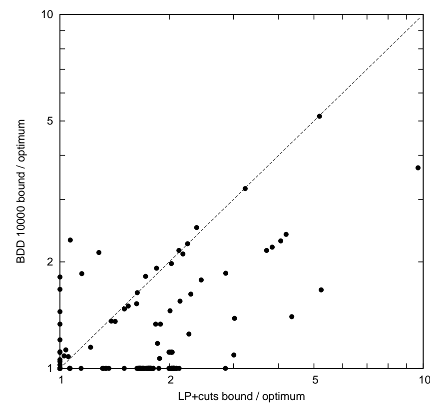


Figure 2.13: Bound quality for an LP relaxation with cuts vs. width 10000 BDDs for **dimacs** instances. The BDD bounds are generally obtained in less time for all but the sparsest instances.

Table 2.1: Bound quality and computation times for LP and BDD relaxations, using **random** instances. The bound quality is the ratio of the bound to the optimal value. The BDD bounds correspond to maximum BDD widths of 100, 1000, and 10000. Each graph density setting is represented by 20 problem instances.

	Bound quality (geometric mean)					Time in seconds (shifted geometric mean)				
	LP relaxation		*BDD relaxation*			*LP relaxation*		*BDD relaxation*		
Density	LP only	LP+cuts	100	1000	10000	LP only	LP+cuts	100	1000	10000
0.1	1.60	1.50	1.64	1.47	1.38	0.02	3.74	0.13	1.11	15.0
0.2	1.96	1.76	1.80	1.55	1.40	0.04	9.83	0.10	0.86	13.8
0.3	2.25	1.93	1.83	1.52	1.40	0.04	7.75	0.08	0.82	11.8
0.4	2.42	2.01	1.75	1.37	1.17	0.05	10.6	0.06	0.73	7.82
0.5	2.59	2.06	1.60	1.23	1.03	0.06	13.6	0.05	0.49	3.88
0.6	2.66	2.04	1.43	1.10	1.00	0.06	15.0	0.04	0.23	0.51
0.7	2.73	1.98	1.28	1.00	1.00	0.07	15.3	0.03	0.07	0.07
0.8	2.63	1.79	1.00	1.00	1.00	0.07	9.40	0.02	0.02	0.02
0.9	2.53	1.61	1.00	1.00	1.00	0.08	4.58	0.01	0.01	0.01
All	**2.34**	**1.84**	**1.45**	**1.23**	**1.13**	**0.05**	**9.15**	**0.06**	**0.43**	**2.92**

Table 2.2: Bound quality and computation times for LP and BDD relaxations, using **dimacs** instances. The bound quality is the ratio of the bound to the optimal value. The BDD bounds correspond to maximum BDD widths of 100, 1000, and 10000.

		Bound quality (geometric mean)					Time in seconds (shifted geometric mean)				
Avg.		*LP relaxation*		*BDD relaxation*			*LP relaxation*		*BDD relaxation*		
Density	Count	LP only	LP+cuts	100	1000	10000	LP only	LP+cuts	100	1000	10000
0.09	25	1.35	1.23	1.62	1.48	1.41	0.53	6.87	1.22	6.45	55.4
0.29	28	2.07	1.77	1.94	1.63	1.46	0.55	50.2	0.48	3.51	34.3
0.50	13	2.54	2.09	2.16	1.81	1.59	4.63	149	0.99	6.54	43.6
0.72	7	3.66	2.46	1.90	1.40	1.14	2.56	45.1	0.36	2.92	10.4
0.89	5	1.07	1.03	1.00	1.00	1.00	0.81	4.19	0.01	0.01	0.01
All	**78**	**1.88**	**1.61**	**1.78**	**1.54**	**1.40**	**1.08**	**27.7**	**0.72**	**4.18**	**29.7**

Table 2.3: Bound comparison for the `dimacs` instance set, showing the optimal value (Opt), the number of vertices (Size), and the edge density (Den). LP times correspond to clique cover generation (Clique), processing at the root node (CPLEX), and total time. The bound (Bnd) and computation time are shown for each BDD width. The best bounds are shown in boldface (either LP bound or one or more BDD bounds

Instance				LP with Cutting Planes				Relaxed BDD					
					Time (sec)			Width 100		Width 1000		Width 10000	
Name	Opt	Size	Den.	Bound	Clique	CPLEX	Total	Bnd	Sec	Bnd	Sec	Bnd	Sec
brock200_1	21	200	0.25	38.51	0	9.13	9.13	**36**	0.08	**31**	0.78	**28**	13.05
brock200_2	12	200	0.50	22.45	0.02	13.56	13.58	**17**	0.06	**14**	.45	**12**	4.09
brock200_3	15	200	0.39	28.20	0.01	11.24	11.25	**24**	0.06	**19**	0.70	**16**	8.31

Continued on next page

Instance				LP with Cutting Planes				Relaxed BDD					
					Time (sec)			Width 100		Width 1000		Width 10000	
Name	Opt	Size	Den.	Bound	Clique	CPLEX	Total	Bnd	Sec	Bnd	Sec	Bnd	Sec
brock200_4	17	200	0.34	31.54	0.01	9.11	9.12	**29**	0.08	**23**	0.81	**20**	10.92
brock400_1	27	400	0.25	66.10	0.05	164.92	164.97	68	0.34	**56**	3.34	**48**	47.51
brock400_2	29	400	0.25	66.47	0.04	178.17	178.21	69	0.34	**57**	3.34	**47**	51.44
brock400_3	31	400	0.25	66.35	0.05	164.55	164.60	67	0.34	**55**	3.24	**48**	47.29
brock400_4	33	400	0.25	66.28	0.05	160.73	160.78	68	0.35	**55**	3.32	**48**	47.82
brock800_1	23	800	0.35	96.42	0.73	1814.64	1815.37	**89**	1.04	**67**	13.17	**55**	168.72
brock800_2	24	800	0.35	97.24	0.73	1824.55	1825.28	**88**	1.02	**69**	13.11	**55**	180.45
brock800_3	25	800	0.35	95.98	0.72	2587.85	2588.57	**87**	1.01	**68**	12.93	**55**	209.72
brock800_4	26	800	0.35	96.33	0.73	1850.77	1851.50	**88**	1.02	**67**	12.91	**56**	221.07
C1000.9	68	1000	0.10	219.934	0.2	1204.41	1204.61	265	3.40	235	28.93	**219**	314.99
C125.9	34	125	0.10	41.29	0.00	1.51	1.51	45	0.05	**41**	0.43	**39**	5.73
C2000.5	16	2000	0.50	154.78	35.78	3601.41	3637.19	**125**	4.66	**80**	67.71	**59**	1207.69
C2000.9	77	2000	0.10	398.924	2.88	3811.94	3814.82	503	13.56	442	118.00	**397**	1089.96
C250.9	44	250	0.10	71.53	0.00	6.84	6.84	80	0.21	75	1.80	**67**	23.69
C4000.5	18	4000	0.50	295.67	631.09	3601.22	4232.31	**234**	18.73	**147**	195.05	**107**	3348.65
C500.9	57	500	0.10	124.21	0.03	64.56	64.59	147	0.85	134	7.42	**120**	84.66
c-fat200-1	12	200	0.92	**12.00**	0.04	0.95	0.99	**12**	0.00	**12**	0.00	**12**	0.00
c-fat200-2	24	200	0.84	**24.00**	0.05	0.15	0.2	**24**	0.00	**24**	0.00	**24**	0.00
c-fat200-5	58	200	0.57	61.70	0.07	35.85	35.92	**58**	0.00	**58**	0.00	**58**	0.00
c-fat500-10	126	500	0.63	**126.00**	1.89	2.80	4.69	**126**	0.01	**126**	0.01	**126**	0.01
c-fat500-1	14	500	0.96	16.00	1.03	27.79	28.82	**14**	0.02	**14**	0.01	**14**	0.01
c-fat500-2	26	500	0.93	**26.00**	0.81	7.71	8.52	**26**	0.01	**26**	0.00	**26**	0.01
c-fat500-5	64	500	0.81	**64.00**	1.51	3.05	4.56	**64**	0.01	**64**	0.01	**64**	0.01
gen200_p0.9_44	44	200	0.10	**44.00**	0.00	0.52	0.52	64	0.14	57	1.17	53	15.94
gen200_p0.9_55	55	200	0.10	**55.00**	0.00	2.04	2.04	65	0.14	63	1.19	61	15.74
gen400_p0.9_55	55	400	0.10	**55.00**	0.02	1.97	1.99	110	0.56	99	4.76	92	59.31

Continued on next page

Instance				LP with Cutting Planes				Relaxed BDD					
					Time (sec)			Width 100		Width 1000		Width 10000	
Name	Opt	Size	Den.	Bound	Clique	CPLEX	Total	Bnd	Sec	Bnd	Sec	Bnd	Sec
gen400_p0.9_65	65	400	0.10	**65.00**	0.02	3.08	3.1	114	0.55	105	4.74	94	56.99
gen400_p0.9_75	75	400	0.10	**75.00**	0.02	7.94	7.96	118	0.54	105	4.64	100	59.41
hamming10-2	512	1024	0.01	**512.00**	0.01	0.22	0.23	549	5.05	540	48.17	542	484.66
hamming10-4	40	1024	0.17	**51.20**	0.50	305.75	306.25	111	3.10	95	30.93	85	322.94
hamming6-2	32	64	0.10	**32.00**	0.00	0.00	0.00	**32**	0.01	**32**	0.09	**32**	1.20
hamming6-4	4	64	0.65	5.33	0.00	0.10	0.10	**4**	0.00	**4**	0.00	**4**	0.00
hamming8-2	128	256	0.03	**128.00**	0.00	0.01	0.01	132	0.26	136	2.45	131	25.70
hamming8-4	16	256	0.36	**16.00**	0.02	2.54	2.56	24	0.10	18	1.01	**16**	10.32
johnson16-2-4	8	120	0.24	**8.00**	0.00	0.00	0.00	12	0.02	**8**	0.10	**8**	0.23
johnson32-2-4	16	496	0.12	**16.00**	0.01	0.00	0.01	33	0.72	29	6.10	29	50.65
johnson8-2-4	4	28	0.44	**4.00**	0.00	0.00	0.00	**4**	0.00	**4**	0.00	**4**	0.00
johnson8-4-4	14	70	0.23	**14.00**	0.00	0.00	0.00	**14**	0.00	**14**	0.06	**14**	0.36
keller4	11	171	0.35	15.00	0.00	0.45	0.45	15	0.05	**12**	0.30	**11**	2.59
keller5	27	776	0.25	**31.00**	0.36	39.66	40.02	55	1.53	55	16.96	50	178.04
keller6	59	3361	0.18	**63.00**	55.94	3601.09	3657.03	194	37.02	152	361.31	136	3856.53
MANN_a27	126	378	0.01	**132.82**	0.00	1.31	1.31	152	0.46	142	3.71	136	41.90
MANN_a45	345	1035	0.00	**357.97**	0.01	1.47	1.48	387	2.83	367	26.73	389	285.05
MANN_a81	1100	3321	0.00	**1129.57**	0.07	11.22	11.29	1263	20.83	1215	254.23	1193	2622.59
MANN_a9	16	45	0.07	17.00	0.00	0.01	0.01	18	0.00	**16**	0.00	**16**	0.00
p_hat1000-1	10	1000	0.76	43.45	5.38	362.91	368.29	**33**	0.76	**20**	13.99	**14**	117.45
p_hat1000-2	46	1000	0.51	93.19	3.30	524.82	528.12	118	1.23	103	16.48	**91**	224.92
p_hat1000-3	68	1000	0.26	**152.74**	1.02	1112.94	1113.96	194	2.20	167	21.96	153	313.71
p_hat1500-1	12	1500	0.75	62.83	21.71	1664.41	1686.12	**47**	2.26	**28**	35.87	**20**	453.13
p_hat1500-2	65	1500	0.49	**138.13**	13.42	1955.38	1968.80	187	3.11	155	36.76	140	476.65
p_hat1500-3	94	1500	0.25	**223.60**	4.00	2665.67	2669.67	295	5.14	260	47.90	235	503.55
p_hat300-1	8	300	0.76	16.778	0.10	20.74	20.84	**12**	0.06	**9**	0.19	**8**	0.22

Continued on next page

Instance				LP with Cutting Planes				Relaxed BDD					
					Time (sec)			Width 100		Width 1000		Width 10000	
Name	Opt	Size	Den.	Bound	Clique	CPLEX	Total	Bnd	Sec	Bnd	Sec	Bnd	Sec
p_hat300-2	25	300	0.51	34.60	0.06	29.73	29.79	42	0.11	38	1.25	**34**	11.79
p_hat300-3	36	300	0.26	55.49	0.02	25.50	25.52	67	0.20	60	2.15	**54**	27.61
p_hat500-1	9	500	0.75	25.69	0.52	42.29	42.81	**19**	0.18	**13**	2.12	**9**	9.54
p_hat500-2	36	500	0.50	54.17	0.30	195.59	195.89	70	0.31	61	4.23	**53**	51.57
p_hat500-3	50	500	0.25	**86.03**	0.11	289.12	289.23	111	0.55	97	5.97	91	85.50
p_hat700-1	11	700	0.75	533.10	1.64	115.55	117.19	**24**	0.35	**15**	5.95	**12**	34.68
p_hat700-2	44	700	0.50	**71.83**	1.00	460.58	461.58	96	0.60	80	8.09	72	82.10
p_hat700-3	62	700	0.25	**114.36**	0.30	646.96	647.26	149	1.08	134	11.32	119	127.37
san1000	15	1000	0.50	16.00	43.14	180.46	223.60	19	1.14	**15**	15.01	**15**	99.71
san200_0.7_1	30	200	0.30	**30.00**	0.02	0.74	0.76	**30**	0.08	**30**	0.62	**30**	7.80
san200_0.7_2	18	200	0.30	**18.00**	0.02	1.55	1.57	19	0.06	**18**	0.50	**18**	6.50
san200_0.9_1	70	200	0.10	**70.00**	0.00	0.16	0.16	71	0.13	**70**	1.08	**70**	12.88
san200_0.9_2	60	200	0.10	**60.00**	0.00	0.49	0.49	66	0.13	**60**	1.14	**60**	14.96
san200_0.9_3	44	200	0.10	**44.00**	0.00	0.46	0.46	60	0.13	54	1.18	49	15.41
san400_0.5_1	13	400	0.50	**13.00**	1.09	10.08	11.17	**13**	0.19	**13**	1.27	**13**	5.00
san400_0.7_1	40	400	0.30	**40.00**	0.33	16.91	17.24	45	0.32	**40**	2.97	**40**	33.58
san400_0.7_2	30	400	0.30	**30.00**	0.31	12.22	12.53	39	0.32	32	3.50	**30**	38.96
san400_0.7_3	22	400	0.30	**22.00**	0.28	6.38	6.66	31	0.31	26	3.68	23	41.45
san400_0.9_1	100	400	0.10	**100.00**	0.02	6.52	6.54	123	0.56	107	4.66	**100**	57.46
sanr200_0.7	18	200	0.30	34.02	0.01	9.00	9.01	**31**	0.08	**28**	0.82	**24**	11.88
sanr200_0.9	42	200	0.10	59.60	0.00	3.32	3.32	67	0.14	60	1.17	**57**	15.51
sanr400_0.5	13	400	0.50	39.30	0.13	281.21	281.34	**31**	0.21	**24**	4.09	**18**	35.88
sanr400_0.7	21	400	0.30	60.05	0.06	168.64	168.70	**58**	0.30	**47**	3.52	**39**	51.93

2.11 Conclusions

In this chapter we presented a novel method, based on binary decision diagrams (BDDs), for obtaining bounds on the optimal value of discrete optimization problems. As a test case, we applied the technique to the maximum independent set problem. We found that the BDD-based bounding procedure often yields better bounds, in less time, than a state-of-the-art mixed-integer solver obtains at the root node for a tight integer programming model.

The performance of both BDD and conventional relaxations is sensitive to the density of the graph. We found, however, that BDDs yield tighter bounds in less time, taking the geometric mean, for random instances of all density classes. For a well-known set of benchmark instances, BDDs provide better mean bounds in less time for all but the sparsest class of instances (i.e., all but those with density less than 0.2). We obtained these results using a barrier LP solver that is generally faster than simplex for these instances.

A further advantage of BDD relaxations is that the quality of the bound can be continuously adjusted by controlling the maximum width of the BDD. This allows one to invest as much or little time as one wishes in improving the quality of the bound. In addition, BDD-based bounds can be obtained for combinatorial problems that are not formulated as mixed integer models. Unlike LP relaxations, BDD relaxations do not presuppose that the constraints take the form of linear inequalities.

BDD bounds can be rapidly updated during a search procedure, much as the LP can be reoptimized after branching. This is achieved simply by removing arcs of the BDD that correspond to excluded values of the branching variable, and recomputing the shortest (or longest) path. Nonetheless, due to the speed at which BDDs can be constructed, it may be advantageous to rebuild the BDD from scratch, so as to obtain a relaxation that is suited to the current subproblem. One may be able to adjust the BDD width to obtain a bound that is just tight enough to fathom the current node of the search tree, thus saving time. These remain as research issues.

The above results suggest that BDD-based relaxations may have promise as a general technique for bounding the optimal value of discrete problems. The BDD algorithms presented here are relatively simple, compared with the highly developed technology of LP and mixed-integer solvers, and nonetheless improve the state of the art for at least one prob-

lem class. Future research may yield improvements in BDD-based bounding and extend its usefulness to a broader range of discrete optimization problems.

Chapter 3

Tightening Bounds from Relaxed Decision Diagrams

3.1 Introduction

Binary Decision Diagrams (BDDs) [2, 50, 16] provide compact graphical representations of Boolean functions, and have traditionally been used for circuit design and formal verification [44, 50]. More recently, however, BDDs and their generalization Multivalued Decision Diagrams (MDDs) [46] have been used in Operations Research for a variety of purposes, including cut generation [6], vertex enumeration [8], and post-optimality analysis [36, 37].

In this chapter, we examine the use of BDDs and MDDs as relaxations for combinatorial optimization problems. Relaxation MDDs were introduced in [3] as a replacement for the domain store relaxation, i.e., the Cartesian product of the variable domains, that is typically used in Constraint Programming (CP). MDDs provide a richer data structure that can capture a tighter relaxation of the feasible set of solutions, as compared with the domain store relaxation. In order to make this approach scalable, MDD relaxations of limited size are applied. Various methods for compiling these discrete relaxations are provided in [38]. The methods described in that chapter focus on iterative splitting and edge filtering algorithms that are used to tighten the relaxations. Similar to classical domain propagation,

such MDD propagation algorithms have been developed for individual (global) constraints, including inequality constraints, equality constraints, *alldifferent* constraints and *among* constraints [38, 39].

The focus of the current work is the application of limited-width MDD relaxations in the context of optimization problems. We explore two main topics. Firstly, we investigate a new method for building approximate MDDs. We introduce a top-down compilation method based on approximating the set of completions of partially assigned solutions. This procedure differs substantially from the ideas in [3] in that we do not compile the relaxation by splitting vertices, but by merging vertices when the size of the partially constructed MDD grows too large.

Secondly, and more specific to optimization, we introduce a method to improve the lower bound provided by an MDD relaxation. It is somewhat parallel to a cutting plane algorithm in that it "cuts off" infeasible solutions so as to tighten the bound. Unlike cutting planes, however, it can begin with any valid lower bound, perhaps obtained by another method, and tighten it. The bound becomes tighter as more time is invested.

The resulting mechanism is a pure inference algorithm that can be used analogously to a pure cutting plane algorithm. We envision, however, that MDD relaxations would be most profitably used as a bounding technique in conjunction with a branch-and-bound search, much as separation algorithms are used in integer programming. Nonetheless we find in this chapter that, even as a pure inference algorithm, MDD relaxation can outperform state-of-the-art integer programming technology on specially structured instances.

One advantage of an MDD relaxation is that it is always easy to solve (as a shortest path problem) whether the original problem is linear or nonlinear. This suggests that MDDs might be most competitive on nonlinear discrete problems. Nonetheless we deliberately put MDDs at a competitive disadvantage by applying them to a problem with linear inequality constraints—namely, to the set covering problem, which is well suited to integer programming methods.

We compare the strength of bounds provided by MDDs with those provided by the linear programming relaxation and cutting planes. We also compare the speed with which MDDs (used as a pure inference method) and integer programming solve the problem. We find

that MDDs are much superior to conventional integer programming when the ones in the constraint matrix lie in a relatively narrow band. That is, the matrix has relatively small bandwidth, meaning that the maximum distance between any two ones in the same row is limited.

The bandwidth of a set covering matrix can often be reduced, perhaps significantly, by reordering the columns. Thus MDDs can solve a given set covering problem much more rapidly than integer programming if its variables can be permuted to result in a relatively narrow bandwidth. Algorithms and heuristics for minimum bandwidth ordering are discussed in [54, 17, 21, 24, 34, 55, 60, 66].

The remainder of the chapter is organized as follows. In Section 3.2 we define MDDs more formally and introduce notation. In Section 3.3 we describe a new top-down compilation method for creating relaxation MDDs. In Section 3.4 we present our value enumeration scheme to produce lower bounds. In Section 3.5 we discuss applying the ideas of the chapter to set covering problems. In Section 3.6 we report on experiments results where we apply the ideas of the chapter to set covering problems. We conclude in Section 3.7.

3.2 Preliminaries

In this work a *Multivalued Decision Diagram* (MDD) is a layered directed acyclic multigraph whose nodes are arranged in $n+1$ layers, $L_1, L_2, \ldots, L_{n+1}$. Layers L_1 and L_{n+1} consist of single nodes; the root r and the terminal t, respectively. All arcs in the MDD are directed from nodes in layer j to nodes in layer $j+1$.

In the context of Constraint Satisfaction Problems (CSPs) or Constraint Optimization Problems (COPs), we use MDDs to represent assignments of values to variables. A CSP is specified by a set of constraints $C = \{C_1, C_2, \ldots C_m\}$ on a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ with respective finite domains $D_1, \ldots, D_n$, and a COP is specified by a CSP together with an objective function f to be minimized. By a *solution* to a CSP (COP) we mean an assignment of values to variables where the values assigned to the variables appear in their respective domains. By a *feasible solution* we mean a solution that satisfies each of the constraints in C, and the *feasible set* is the set of all feasible solutions.

For a COP, an *optimal* solution is a feasible solution x^* such that for any other feasible solution $\tilde{x}, f(x^*) \leq f(\tilde{x})$.

We use MDDs to represent a set of solutions to a CSP, or COP, as follows. We let the layers $L_1, \ldots, L_n$ correspond to the problem variables $x_1, \ldots, x_n$, respectively. Node $u \in L_j$ has label $\text{var}(u) = j$, representing its variable index. Arc (u, v) with $\text{var}(u) = j$ is labeled with *arc domain* $d_{u,v}$, by an element of the domain of variable x_j, i.e., $d_{u,v} \in D_j$. All arcs directed out of a node must have distinct labels.

A path p from node u_i to node u_k, $i < k$, along arcs $a_i, a_{i+1}, \ldots, a_{k-1}$ corresponds to the assignment of the values d_{a_j} to the variables x_j, for $j = i, i+1, \ldots, k-1$. In particular, we see that any path from the root r to the terminal $t, p = (a_1, \ldots, a_n)$, corresponds to the solution x^p, where $x^p_j = d_{a_j}$. We note that as an MDD is a multi-graph, two paths p_1, p_2, along nodes $r = u_1, \ldots, u_n, t$ may correspond to multiple solutions as there may be multiple arcs from u_j to u_{j+1} corresponding to different assignments of values to the variable x_j.

The set of solutions represented by MDD M is $\text{Sol}(M) = \{x^p | p \in P\}$ where P is the set of paths from r to t. The *width* of layer L_j is given by $\omega_j = |L_j|$, and the *width* of MDD M is given by $\omega(M) = \max_{j \in \{1,2,\ldots,n\}} \omega_j$. The *size* of M is denoted by $|M|$, the number of nodes in M.

For a given CSP $\mathcal{P}$, let $X(\mathcal{P})$ be the set of feasible solutions for $\mathcal{P}$. An *exact* MDD M for $\mathcal{P}$ is any MDD for which $\text{Sol}(M) = X(\mathcal{P})$. A *relaxation* MDD M_{rel} for $\mathcal{P}$ is any MDD for which $\text{Sol}(M_{\text{rel}}) \supseteq X(\mathcal{P})$. For the purposes of this chapter, relaxation MDDs are of limited width, in that we require that $\omega_j \leq W$, for some predefined W. This ensures that the relaxation has limited size which is necessary since even for single constrained problems, the feasible set may correspond to an MDD of exponential size (for example inequality constrained problems [7]).

Finally, we note that for a large class of objective functions (e.g., for separable functions), optimizing over the solutions represented by an MDD corresponds to finding a shortest path in the MDD [3]. For example, given a linear objective function $\min cx$, we associate with each arc (u, v) in the MDD a *cost* $c(u, v)$, where $c(u, v) = c_{\text{var}(u)} \cdot d_{u,v}$. Then it is clear that a shortest path from r to t corresponds to the lowest cost solution represented by the MDD.

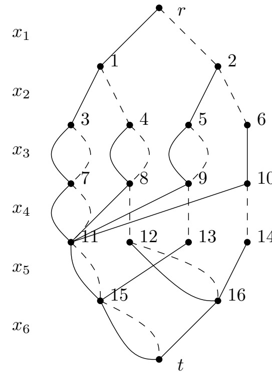


Figure 3.1: Exact MDD for Example 1.

Example 1 As an illustration, consider the CSP consisting of binary variables $x_1, x_2, \ldots, x_6$, and constraints

$$C_1: \quad x_1 + x_2 + x_3 \geq 1,$$
$$C_2: \quad x_1 + x_4 + x_5 \geq 1,$$
$$C_3: \quad x_2 + x_4 + x_6 \geq 1.$$

An exact MDD representation of the feasible set is given in Fig. 3.1, where arc (u, v) being solid/dashed corresponds to the arc setting var(u) to 1/0.

□

3.3 Top-Down MDD Compilation

As discussed above, there are several methods that can be used to construct both exact and approximate MDDs. In this section we propose a new top-down method for creating approximate MDDs.

3.3.1 Exact Top-Down Compilation

We first discuss an exact top-down compilation method, which is based on the notion of *node equivalence*.

Given a path p from r to u, let $F(p)$ be the set of feasible completions of the corresponding

partial assignment. That is, if $(x_1, \ldots, x_k) = (d_1, \cdots, d_k) = d$ is the partial assignment represented by p, then $F(p) = \{y \in D_{k+1} \times \cdots \times D_n | (d, y) \text{ is feasible } \}$. We say that two paths p, p' from r to the same layer are *equivalent* if $F(p) = F(p')$.

Analogously, we define $F(u)$ to be the set of completions at node u, so that $F(u) = \bigcup_{p \in P} F(p)$, where P is the set of paths from r to u. We note that in an exact MDD all paths terminating at a node u are equivalent.

A *node equivalence test* determines when two nodes u, u' on the same layer have the same set of feasible completions. In other words, this test determines when $F(u) = F(u')$. Testing whether two nodes have the same set of feasible completions requires maintaining a *state* I_u at each node [39]. The state of node u should contain all facts about the paths ending at u to run an equivalence test. In addition, it is useful to know when a partial assignment cannot be completed to a feasible solution for a CSP. In such a case, we let the state of such a path, or more generally a node, be $\hat{0}$, to signal that there are no completions of this path/node.

Now, using a properly defined node equivalence test, one can create an exact MDD using Algorithm 4. Given that layers $L_1, \ldots, L_j$ have been created, we examine the nodes in L_j one by one. When examining node u, for each domain value $d \in D_j$ we calculate the new state I_{new} that results from adding $x_j = d$ to the partial paths ending at u. If no other nodes on layer L_{j+1} have the same state (i.e. the same set of feasible completions) we add a new node v to L_{j+1} and the arc (u, v) with arc domain d, and set $I_v = I_{\text{new}}$. If however there is some node $w \in L_{j+1}$ with $I_w = I_{new}$ we know that all paths starting at r, ending at u and having $x_j = d$ will have the same set of feasible completions as w. Therefore, we simply add the arc (u, w) with arc domain d.

We will be modifying Algorithm 4 later to create approximate MDDs. First, however, we discuss specific exact MDDs for the feasible set *satisfying a single equality constraint*. Such MDDs will be applied in our value enumeration method for tightening lower bounds, presented in Section 3.4.

Lemma 5 *Let $\mathcal{P}$ be a CSP on n binary variables with the single constraint $\sum_{j=1}^{n} c_j x_j = c$, for a given integer c, and integer coefficients $c_j \geq 0$. An exact MDD for $\mathcal{P}$ has maximum width $c + 1$.*

Algorithm 4 Top-Down MDD Compilation

```
1: L_1 = {r}
2: for j = 1 to n do
3:   L_{j+1} = ∅
4:   for all u ∈ L_j do
5:     for all d ∈ D(x_j) do
6:       calculate I_new, the state for all paths starting at r, ending at u, and including
         x_j = d
7:       if I_new ≠ 0̂ then
8:         if there exists w ∈ L_{j+1} with I_w = I_new then
9:           add arc (u, w) with d_{u,w} = d
10:        else
11:          add node v to L_{j+1}
12:          add arc (u, v) with d_{u,v} = d
13:          set I_v = I_new
```

Proof. We apply Algorithm 4. Given a node u, let p be any path from r to u, and let $a_1, \ldots, a_k$ be the arcs along this path, which set variables $x_1, \ldots, x_k$ to the arc domain values $d_{a_1}, \ldots, d_{a_k}$. We define $I_u = \sum_{j=1}^{k} c_j \cdot d_{a_j}$. Using this label as the state of node u we see that two nodes u and v have the same set of feasible completions if and only if $I_u = I_v$. In addition, if $I_w \geq c + 1$ for some node w, it is clear that all paths from r to w have no feasible completions. Therefore we can have at most $c + 1$ nodes on any layer. □

We note that Lemma 5 is very similar to the classical pseudo-polynomial characterization of knapsack constraints.

3.3.2 Approximate Top-Down Compilation

In general, an exact MDD representation of all feasible solutions to a CSP may be of exponential size, and therefore generating exact MDDs for combinatorial optimization problems is not practical. In light of this we use relaxation MDDs to approximate the set of feasible solutions. In this section we outline one possible method for generating approximate MDDs,

Algorithm 5 Top-Down Relaxation Compilation

1: **while** $|S_{j+1}| > W$ **do**
2: **select** nodes $u_1, u_2 \in S_{j+1}$
3: create node u
4: for every arc directed at u_1 or u_2 redirect arc to u with the same arc domain
5: $I(u) = I(u_1) \oplus I(u_2)$
6: $S_{j+1} \leftarrow S_{j+1} \setminus \{u_1, u_2\} \cup \{u\}$

by modifying Algorithm 4.

In order to create a relaxation MDD we merge nodes during the top-down compilation method presented in Algorithm 4 when the width of layer j exceeds a certain preset maximum allotted width W. To accomplish this, we select two nodes and modify their states in a relaxed fashion, ensuring that all feasible solutions will remain in the MDD when it is completed. More formally, if we select nodes u_1 and u_2 to merge, we need to modify their states I_{u_1}, I_{u_2} in such a way as to make them equivalent with respect to the equivalence test used to merge nodes during the top-down compilation. We define a certain *relaxation operation* $\oplus$ on the state of nodes as follows.[1] If for nodes u_1 and u_2 we change their associated states to $I(u_1) \oplus I(u_2)$, any feasible completion of the paths from the root to u_1 and u_2 will remain when the terminal is reached. This is outlined in Algorithm 5, which is to be inserted between lines 17 and 18 in Algorithm 4. In Section 3.5 we describe such an operation in detail, for set covering problems.

The quality of the relaxation MDD generated using the modification of Algorithm 4 hinges largely on the method used for selecting two nodes to combine. We propose several heuristics for this choice in the following table:

Name	Node selection method
H_1	select u_1, u_2 uniformly at random among all pairs in S_{j+1}
H_2	select u_1, u_2 such that $f(u_1), f(u_2) \geq f(v), \forall v \in S_{j+1}, v \neq u_1, u_2$
H_3	select u_1, u_2 such that I_{u_1} and I_{u_2} are *closest* among all pairs in S_{j+1}

[1] Here we follow the notation $\oplus$ that was used in [39] for their analogous operation for aggregating node information.

The rationale behind each of the methods are the following. Method H_1 calls for randomly choosing which nodes to combine. Randomness often helps in combinatorial optimization and applying it in this context may work as well. H_2 combines nodes that have the greatest shortest path lengths. For this we let $f(u)$ be the shortest path length from the root to u in the partially constructed MDD. Choosing such a pair of nodes allows for approximating the set of feasible solutions in parts of the MDD where the optimal solution is unlikely to lie, and retaining the exact paths in sections of the MDD where the optimal solution is likely to lie. H_3 combines nodes that have similar states. For particular types of states and equivalence tests, we must determine the notion of *closest*. This method is sensible because these nodes will most likely have similar sets of completions, allowing the relaxation to better capture the set of feasible solutions.

3.4 Value Enumeration

We next discuss the application of MDD relaxations for obtaining lower bounds on the objective function, in the context of COPs. We propose to obtain and strengthen these bounds by means of successive value enumeration. Value enumeration is a method that can be used to increase any lower bound on a COP via a relaxation MDD.

Suppose we have generated a relaxation MDD M_{rel}. We then generate an MDD representing every solution in M_{rel} with objective function value equal to the best lower bound. There are several ways to accomplish this, but in general this MDD can have exponential size. However, for some important cases the MDD representing every solution equal to a particular value has polynomial size.

For example, suppose we have a COP with objective function equal to the sum of the variables, i.e., $f(x) = \sum_{j=1}^{n} x_j$, where we assume that the variable domains are integral. Given a lower bound z_{LB}, the reduced MDD for the set of solutions with objective value equal to z_{LB}, $M_{z_{LB}}$, has width $\omega(M_{z_{LB}}) = z_{LB} + 1$, by Lemma 5. The same holds for other linear objective functions as well.

In any case, suppose we have the desired MDD $M_{z_{LB}}$, where $\mathrm{Sol}(M_{z_{LB}})$ is the set of all solutions with objective value equal to z_{LB}. Now, consider the set of solutions $S =$

$\text{Sol}(M_{z_{LB}}) \cap \text{Sol}(M_{\text{rel}})$. As this is the intersection between the solutions represented by the relaxation and every solution equal to the lower bound z_{LB}, showing that there is no feasible solution in S allows us to increase the lower bound.

Constructing an MDD M representing the set of solutions $S = \text{Sol}(M_{z_{LB}}) \cap \text{Sol}(M_{\text{rel}})$ can be done in time $\mathcal{O}(|M_{z_{LB}}| \cdot |M_{\text{rel}}|)$ and has maximum width $\omega(\tilde{M}) \leq \omega(M_{z_{LB}}) \cdot \omega(M_{\text{rel}})$ [16]. As the width of $M_{z_{LB}}$ has polynomial size for certain objective functions and the width of M_{rel} is bounded by some preset W, the width of the resulting MDD will not grow too large in these cases.

The value enumeration scheme proceeds by enumerating all of the solutions in M. If we find a feasible solution, we have found a witness for our lower bounds. Otherwise, we can increase the lower bound by 1. Of course, this method is only practical if we can enumerate these paths efficiently.

Observe that we do not need to start the value enumeration scheme with the value of the shortest path in the original MDD. In fact, we can start with any lower bound. For example, we can use LP to find a strong lower bound and then apply this procedure to any relaxation MDD.

As described above, in order to increase the bound, we are required to certify that none of the paths in M correspond to feasible solutions. Of course this can be done by a naive enumeration of all of the paths in M. However, we use MDD-based CP, as described in [3], in unison with a branching procedure to certify this. In particular we apply MDD filtering algorithms to reduce the size of the MDD $M_{z_{LB}}$, based on the constraints that constitute the COP. In Section 3.5.3 we will describe a new MDD filtering algorithm that we apply to set covering problems.

3.5 Application to Set Covering

In this section we describe how to apply the ideas of the chapter to set covering problems. We describe a node equivalence test and the state that is necessary to carry out the test. We also describe the operation $\oplus$ that can be used to change the states of the nodes so that we can generate relaxation MDDs.

3.5.1 Equivalence Test

The well-studied set covering problem is a COP with n binary variables and m constraints, each on a subset C_i of the variables, which require that $\sum_{j \in C_i} x_j \geq 1, i \in \{1, \ldots, m\}$. The objective is to minimize the sum of the variables (or a weighted sum).

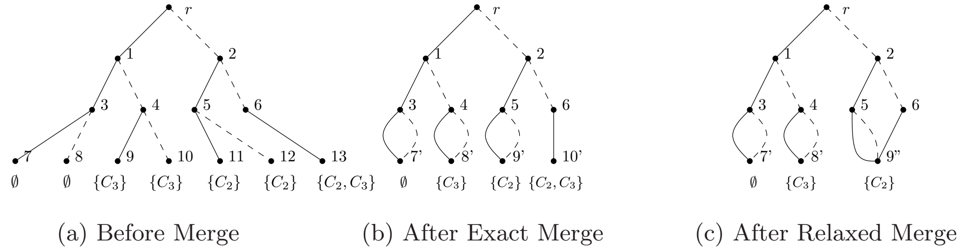
The first step in applying our top-down compilation method is defining an equivalence test between partial assignments of values to variables. For set covering problems we do this by equating a set covering instance with its equivalent logic formula. Each constraint C_i can be viewed as a clause $\vee_{j \in C_i} x_j$ and the set covering problem is equivalent to satisfying $F = \bigwedge_i \vee_{j \in C_i} x_j$.

Using this interpretation of set covering problems, one can develop a complete equivalence test by removing clauses that are implied by other clauses. Clause C *absorbs* clause D if all of the literals of C are contained in D. In such a case, satisfying clause C implies that clause D will be satisfied. As an example, consider the two clauses $C = (x_1 \vee x_2)$ and $D = (x_1 \vee x_2 \vee x_3)$. It is clear that if some literal in C is set to `true` then clause D will be satisfied.

Therefore, to develop the equivalence test, for any partial assignment x we delete any clause C_i for which there exists a variable in the clause that is already set to 1, and then delete all absorbed clauses, resulting in the logical formula $F(x)$. We let I_x be the set of clauses which remain in $F(x)$. Doing so ensures that two partial assignments x^1 and x^2, will have the same set of feasible completions if and only if $I_{x^1} = I_{x^2}$. Note that since each literal is positive in all clauses of a set covering instance, this test can be performed in polynomial time [41].

To create an exact MDD for a set covering instance (using Algorithm 4), we let the state I_u at node u be equal to I_x for the partial assignment given by the arc domains on all paths from the root to u. Two nodes u and v will have the same set of feasible completions if and only if $I_u = I_v$, and so the node equivalence test simply compares I_u with I_v.

Example 2 Continuing Example 1, we interpret the constraints C_1, C_2, and C_3 as set covering constraints. In Fig. 3.2(a) we see the result of applying the top-down compilation algorithm (following the variable order $x_1, x_2, \ldots, x_6$) and never combining nodes based on



(a) Before Merge (b) After Exact Merge (c) After Relaxed Merge

Figure 3.2: Relaxed MDD Construction for the Set Covering Problem

their associated states , for the first three layers of the MDD. Below the bottom nodes, we depict the states of the partially constructed paths ending at those nodes. For example, along this path $(r, 2, 5, 11)$, variables x_2 and x_3 are set to 1. Therefore, constraints C_1 and C_3 are satisfied for any possible completion of this path, and so the state at node 11 is C_2. Since node 11 and node 12 have the same state, we can combine these nodes into node 9', as shown in Fig. 3.2(b). Similarly, nodes 7 and 8 are combined into node 7' and nodes 9 and 10 are combined into node 8'. □

3.5.2 Relaxation Operation

We next discuss our relaxation operator $\oplus$ that is applied to merge two nodes in a layer. For set covering problems, we let $\oplus$ represent the typical set intersection. As an illustration, for the instance in Example 2, suppose we decided that we want to decrease the width of layer 4 by 1. We would select two nodes (in Fig. 3.2(b) we select nodes 9' and 10') and combine them (making node 9" as seen in Fig. 3.2(c)), modifying their states to ensure that all feasible paths remain upon completing the MDD. Notice that by taking the intersection of the states of the nodes 9' and 10' we now label 9" with C_2. Before merging the nodes, all partial paths ending at node 10' needed a variable in both constraint C_2 and C_3 to be set to 1. After taking the intersection, we are relaxing this condition, and only require that for all partial paths ending at 9", all completions of this path will set some variable in constraint C_2 to 1, ignoring that this needs to also happen for constraint C_3.

3.5.3 Filtering

As discussed above, during the value enumeration procedure, it is desirable to perform some MDD filtering to decrease the search space. This filtering can be applied to arc domain values, as described in [3], but also to the states represented in the nodes themselves, as we will describe here in the context of set covering problems.

We associate two 0/1 m-dimensional state variables, $s(v), z(v)$, to each node v in the MDD. The value $s(v)_i$ will be 1 if for all paths from the root to v, there is no variable in constraint C_i which is set to 1. Similarly, $z(v)_i$ will be 1 if for all paths from v to the terminal, there is no variable in C_i which is set to 1.

Finding the values $s(v)_i, z(v)_i$ is easily accomplished by the following simple algorithm. Start with $s(r)_i = 1$ for all i. Now, let node v have parents $u_1, u_2, \ldots, u_k$, and each edge (u_p, u) fixes variable x_j to value $v_p \in \{0, 1\}$. Then,

$$s(v)_i = \prod_{p=1}^{k} s'(u_p)_i,$$

where

$$s'(u_p)_i = \begin{cases} 0 & \text{if } x_j \in C_i \text{ and } v_p = 1 \\ s(u_p)_i & \text{otherwise} \end{cases}$$

The values $z(v)_i$ are calculated in the same fashion, except switching the direction of all arcs in the MDD and starting with $z(t)_i = 1$, where t is the terminal of the MDD.

A node v can now be eliminated whenever there is an index i such that $s(v)_i = z(v)_i = 1$. This is because for all paths from r to v there is no variable in C_i set to 1, and on all paths from v to t, there is no variable in C_i set to 1.

We note here that as in domain store filtering, certain propagators for MDDs are *idempotent*, in that reapplying the filtering algorithm with no additional changes results in no more filtering. The filtering algorithm presented here is not idempotent, i.e, applying it multiple times could result in additional filtering. In our computational experiments we address how this impacts the efficiency of the overall method.

3.6 Computational Results

In this section, we present experimental results on randomly generated set covering instances. Our results provide evidence that relaxations based on MDDs perform well when the constraint matrix of a set covering instance has a small bandwidth. We test this by generating random set covering instances with varying bandwidths and comparing solution times via pure-IP (using CPLEX), pure-MDD, and a hybrid MDD-IP method.

In all of the reported results, unless specified otherwise, we apply our MDD-based algorithm until it finds a feasible solution. That is, we solve these set covering problems by continuously improving the relaxation through our value enumeration scheme until we find a feasible (optimum) solution.

3.6.1 Bandwidth and the Minimum Bandwidth Problem

The *bandwidth* of a matrix A is defined as

$$b_w(A) = \max_{i \in \{1,2,\ldots,m\}} \{ \max_{j,k: a_{i,j}, a_{i,k}=1} \{j-k\} \}.$$

The bandwidth represents the largest distance, in the variable ordering given by the constraint matrix, between any two variables that share a constraint. The smaller the bandwidth, the more structured the problem, in that the variables participating in common constraints are close to each other in the ordering. The *minimum bandwidth problem* seeks to find a variable ordering that minimizes the bandwidth [54, 17, 21, 24, 35, 55, 60, 66]. This underlying structure, when present in A, can be captured by MDDs and results in good computational performance.

3.6.2 Problem Generation

To test the statement that MDD based relaxations provide strong relaxations for structured problems, we generate set covering instances with a fixed constraint matrix density d (the number of ones in the matrix divided by $n \cdot m$) and vary the bandwidth b_w of the constraint matrix.

We generate random instances with a fixed number of variables n, constraint matrix density d, and bandwidth b_w, where each row i has exactly $k = d \cdot n$ ones. For constraint

i the k ones are chosen uniformly at random from variables $x_{i+1}, x_{i+2}, \ldots, x_{i+b_w}$ As an example, a constraint matrix with $n = 9, d = \frac{1}{3}$ and $b_w = 4$ may look like

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

As b_w grows, the underlying staircase-like structure of the instances dissolves. Hence, by increasing b_w, we are able to test the impact of the structure in the set covering instances on our MDD-based approach.

Consider the case when $b_w = k$. For such problems, as A is totally unimodular [26], the LP optimal solution will be integral, and so the corresponding IP will solve the problem at the root node. Similarly, we show here that the set of feasible solutions can be exactly represented by an MDD with width bounded by $m+1$. In particular, for any node u created during the top-down compilation method, I_u must be of the form $(0, 0, \ldots, 0, 1, 1, \ldots, 1)$. This is because, given any partial assignment fixing the top j variables, if some variable in constraint C_i is fixed to 1, then for any constraint C_k, with $k \leq i$, there must be some variable also fixed to 1. Hence, $\omega(M) \leq m + 1$. Therefore, such problems are also easily handled by MDD-based methods. Increasing the bandwidth, however, destroys the totally unimodular property of A and the bounded width of M. Therefore, increasing the bandwidth allows us to test how sensitive the LP and the relaxation MDDs are to changes in the structure of A.

3.6.3 Evaluating the MDD Parameters

In Section 3.3.2 we presented three possible heuristics for selecting nodes to merge. In preliminary computational tests, we found that using the heuristic based on shortest partial path lengths, H_2, seemed to provide the strongest MDD relaxations, and so we employ this heuristic.

The next parameter that must be fixed is the preset maximum width W that we allow for

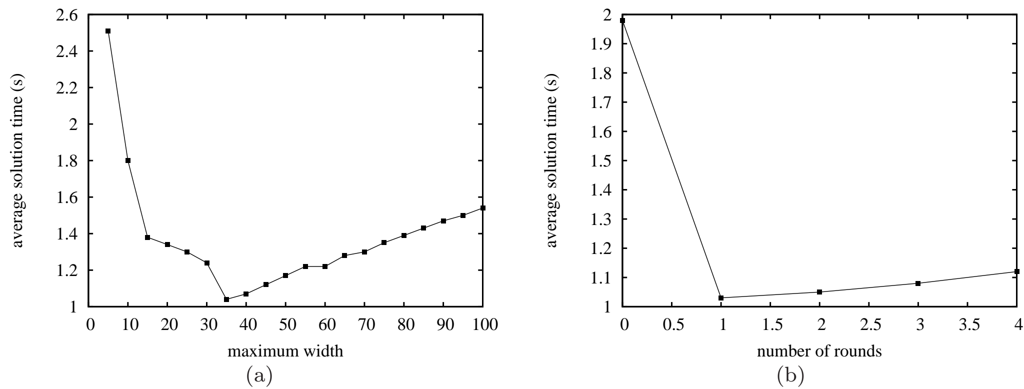


Figure 3.3: (a) Maximum width W vs. solution time, (b) Number of rounds of filtering vs. solution time.

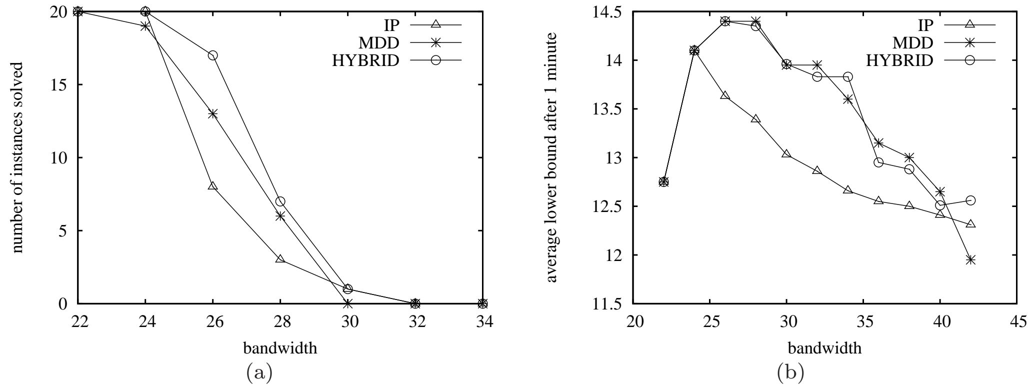
the MDD relaxations. Each problem (and even more broadly for each application of MDD relaxations to CSP/COPs) has a different optimal width. To test for an appropriate width for this class of problems, we generate 100 instances with $n = 100, k = 20$ and $b_w = 35$.

In Figure 3.3(a) we report the average solution time, over the 100 instances, for different maximum allowed widths W. Near $W = 35$ we see the fastest solution times, and hence for the remainder of the experimental testing we fix W at 35. We note here that during our preliminary computational tests, the range of widths that seemed to perform best was $W \in [20, 40]$.

Another parameter of interest is the number of times we allow the filtering algorithm to run before branching. As discussed in Section 3.5.3 the filtering algorithm presented above for set covering problems is not idempotent and applying the filtering once or for several rounds has different impacts on the solution time. In Figure 3.3(b) we report solution time versus the number of rounds of filtering averaged over the 100 instances with $W = 35$. Applying the filtering algorithm once yielded the fastest solution times and so we use this for the remainder of the experiments.

3.6.4 Evaluating the Impact of the Bandwidth

Next we compare the performance of our MDD-based approach with IP. We also compare the performance of these two methods with a hybrid MDD/IP approach. For the hybrid



(a) (b)

Figure 3.4: (a) Number of instances solved in 1 minute for different bandwidths, (b) Average lower bound in 1 minutes for different bandwidths.

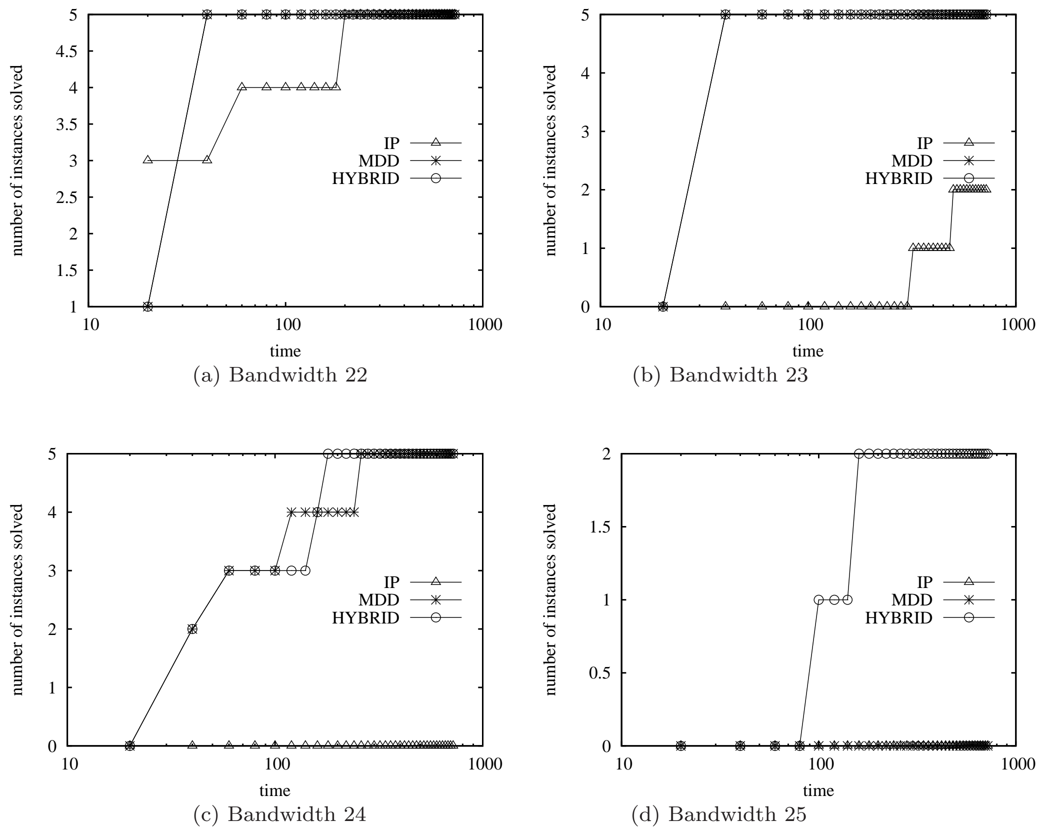
method, the MDD algorithm runs for a fixed amount of time and then passes the lower bound on the objective function to IP as a initial lower bound on the objective function.

We report results for random instances with $n = 250$, $k = 20$ and bandwidth $b_w \in \{22, 24, \dots, 44\}$ (20 instances per configuration). In Figure 3.4(a) we show, for increasing bandwidths, the number of instances solved in 60 seconds using the three proposed methods. For the hybrid method, we let the MDD method run for 10 seconds, and then pass the bound z_{LB} given by the MDD method to the IP and let the IP solver run for an additional 50 seconds. In addition, in Figure 3.4(b) we show, for increasing bandwidths, the best lower bound provided by the three methods after one minute.

For the lower bandwidths, we see that both the MDD-based approach and the hybrid approach outperform IP, with the hybrid method edging out the pure MDD method. As the bandwidth grows, however, the underlying structure that the MDD is able to exploit dissolves, but still the hybrid approach performs best.

3.6.5 Scaling Up

Here we present results on instances with 500 variables, and again with $k = 20$, to evaluate how the algorithms scale up. We have generated instances for various bandwidths b_w between 21 and 50 (5 random instances per configuration), and we report the most interesting results corresponding to the 'phase transition', i.e., $b_w \in \{22, 23, 24, 25\}$. We compare the



(a) Bandwidth 22

(b) Bandwidth 23

(c) Bandwidth 24

(d) Bandwidth 25

Figure 3.5: Performance profile for pure-IP, pure-MDD, and hybrid MDD/IP for instances width various bandwidth. Time is reported in log-scale.

three solution methods, allowing the algorithms to run for 12 minutes.

In the four plots given in Figure 3.5, we depict the performance profile of the three methods for the different bandwidths. We show for each bandwidth the number of instances solved by time t. As the bandwidth increases, we see that the IP is unable to solve many of the instances that the MDD-based method can, and for $b_w = 25$, neither the pure-IP nor the pure-MDD based methods can solve the instances, while the hybrid method was able to solve 2 of the 5 instances.

Figure 3.6(a) displays the lower bound given by the three approaches versus time, averaged over the 5 instances. We run the algorithms for 5 minutes and see that the lower bound given by the MDD-based approach dominates the IP bound, especially at small bandwidths.

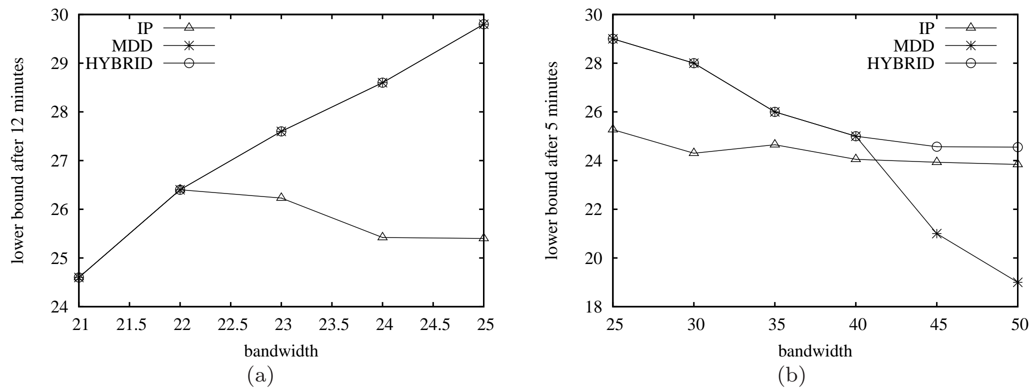


Figure 3.6: (a) Bandwidth versus lower bound (12 minute time limit), (b) Larger bandwidths versus lower bound (5 minute time limit).

However, as the bandwidth grows, as shown in Figure 3.6(b), the structure captured by the relaxation MDDs no longer exists and the pure-IP method is able to find better bounds. However, even at the larger bandwidths, the hybrid method provides the best bounds.

3.7 Conclusion

In conclusion, we have examined how relaxation MDDs can help in providing lower bounds for combinatorial optimization problems. We discuss methods for providing lower bounds via relaxation MDDs and provide computational results on applying these ideas to randomly generated set covering problems. We show that in general we can quickly improve upon LP bounds, and even outperform state-of-the-art integer programming technology on problem instances for which the bandwidth of the constraint matrix is limited. Finally, we have shown how a hybrid combination of IP and MDD-based relaxation can be even more effective.

Chapter 4

Restriction Decision Diagrams

4.1 Introduction

Binary optimization problems are ubiquitous across many problem domains. Over the last fifty years there have been significant advances in algorithms dedicated to solving problems in this class. In particular, general-purpose algorithms for binary optimization are commonly branch-and-bound methods that rely on two fundamental components: a relaxation of the problem, such as a linear programming relaxation of an integer programming model, and heuristics. Heuristics are used to provide feasible solutions during the search for an optimal one, which in practice is often more important than providing a proof of optimality.

Much of the research effort dedicated to developing heuristics for binary optimization has primarily focused on specific combinatorial optimization problems; this includes, e.g., the set covering problem [18] and the maximum clique problem [30, 61]. In contrast, general-purpose heuristics have received much less attention in the literature. The vast majority of the general techniques are embodied in integer programming technology, such as the *feasibility pump* [25] and the *pivot, cut, and dive* heuristic [23]. A survey of heuristics for integer programming is presented by [28, 29] and [13]. Local search methods for general binary problems can also be found in [1] and [14].

We introduce a new general-purpose method for obtaining a set of feasible solutions for binary optimization problems. Our method is based on an under-approximation of the

feasible solution set using binary decision diagrams (BDDs). BDDs are compact graphical representations of Boolean functions [2, 50, 16], originally introduced for applications in circuit design and formal verification [44, 50]. They have been recently used for a variety of purposes in combinatorial optimization, including post-optimality analysis [36, 37], cut generation in integer programming [6], and 0-1 vertex and facet enumeration [8]. The techniques presented here can also be readily applied to arbitrary discrete problems using *multivalued decision diagrams* (MDDs), a generalization of BDDs for discrete-valued functions.

Our method is a counterpart of the concept of *relaxed* MDDs, recently introduced by [3] as an over-approximation of the feasible set of a discrete constrained problem. The authors used relaxed MDDs for the purpose of replacing the typical domain store relaxation used in constraint programming by a richer data structure. They found that relaxed MDDs drastically reduce the size of the search tree and allow much faster solution of problems with multiple all-different constraints, which are equivalent to graph coloring problems. Analogous methods were applied to other types of constraints in [38] and [39].

Using similar techniques, [12] proposed the use of *relaxed BDDs* to derive relaxation bounds for binary optimization problem. The authors developed a general top-down construction method for relaxed BDDs and reported good results for structured set covering instances. Relaxed BDDs were also applied in the context of the maximum independent set problem, where the ordering of the variables in the BDD were shown to have a significant bearing on the effectiveness of the relaxation it provides [9].

We use BDDs to provide heuristic solutions, rather than relaxation bounds. Our main contributions include:

1. Introducing a new heuristic for binary optimization problems;

2. Discussing the necessary ingredients for applying the heuristic to specific classes of problems;

3. Providing an initial computational evaluation of the heuristic on the well-studied set covering and set packing problems. We show that, on a set of randomly generated instances, the solutions produced by our algorithm are comparable to those obtained with state-of-the-art integer programming optimization software (CPLEX).

The remainder of the chapter is organized as follows. We begin by defining BDDs in Section 4.2. In Section 4.3, we describe how to generate and use BDDs to exactly represent the set of feasible solutions to a problem. In Section 4.4, we describe how the algorithm in Section 4.3 can be modified to provide an under-approximation of the feasible set and to deliver a set of solutions to a problem. We discuss the application of the algorithm to two problem classes in Section 4.5, and present computational experiments in Section 4.6.

4.2 Binary Decision Diagrams

Binary optimization problems (BOPs) are specified by a set of binary variables $X = \{x_1, \ldots, x_n\}$, an objective function $f : \{0,1\}^n \to \mathbb{R}$ to be minimized, and a set of m constraints $C = \{C_1, \ldots, C_m\}$, which define relations among the problem variables. A *solution* to a BOP P is an assignment of values 0 or 1 to each of the variables in X. A solution is *feasible* if it satisfies all the constraints in C. The set of feasible solutions of P is denoted by $\mathrm{Sol}(P)$. A solution x^* is *optimal* for P if it is feasible and satisfies $f(x^*) \leq f(\tilde{x})$ for all $\tilde{x} \in \mathrm{Sol}(P)$.

A *binary decision diagram* (BDD) $B = (U, A)$ for a BOP P is a layered directed acyclic multi-graph that encodes a set of solutions of P. The nodes U are partitioned into $n+1$ layers, $L_1, L_2, \ldots, L_{n+1}$, where we let $\ell(u)$ be the layer index of node u. Layers L_1 and L_{n+1} consist of single nodes; the root r and the terminal t, respectively. The *width* of layer j is given by $\omega_j = |L_j|$, and the *width* of B is $\omega(B) = \max_{j \in \{1,2,\ldots,n\}} \omega_j$. The *size* of B, denoted by $|B|$, is the number of nodes in B.

Each arc $a \in A$ is directed from a node in some layer j to a node in the adjacent layer $j+1$, and has an associated *arc-domain* $d_a \in \{0,1\}$. The arc a is called a *1-arc* when $d_a = 1$ and a *0-arc* when $d_a = 0$. For any two arcs a, a' directed out of a node u, $d_a \neq d_{a'}$, so that the maximum out-degree of a node in a BDD is 2, with each arc having a unique arc-domain. Given a node u, we let $a_0(u)$ be the 0-arc directed out of u (if it exists) and $b_0(u)$ be the node in $L_{\ell(u)+1}$ at its opposite end, and similarly for $a_1(u)$ and $b_1(u)$.

A BDD B represents a set of solutions to P in the following way. An arc a directed out of a node u represents the assignment $x_{\ell(u)} = d_a$. Hence, for two nodes u, u' with $\ell(u) < \ell(u')$, a directed path p from u to u' along arcs $a_{\ell(u)}, a_{\ell(u)+1}, \ldots, a_{\ell(u')-1}$ corresponds

to the assignment $x_j = d_{a_j}$, $j = \ell(u), \ell(u)+1, \ldots, \ell(u')-1$. In particular, an r–t path $p = (a_1, \ldots, a_n)$ corresponds to a solution x^p, where $x^p_j = d_{a_j}$ for $j = 1, \ldots, n$. The set of solutions represented by a BDD B is denoted by $\mathrm{Sol}(B) = \{x^p \,|\, p \text{ is an } r\text{–}t \text{ path}\}$. An *exact* BDD B for P is any BDD for which $\mathrm{Sol}(B) = \mathrm{Sol}(P)$.

For two nodes $u, u' \in U$ with $\ell(u) < \ell(u')$, let $B_{u,u'}$ be the BDD induced by the nodes that belong to some directed path between u and u'. In particular, $B_{r,t} = B$. A BDD is called *reduced* if $\mathrm{Sol}(B_{u,u'})$ is unique for any two nodes u, u' of B. The reduced BDD B is unique when the variable ordering is fixed, and therefore the most compact representation in terms of size for that ordering [68].

Finally, for a large class of objective functions (e.g., for additively separable functions), optimizing over the solutions represented by a BDD B can be reduced to finding a shortest path in B. For example, given a real cost vector c and a linear objective function $c^T x$, we can associate an *arc-cost* $c(u,v) = c_{\ell(u)} d_{u,v}$ with each arc $a = (u,v)$ in the BDD. This way, a shortest r–t path corresponds to a minimum cost solution in $\mathrm{Sol}(B)$. If B is exact, then this shortest path corresponds to an optimal solution for P.

Example 3 Consider the following BOP P.

$$\begin{aligned} \text{minimize} \quad & -2x_1 - 3x_2 - 5x_3 - x_4 - 3x_5 \\ \text{subject to} \quad & 2x_1 + 2x_2 + 3x_3 + 3x_4 + 2x_5 \le 5 \\ & x_j \in \{0,1\}, \quad j = 1, \ldots, 5 \end{aligned}$$

In Figure 4.1 we show an exact reduced BDD for P. The 0-arcs are represented by dashed lines, while the 1-arcs are represented by solid lines. There are 13 paths in the BDD, which correspond to the 13 feasible solutions of this BOP. Assigning arc costs of 0 to all of the 0-arcs and the cost coefficient of x_j to the 1-arcs on layer j, $j = 1, \ldots, 5$, the two shortest paths in the BDD correspond to the solutions $(0,1,1,0,0)$ and $(0,0,1,0,1)$, both optimal solutions for P. $\square$

x_1
x_2
x_3
x_4
x_5

Figure 4.1: Reduced BDD for the BOP in Example 3.

4.3 Exact BDDs

An exact reduced BDD $B = (U, A)$ for a BOP P can be interpreted as a compact search tree for P, where infeasible leaf nodes are removed, isomorphic subtrees are superimposed, and the feasible leaf nodes are merged into t. In principle, B can be obtained by first constructing the branching tree for P and reducing it accordingly, which is impractical for our purposes.

We present here an efficient top-down algorithm for constructing an exact BDD B for P. It relies on problem-dependent information for merging BDD nodes and thus reducing its size. If this information satisfies certain conditions, the resulting BDD is reduced. The algorithm is a *top-down* procedure since it proceeds by compiling the layers of B one-by-one, where layer L_{j+1} is constructed only after layers $L_1, \ldots, L_j$ are completed.

We first introduce some additional definitions. Let $x' = (x'_1, \ldots, x'_j)$, $j < n$, be a *partial solution* that assigns a value to variables $x_1, \ldots, x_j$. We define

$$F(x') = \{x'' \in \{0,1\}^{n-j} \mid x = (x', x'') \text{ is feasible for } P\}$$

as the set of *feasible completions* of x'. We say that two distinct partial solutions x^1, x^2 on variables $x_1, \ldots, x_j$ are *equivalent* if $F(x^1) = F(x^2)$.

The algorithm requires a method for establishing when two partial solutions are necessarily equivalent. If this is possible, then the last nodes u, u' of the BDD paths corresponding to these partial solutions can be merged into a single node, since $B_{u,t}$ and $B_{u',t}$ are the same. To this end, with each partial solution x' of dimension k we associate a *state function* $\mathtt{s} : \{0,1\}^k \to S$, where S is a problem-dependent *state space*. The state of x' corresponds to the information necessary to determine if x' is equivalent to any other partial solution on the same set of variables.

Formally, let x^1, x^2 be partial solutions on the same set of variables. We say that the function $\mathtt{s}(x)$ is *sound* if $\mathtt{s}(x^1) = \mathtt{s}(x^2)$ implies that $F(x^1) = F(x^2)$, and we say that $\mathtt{s}$ is *complete* if the converse is also true. The algorithm requires only a sound state function, but if $\mathtt{s}$ is complete, the resulting BDD will be reduced.

For simplicity of exposition, we further assume that it is possible to identify when a partial solution x' cannot be completed to a feasible solution, i.e. $F(x') = \emptyset$. It can be shown that this assumption is not restrictive, but rather makes for an easier exposition of the algorithm. We write $\mathtt{s}(x') = \hat{0}$ to indicate that x' is not able to be completed to a feasible solution. If x is a solution to P, we write $\mathtt{s}(x) = \emptyset$ if x is feasible and $\mathtt{s}(x) = \hat{0}$ otherwise.

We now extend the definition of state functions to nodes of the BDD B. Suppose that $\mathtt{s}$ is a complete state function and B is an exact (but not necessarily reduced) BDD. For any node u, the fact that B is exact implies that any two partial solutions $x^1, x^2 \in \text{Sol}(B_{r,u})$ have the same feasible completions, i.e. $F(x^1) = F(x^2)$. Since $\mathtt{s}$ is complete, we must have $\mathtt{s}(x^1) = \mathtt{s}(x^2)$. We henceforth define the state of a node u as $\mathtt{s}(u) = \mathtt{s}(x)$ for any $x \in \text{Sol}(B_{r,u})$, which is therefore uniquely defined for a complete function $\mathtt{s}$.

We also introduce a function $\mathtt{update} : S \times \{0,1\} \to S$. Given a partial solution x' on variables $x_1, \ldots, x_j$, $j < n$, and a domain value $d \in \{0,1\}$, the function $\mathtt{update}(\mathtt{s}(x'), d)$ maps the state of x' to the state of the partial solution obtained when x' is appended with d, $\mathtt{s}((x', d))$. This function is similarly extended to nodes: $\mathtt{update}(s(u), d)$ represents the state of all partial solutions in $\text{Sol}(B_{r,u})$ extended with value d for a node u.

The top-down compilation procedure is presented in Algorithm 6. We start by setting $L_1 = \{r\}$ and $\mathtt{s}(r) = s_0$, where s_0 is an initial state appropriately defined for the problem.

Algorithm 6 Exact BDD Compilation

1: Create node r with $\mathtt{s}(r) = s_0$
2: $L_1 = \{r\}$
3: **for** $j = 1$ to n **do**
4: $L_{j+1} = \emptyset$
5: **for all** $u \in L_j$ **do**
6: **for all** $d \in \{0, 1\}$ **do**
7: $\mathtt{s}_{\text{new}} := \mathtt{update}(\mathtt{s}(u), d)$
8: **if** $\mathtt{s}_{\text{new}} \neq \hat{0}$ **then**
9: **if** $\exists u' \in L_{j+1}$ with $\mathtt{s}(u') = \mathtt{s}_{\text{new}}$ **then**
10: $b_d(u) = u'$
11: **else**
12: Create node u_{new} with $\mathtt{s}(u_{\text{new}}) = \mathtt{s}_{\text{new}}$
13: $b_d(u) = u_{\text{new}}$
14: $L_{j+1} \leftarrow L_{j+1} \cup u_{\text{new}}$

Now, having constructed layers $L_1, \ldots, L_j$, we create layer L_{j+1} in the following way. For each node $u \in L_j$ and for $d \in \{0, 1\}$, let $\mathtt{s}_{\text{new}} = \mathtt{update}(\mathtt{s}(u), d)$. If $\mathtt{s}_{\text{new}} = \hat{0}$ we do not create arc $a_d(u)$. Otherwise, if there exists some $u' \in L_{j+1}$ with $\mathtt{s}(u') = \mathtt{s}_{\text{new}}$, we set $b_d(u) = u'$; if such a node does not exist, we create node u_{new} with $\mathtt{s}(u_{\text{new}}) = \mathtt{s}_{\text{new}}$ and set $b_d(u) = u_{\text{new}}$.

Example 4 Consider the following simple binary optimization problem:

$$\begin{aligned} \text{maximize} \quad & 5x_1 + 4x_2 + 3x_3 \\ \text{subject to} \quad & x_1 + x_2 + x_3 \leq 1 \\ & x_j \in \{0, 1\}, \quad j = 1, 2, 3 \end{aligned}$$

We can define $\mathtt{s}(x)$ to equal the number of variables set to 1 in x. In this way, whenever $\mathtt{s}(x^1) = \mathtt{s}(x^2)$ for two partial solutions we have $F(x^1) = F(x^2)$. For example, $\mathtt{s}\left((1, 0)\right) = 1$ and $\mathtt{s}\left((0, 1)\right) = 1$, with the only feasible completion being (0).

In addition, we can let

$$\texttt{update}(\texttt{s}(u), d) = \begin{cases} \hat{0} & , d = 1 \text{ and } \texttt{s}(u) = 1 \\ 1 & , d = 1 \text{ and } \texttt{s}(u) = 0 \\ s(u) & , d = 0 \end{cases}$$

With this update function, if in a partial solution there is already 1 variable set to 1, the update operation will assign $\hat{0}$ to the node on the 1-arc (signifying that the solution cannot be completed to a feasible solution) and 1 to the node on the 1-arc (to signify that still only one variable is set to 1). On the other hand, if a partial solution has no variable set to 1, the 1-arc will now be directed to a node that has state 1 and the 0-arc will be directed to a node with state 0. □

Theorem 6 *Let* $\texttt{s}$ *be a sound state function for a binary optimization problem* P*. Algorithm 6 generates an exact BDD for* P*.*

Proof. We show by induction that at the end of iteration j, the set

$$\bigcup_{u \in L_{j+1}} \text{Sol}(B_{r,u})$$

exactly corresponds to the set of feasible partial solutions of P on $x_1, \ldots, x_j$. This implies that after iteration n, $\text{Sol}(B_{r,t}) = \text{Sol}(P)$, since all feasible solutions x have the same state $\texttt{s}(x) = \emptyset$ and hence L_{n+1} will contain exactly one node at the end of the procedure, which is the terminal t.

Consider the first iteration. We start with the root r and $\texttt{s}(r) = s_0$, which is the initial state corresponding to not assigning any values to any variables. r is the only node in L_1. When $d = 0$, if there exists no feasible solution with $x_1 = 0$, no new node is created, and hence no solutions are introduced into B. If otherwise there exists at least one solution with $x_1 = 0$, we create a new node, add it to L_2, and introduce a 0-arc from r to the newly created node. This will represent the partial solution $x_1 = 0$. This is similarly done for $d = 1$.

Now, consider the end of iteration j. Each solution $x' = (x'', d)$ that belongs to $\text{Sol}(B_{r,u})$ for some node $u \in L_{j+1}$ must go through some node $u' \in L_j$ with $b_d(u') = u$. By induction,

x'' is a feasible partial solution with $\mathtt{s}(u') = \mathtt{s}(x'') \neq \hat{0}$. But when the arc $a_d(u')$ is considered, we must have $\mathtt{update}(u', d) \neq \hat{0}$, for otherwise this arc would not have been created. Therefore, each solution in $\mathrm{Sol}(B_{r,u})$ is feasible. Since $u \in L_{j+1}$ was chosen arbitrarily, only feasible partial solutions exists in $\mathrm{Sol}(B_{r,u})$ for all nodes $u \in L_{j+1}$.

What remains to be shown is that all feasible partial solutions exist in $\mathrm{Sol}(B_{r,u})$ for some $u \in L_{j+1}$. This is trivially true for the partial solutions $x_1 = 0$ and $x_1 = 1$. Take now any partial feasible solution $x' = (x'', d)$ on the first j variables, $j \geq 2$. Since x' is a partial feasible solution, x'' must also be a partial feasible solution. By induction, x'' belongs to $\mathrm{Sol}(B_{r,u})$, for some $u \in L_j$. When Algorithm 6 examines node u, $\mathtt{update}(\mathtt{s}(u), d)$ must not return $\hat{0}$ because $F(x') \neq \emptyset$. Therefore, the d-arc directed out of u is created, ending at some node $b_d(u) \in L_{j+1}$, as desired. □

Theorem 7 *Let* $\mathtt{s}$ *be a complete state function for a binary optimization program* P*. Algorithm 6 generates an exact reduced BDD for* P*.*

Proof. By Theorem 6, B is exact. Moreover, for each j, each node $u \in L_j$ will have a unique state because of line 9. Therefore, any two partial solutions x', x'' ending at unique nodes $u', u'' \in L_j$ will have $F(x') \neq F(x'')$.□

Theorem 8 *Let* $B = (U, A)$ *be the exact BDD outputted by Algorithm 6 for a BOP* P *with a sound state function* $\mathtt{s}$*. Algorithm 6 runs in time* $O(|U|K)$*, where* K *is the time complexity for each call of the* `update` *function.*

Proof. Algorithm 6 performs two calls of `update` for every node u added to B. Namely, one call to verify if u has a d-arc for each domain value $d \in \{0, 1\}$. □

Theorem 8 implies that, if `update` can be implemented efficiently, then Algorithm 6 runs in polynomial time in the size of the exact BDD B. Indeed, there are structured problems for which one can define complete state functions with a polynomial time-complexity for `update` [3, 12, 9]. This will be further discussed in Section 4.5.

4.4 Restricted BDDs

Constructing exact BDDs for general binary programs using Algorithm 6 presents two main difficulties. First, the `update` function may take time exponential in the input of the problem. This can be circumvented by not requiring a complete state function, but rather just a sound state function. The resulting BDD is exact according to Theorem 6, but perhaps not reduced. This poses only a minor difficulty, as there exist algorithms for reducing a BDD B that have a polynomial worst-case complexity in the size of B [68]. A more confining difficulty, however, is that even an exact reduced BDD may be exponentially large in the size of the BOP P. We introduce the concept of *restricted BDDs* as a remedy for this problem. These structures provide an under-approximation (i.e. a subset) of the set of feasible solutions to a problem P. Such BDDs can therefore be used as a generic heuristic procedure for any BOP.

More formally, let P be a BOP. A BDD B is called a restricted BDD for P if $\text{Sol}(B) \subseteq \text{Sol}(P)$. Analogous to exact BDDs, optimizing additively separable objective functions over $\text{Sol}(B)$ reduces to a shortest path computation on B if the arc weights are assigned appropriately. Thus, once a restricted BDD is generated, we can readily extract the best feasible solution from B and provide an upper bound to P.

We will focus on *limited-width* restricted BDDs, in which we limit the size of the BDD B by requiring that $\omega(B) \leq W$ for some pre-set maximum allotted width W.

Example 5 Consider the BOP from Example 3. Figure 4.2 shows a width-2 restricted BDD. There are eight paths in the BDD which correspond to eight feasible solutions. Assigning arc costs as in Example 3, a shortest path from the root to the terminal corresponds to the solution $(0, 1, 0, 0, 1)$ with an objective function value of -6 (the optimal value is -8). □

Limited-width restricted BDDs can be easily generated by performing a simple modification to Algorithm 6. Namely, we insert the procedure described in Algorithm 7 immediately after line 3 of Algorithm 6. This procedure is described as follows. We first verify whether $\omega_j = |L_j| > W$. If so, we delete a set of $|L_j| - W$ nodes in the current layer, which is chosen by a function $\texttt{node_select}(L_j)$. We then continue building the BDD as in Algorithm 6.

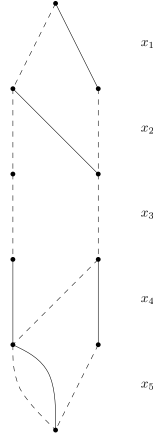


Figure 4.2: Width-2 restricted BDD for the BOP presented in Example 3.

Algorithm 7 `delete_nodes`

Insert immediately after line 3 of Algorithm 6.

1: **if** $\omega_j = |L_j| > W$ **then**

2: $\quad M :=$ `node_select`(L_j) // *where* $|M| = \omega_j - W$

3: $\quad L_j \leftarrow L_j \setminus M$

It is clear that the modified algorithm produces a BDD B satisfying $\omega(B) \leq W$. In addition, it must create a restricted BDD since we are never changing the states of the nodes during the construction, but rather just deleting nodes. Since Algorithm 6 produces an exact BDD, this modified algorithm must produce a restricted BDD.

Theorem 9 describes how the time complexity of Algorithm 6 is affected by the choice of the maximum allotted width W.

Theorem 9 *The modified version of Algorithm 6 for width-W restricted BDDs has a worst-case time complexity of $O(nL + nWK)$, where L and K are the time complexity for each call of the* `node_select` *and* `update` *functions, respectively.*

Proof. Because the function `node_select` is called once per layer, it contributes to $O(nL)$ to the overall time complexity. The `update` function is called twice for each BDD node. Since there will be at most $O(nW)$ nodes in a width-W restricted BDD, the theorem follows. □

The selection of nodes in `node_select`(L_j) can have a dramatic impact on the quality of

the solutions encoded by the restricted BDD. In fact, as long as we never delete the nodes $u_1, \dots, u_n$ that are traversed by some optimal solution x^*, we are sure to have the optimal solution in the final BDD.

We observed that the following `node_select` procedure yields restricted BDDs with the best quality solutions in our computational experiments. Each node $u \in L_j$ is first assigned a value $lp(u) = \min f(x) \in \text{Sol}(B_{r,u})$, where f is the objective function of P. (We are assuming a minimization problem; a maximization problem can be handled in an analogous way.) This can be easily computed for a number of objective functions by means of a dynamic programming algorithm; for example linear cost functions whose arc weights are as described in Section 4.2. The `node_select`(L_j) function then deletes the nodes in L_j with the largest $lp(u)$ values. We henceforth use this heuristic for `node_select` in the computational experiments of Section 4.6. It can be shown that the worst-case complexity of this particular heuristic is $O(W \log W)$.

4.5 Applications

We now describe the application of restricted BDDs to two fundamental problems in binary optimization: the set covering problem and the set packing problem. For both applications, we describe the problem and provide a sound state function. We then present the `update` operation based on this state function which can be used by the modified version of Algorithm 6.

4.5.1 Set Covering Problem

The (weighted) *set covering problem* (SCP) is the binary program

$$\begin{aligned} \text{minimize } & c^T x \\ \text{subject to } & Ax \geq e \\ & x_j \in \{0,1\}, \;\; j = 1, \dots, n \end{aligned}$$

where c is an n-dimensional real-valued vector, A is a 0–1 $m \times n$ matrix, and e is the m-dimensional unit vector. Let $a_{i,j}$ be the element in the i-th row and j-th column of A,

and define $A_j = \{i \mid a_{i,j} = 1\}$ for $j = 1, \ldots, n$. The SCP asks for a minimum-cost subset $V \subseteq \{1, \ldots, n\}$ of the sets A_j such that for all $i, a_{i,j} = 1$ for some $j \in V$, i.e. V *covers* $\{1, \ldots, m\}$.

State Function

We now present a sound state function for the purpose of generating restricted BDDs by means of Algorithm 6. Let C_i be the set of indices of the variables that participate in constraint i, $C_i = \{j \mid a_{i,j} = 1\}$, and let $\texttt{last}(C_i) = \max\{j \mid j \in C_i\}$ be the largest index of C_i. We consider the state space $S = 2^{\{1,\ldots,m\}} \cup \{\hat{0}\}$. For a partial solution x' on variables $x_1, \ldots, x_j$, we write the state function

$$\texttt{s}(x') = \begin{cases} \hat{0}, & \text{if } \exists\, i : \sum_{k=1}^{j} a_{i,k} x'_k = 0 \text{ and } j \geq \texttt{last}(C_i), \\ \\ \left\{ i : \sum_{k=1}^{j} a_{i,k} x'_k = 0 \right\}, & \text{otherwise.} \end{cases}$$

We first argue that the function above assigns a state $\hat{0}$ to a partial solution x' if and only if $F(x') = \emptyset$. Indeed, the condition $\sum_{k=1}^{j} a_{i,k} x'_k = 0$, $j \geq \texttt{last}(C_i)$ for some i implies that all variables that relate to the i-th constraint $\sum_{k=1}^{n} a_{i,j} x_j \geq 1$ are already zero in x', and hence the constraint can never be satisfied. If otherwise that condition does not hold, then $(1, \ldots, 1)$ is a feasible completion of x'.

In addition, the following Lemma shows that $\texttt{s}$ is a sound state function for the SCP.

Lemma 6 *Let x^1, x^2 be two partial solutions on variables $x_1, \ldots, x_j$. Then, $\texttt{s}(x^1) = \texttt{s}(x^2)$ implies that $F(x^1) = F(x^2)$.*

Proof. Let x^1, x^2 be two partial solutions with dimension j for which $\texttt{s}(x^1) = \texttt{s}(x^2) = s'$. If $s' = \hat{0}$ then both have no feasible completions, so it suffices to consider the case when $s' \neq \hat{0}$. Take any completion $\tilde{x} \in F(x^1)$. We show that $\tilde{x} \in F(x^2)$.

Suppose, for the purpose of contradiction, that $(x^2, \tilde{x})$ violates the i^*-th SCP inequality,

$$\sum_{k=1}^{j} a_{i^*,k}\, x^2_k + \sum_{k=j+1}^{n} a_{i^*,k}\, \tilde{x}_k = 0, \tag{4.1}$$

while

$$\sum_{k=1}^{j} a_{i^*,k}\, x^1_k + \sum_{k=j+1}^{n} a_{i^*,k}\, \tilde{x}_k \geq 1 \tag{4.2}$$

since $(x^1, \tilde{x})$ is feasible.

By (4.1), we have that

$$\sum_{k=j+1}^{n} a_{i^*,k}\, \tilde{x}_k = 0 \tag{4.3}$$

and

$$\sum_{k=1}^{j} a_{i^*,k}\, x_k^2 = 0. \tag{4.4}$$

The equality (4.4) implies that $i^* \in \mathtt{s}(x^2)$ and therefore $i^* \in \mathtt{s}(x^1)$. But then $\sum_{k=1}^{j} a_{i^*,k} x_k^1 = 0$. This, together with (4.3), contradicts (4.2). $\square$

Assuming a partial solution x' on variables $x_1, \ldots, x_j$ and that $s(x') \neq \hat{0}$, the corresponding `update` operation is given by

$$\mathtt{update}(\mathtt{s}(x'), d) = \begin{cases} \mathtt{s}(x') \setminus \{i \mid a_{i,j+1} = 1\}, & d = 1 \\ \mathtt{s}(x'), & d = 0, \forall\, i^* \in \mathtt{s}(x') : \mathtt{last}(C_{i^*}) > j+1 \\ \hat{0}, & d = 0, \exists\, i^* \in \mathtt{s}(x') : \mathtt{last}(C_{i^*}) = j+1 \end{cases}$$

and has a worst-case time complexity of $O(m)$ for each call.

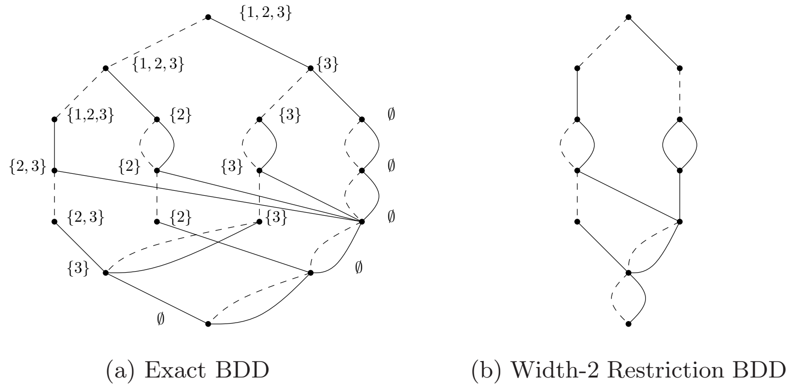
Example 6 Consider the SCP instance with

$$c = (2, 1, 4, 3, 4, 3)$$

and

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Figure 4.3(a) shows an exact reduced BDD for this SCP instance where the nodes are labeled with their corresponding states. If outgoing 1-arcs (0-arcs) of nodes in layer j are assigned a cost of c_j (zero), a shortest r–t path corresponds to solution $(1, 1, 0, 0, 0, 0)$ and proves an optimal value of 3. Figure (b) depicts a width-2 restricted BDD where a shortest r–t path corresponds to solution $(0, 1, 0, 1, 0, 0)$, which proves an upper bound of 4. $\square$



(a) Exact BDD (b) Width-2 Restriction BDD

Figure 4.3: Exact and Restricted BDD for the Set Covering Problem

Example 7 The implication in Lemma 6 is not sufficient as the state function is not complete. Consider the set covering problem

$$\begin{aligned} \text{minimize} \quad & x_1 + x_2 + x_3 \\ \text{subject to} \quad & x_1 + x_3 \geq 1 \\ & x_2 + x_3 \geq 1 \\ & x_1, x_2, x_3 \in \{0, 1\} \end{aligned}$$

and the two partial solutions $x^1 = (1, 0), x^2 = (0, 1)$. We have $\mathtt{s}(x^1) = \{2\}$ and $\mathtt{s}(x^2) = \{1\}$. However, both have the single feasible completion $\tilde{x} = (1)$. $\square$

There are several ways to modify the state function to turn it into a complete one [12]. The state function can be strengthened to a complete state function, which requires only polynomial time to compute per partial solution, but nonetheless at an additional computational cost. For the computational experiments in Section 4.6 we report results for the simpler (sound) state function presented above.

4.5.2 The Set Packing Problem

A problem closely related to the SCP, the (weighted) *set packing problem* (SPP), is the binary program

$$\begin{aligned} \text{maximize } & c^T x \\ \text{subject to } & Ax \leq e \\ & x_j \in \{0,1\}, \quad j = 1, \ldots, n \end{aligned}$$

where c, A, and e are as in the SCP. Letting A_j be as in Section 4.5.1, the SPP asks for the maximum-cost subset $V \subseteq \{1, \ldots, n\}$ of the sets A_j such that for all $i, a_{i,j} = 1$ for at most one $j \in V$.

State Function

For the SPP, the state function identifies the set of constraints for which no variables have been assigned a one and could still be violated. More formally, consider the state space $S = 2^{\{1,\ldots,m\}} \cup \{\hat{0}\}$. For a partial solution x' on variables $x_1, \ldots, x_j$, we write the state function

$$\mathtt{s}(x') = \begin{cases} \hat{0}, & \text{if } \exists\, i : \sum_{k=1}^{j} a_{i,k} x'_k > 1 \\ \left\{ i : \sum_{k=1}^{j} a_{i,k} x'_k = 0 \text{ and } \mathtt{last}(C_i) > j \right\}, & \text{otherwise.} \end{cases}$$

We first argue that the function above assigns a state $\hat{0}$ to a partial solution x' if and only if $F(x') = \emptyset$. Indeed, the condition $\sum_{k=1}^{j} a_{i,k} x'_k > 1$ for some i immediately implies that x' is infeasible; otherwise, $(0, \ldots, 0)$ is at least one feasible completion to x'.

As the following lemma shows, if the states of two partial solutions on the same set of variables are the same, then the set of feasible completions for these partial solutions are the same, thus proving that this state function is sound.

Lemma 7 *Let x^1, x^2 be two partial solutions on variables $x_1, \ldots, x_j$. Then, $\mathtt{s}(x^1) = \mathtt{s}(x^2)$ implies that $F(x^1) = F(x^2)$.*

Proof. Let x^1, x^2 be two partial solutions for which $\mathtt{s}(x^1) = \mathtt{s}(x^2) = s'$. If $s' = \hat{0}$ then both have empty sets of feasible completions, so it suffices to consider the case when $s' \neq \emptyset$. Take any partial solution $\tilde{x} \in F(x^1)$. We show that $\tilde{x} \in F(x^2)$.

Suppose, for the purpose of contradiction, that $(x^2, \tilde{x})$ violates the i^*-th SPP inequality,

$$\sum_{k=1}^{j} a_{i^*,k}\, x_k^2 + \sum_{k=j+1}^{n} a_{i^*,k}\, \tilde{x}_k > 1, \tag{4.5}$$

while

$$\sum_{k=1}^{j} a_{i^*,k}\, x_k^1 + \sum_{k=j+1}^{n} a_{i^*,k}\, \tilde{x}_k \leq 1, \tag{4.6}$$

since $(x^1, \tilde{x})$ is feasible.

First suppose that $\sum_{k=j+1}^{n} a_{i^*,k}\, \tilde{x}_k = 1$. By (4.6), $\sum_{k=1}^{j} a_{i^*,k}\, x_k^1 = 0$. This implies that $F(x^1)$ contains i^* since no variables in C_{i^*} are set to 1 and there exists $\ell \in C_{i^*}$ with $\ell > j$. Therefore $F(x^2)$ also contains i^*, implying that no variable in C_{i^*} is set to one in the partial solution x^2. Hence $\sum_{k=1}^{j} a_{i^*,k}\, x_k^2 = 0$, contradicting (4.5).

Now suppose that $\sum_{k=j+1}^{n} a_{i^*,k}\, \tilde{x}_k = 0$. Then $\sum_{k=1}^{j} a_{i^*,k}\, x_k^2 > 1$, contradicting the assumption that $s' = \mathtt{s}(x^2) \neq \emptyset$. □

Given a partial solution x' on variables $x_1, \ldots, x_j$ with $\mathtt{s}(x') \neq \hat{0}$, the corresponding `update` operation is

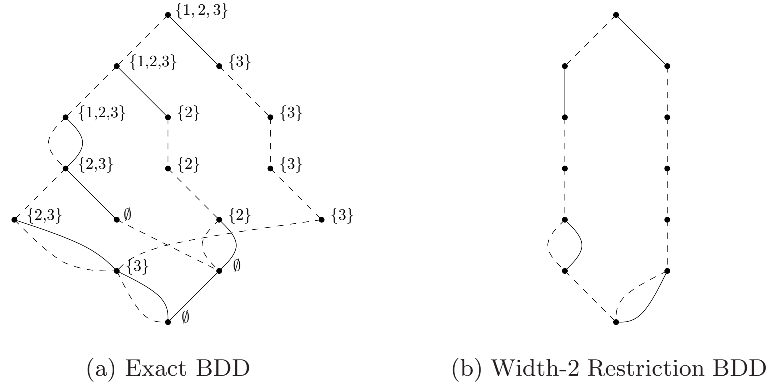
$$\mathtt{update}(s(x'), d) = \begin{cases} \mathtt{s}(x') \setminus \{i \mid \mathtt{last}(C_i) = j+1\}, & d = 0 \\ \mathtt{s}(x') \setminus \{i \mid j+1 \in C_i\}, & d = 1, A_{j+1} \subseteq \mathtt{s}(x') \\ \hat{0}, & d = 1, A_{j+1} \not\subseteq \mathtt{s}(x') \end{cases}$$

and has a worst-case time complexity of $O(m)$ for each call.

Example 8 Consider the SPP instance with the same constraint matrix A as in the Example 6, but with weight vector

$$c = (1, 1, 1, 1, 1, 1).$$

Figure 4.4a shows an exact reduced BDD for this SPP instance. The nodes are labeled with their corresponding states. Assigning arc costs 1/0 to each 1/0-arc, a longest r–t path (which can be computed by a shortest path on arc weights $c' = -c$, since the BDD is acyclic)



(a) Exact BDD (b) Width-2 Restriction BDD

Figure 4.4: Exact and Restricted BDD for the Set Packing Problem

corresponds to solution $(0, 0, 1, 0, 1, 1)$ and proves an optimal value of 3. Figure 4.4b depicts a width-2 restricted BDD where a longest r–t path, for example, corresponds to solution $(1, 0, 0, 0, 0, 1)$, which has length 2. □

Example 9 As in the case of the SCP, the above state function is not complete. For example, consider the problem

$$\begin{aligned} \text{maximize} \quad & x_1 + x_2 + x_3 \\ \text{subject to} \quad & x_1 + x_3 \leq 1 \\ & x_2 + x_3 \leq 1 \\ & x_1, x_2, x_3 \in \{0, 1\} \end{aligned}$$

and the two partial solutions $x^1 = (1, 0), x^2 = (0, 1)$. We have the states $\mathtt{s}(x^1) = \{2\}$ and $\mathtt{s}(x^2) = \{1\}$. However, both have the single feasible completion, $\tilde{x} = (0)$. □

There are several ways to modify the state function above to turn it into a complete one. For example, one can reduce the SPP to an independent set problem and apply the state function defined in [9]. We only consider the sound state function in this work.

4.6 Computational Experiments

In this section, we perform a computational study on randomly generated set covering and set packing instances. We evaluate our method comparing the bounds provided by a restricted BDD with the ones obtained via state-of-the-art integer programming technology (IP). We acknowledge that a procedure solely geared toward constructing heuristic solutions for BOPs is in principle favored against general-purpose IP solvers. Nonetheless, we sustain that this is still a meaningful comparison, as modern IP solvers are the best-known general bounding technique for 0-1 problems due to their advanced features and overall performance, specially for set packing and set covering problems. This method of testing new heuristics for binary optimization problems was employed by the authors in [14] and we provide a similar study here to evaluate the effectiveness of the algorithm.

The tests ran on an Intel Xeon E5345 with 8 GB of RAM. The BDD code was implemented in C++. We used Ilog CPLEX 12.4 as our IP solver. In particular, we took the bound obtained from the root node relaxation. We set the solver parameters in a way to balance the quality of the bound value and the CPU time to process the root node. The CPLEX parameters that are distinct from the default settings are presented in Table 4.1. We note that all cuts were disabled, since we observed that the root node would be processed orders of magnitude faster without such cuts, which did not have a significant effect on the quality of the heuristic solution obtained for the instances tested.

Table 4.1: CPLEX Parameters

Parameters (CPLEX internal name)	*Value*
Version	12.4
Number of explored nodes (`NodeLim`)	0 (only root)
Parallel processes (`Threads`)	1
Cuts (`Cuts`, `Covers`, `DisjCuts`, ...)	-1 (off)
Emphasis (`MIPEmphasis`)	4 (find hidden feasible solutions)
Time limit (`TiLim`)	3600

Our experiments focus on instances with a particular structure. Namely, we provide evidence that restricted BDDs perform well when the constraint matrix has a small *bandwidth*.

The bandwidth of a matrix A is defined as

$$b_w(A) = \max_{i\in\{1,2,\ldots,m\}}\{\max_{j,k:a_{i,j},a_{i,k}=1}\{j-k\}\}.$$

The bandwidth represents the largest distance, in the variable ordering given by the constraint matrix, between any two variables that share a constraint. The smaller the bandwidth, the more structured the problem, in that the variables participating in common constraints are close to each other in the ordering. The *minimum bandwidth problem* seeks to find a variable ordering that minimizes the bandwidth ([54, 21, 24, 35, 55, 60, 66]). This underlying structure, when present in A, can be captured by BDDs and results in good computational performance.

4.6.1 Problem Generation

Our random matrices are generated according to three parameters: the number of variables n, the number of ones per row k, and the bandwidth b_w. For a fixed n, k, and b_w, a random matrix A is constructed as follows. We first initialize A as a zero matrix. For each row i, we assign the ones by selecting k columns uniformly at random from the index set corresponding to the variables $\{x_i, x_{i+1}, \ldots, x_{i+b_w}\}$. As an example, a constraint matrix with $n = 9$, $k = 3$, and $b_w = 4$ may look like

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

Consider the case when $b_w = k$. The matrix A has the *consecutive ones property* and is totally unimodular [26] and IP finds the optimal solution for the set packing and set covering instances at the root node. Similarly, we argue that an $(m+1)$-width restricted BDD is an exact BDD for both classes of problems, hence also yielding an optimal solution for this structure. Indeed, this structure implies that the state of a BDD node u is always of the form $\{j, j+1, \ldots, m\}$ for some $j \geq \ell(u)$ during top-down compilation.

To see this, consider the set covering problem. We claim that for any partial solution x' (that can be completed to a feasible solution), $\mathtt{s}(x') = \{i(x'), i(x')+1, \ldots, m\}$ for some index $i(x')$ (or $\mathtt{s}(x') = \emptyset$ if x' completed with 0's already satisfies all of the constraints). Let $j' \leq j$ be the largest index in x' with $x'_j = 1$. Because x' can be completed to a feasible solution, for each $i \leq b_w + j - 1$ there is a variable x_{j_i} with $a_{i,j_i} = 1$. All other constraints must have $x_j = 0$ for all i with $a_{i,j} = 0$. Therefore $\mathtt{s}(x') = \{b_w + j, b_w + j + 1, \ldots, m\}$, as desired. Therefore the state for every partial solution must be of the form $i, i+1, \ldots, m$ or $\emptyset$. There are at most $m+1$ such states, and so the size of any layer cannot exceed $(m+1)$. A similar argument works in the case of the SPP.

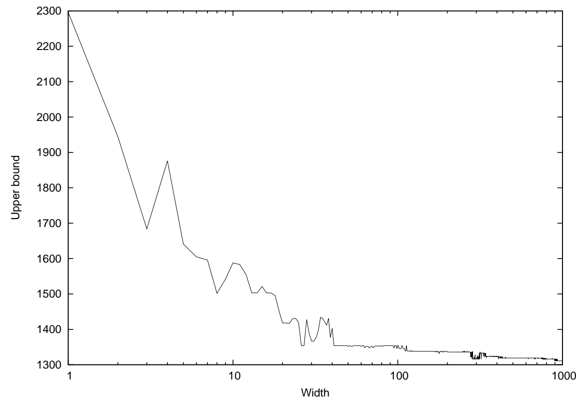
Increasing the bandwidth b_w, however, destroys the totally unimodular property of A and the bounded width of B. Hence, by changing b_w, we can test how sensitive IP and the BDD-based heuristics are as the staircase structure dissolves.

We note here that generating instances of this sort is not restrictive. Once the bandwidth is large, the underlying structure dissolves and each element of the matrix becomes randomly generated. In addition, as mentioned above, algorithms to exactly (or approximately) solve the minimum bandwidth problem have been investigated. To any SCP or SPP one can therefore apply these methods to reorder the matrix and then apply the BDD-based algorithm.

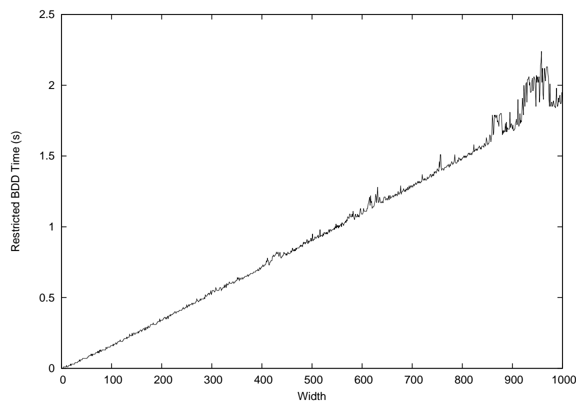
4.6.2 Relation between Solution Quality and Maximum BDD Width

We first analyze the impact of the maximum width W on the solution quality provided by a restricted BDD. To this end, we report the generated bound versus maximum width W obtained for a set covering instance with $n = 1000$, $k = 100$, $b_w = 140$, and a cost vector c where each c_j was chosen uniformly at random from the set $\{1, \ldots, nc_j\}$, where nc_j is the number of constraints in which variable j participates. We observe that the reported results are common among all instances tested.

Figure 4.5a depicts the resulting bounds, where the width axis is in log-scale, and Figure 4.5b presents the total time to generate the W-restricted BDD and extract its best solution. We tested all W in the set $\{1, 2, 3, \ldots, 1000\}$. We see that as the width increases, the bound approaches the optimal value, with a super-exponential-like convergence in W.



(a) Upper Bound



(b) Time

Figure 4.5: Restricted BDD performance versus the maximum allotted width for a set covering instance with $n = 1000$, $k = 100$, $b_w = 140$, and random cost vector.

The time to generate the BDD grows linearly in W, as expected from the complexity result in Section 4.4.

4.6.3 Set Covering

We report the results for two representative classes of instances for the set covering problem. In the first class, we studied the effect of b_w on the quality of the bound. To this end, we fixed $n = 500$, $k = 75$, and considered b_w as a multiple of k, namely $b_w \in \{\lfloor 1.1k \rfloor, \lfloor 1.2k \rfloor, \ldots, \lfloor 2.6k \rfloor\}$. In the second class, we analyzed if k, which is propor-

tional to the density of A, also has an influence on the resulting bound. We then considered $n = 500$, $k \in \{25, 50, \ldots, 250\}$, and fixed $b_w = 1.6k$ for this case. In all classes, we generated 30 instances for each triple (n, k, b_w) and fixed 500 as the restricted BDD maximum width.

It is well-known that the objective function coefficients play an important role in the bound provided by IP solvers for the set covering problem. We considered two types of cost vectors c in our experiments. The first is $c = \mathbf{1}$, which yields the *combinatorial* set covering problem. For the second cost function, let nc_j be the number of constraints that include variable x_j, $j = 1, \ldots, n$. We chose the j-th cost of variable x_j uniformly at random from the range $[0.75nc_j, 1.25nc_j]$. As a result, variables that participate in more constraints have a higher cost, thereby yielding harder set covering problems to solve. This cost vector yields the *weighted* set covering problem.

To compare the quality of the feasible solutions, we first obtained a lower bound for each instance by running CPLEX with its default settings. We then measured the gap difference with respect to the IP bound, which is obtained by taking the difference between the IP gap and the BDD gap and dividing the result by the IP gap.

The results for the first instance class are presented in Table 4.2 and Figure 4.6. The labels *IP-G*, *RB-G*, *IP-T*, and *RB-T* represent the IP gap, the restricted BDD gap, the IP time, and the restricted BDD time, respectively. Each data point in the figure represents the average over the gaps of the individual instances for a particular triple (n, k, b_w). We observe that the restricted BDD yields a significantly better solution for small bandwidths in the combinatorial set covering version. As the bandwidth increases, the staircase structure is lost and the BDD gap becomes progressively worse in comparison to the IP gap. This happens since an exact reduced BDD for larger bandwidth matrices would have larger width as well. Thus, more information is lost when we restrict the BDD size. The same behavior is observed for the weighted set covering problem, although we notice that the gap provided by the restricted BDD is generally better in comparison to the IP gap even for larger bandwidths. Finally, we note that the restricted BDD time is also comparable to the IP time. This time takes into account both BDD construction and extraction of the best solution it encodes by means of a shortest path algorithm.

The results for the second instance class are presented in Table 4.3 and Figure 4.7,

	Combinatorial				Weighted			
b_w/k	IP-G	RB-G	IP-T(s)	RB-T(s)	IP-G	RB-G	IP-T(s)	RB-T(s)
1.1	36.16	22.99	0.59	0.27	37.02	24.25	0.36	0.27
1.2	37.67	27.21	0.74	0.31	41.51	31.05	0.50	0.33
1.3	40.43	30.06	0.78	0.33	44.55	33.51	0.56	0.35
1.4	43.49	31.69	0.76	0.35	48.78	37.40	0.64	0.37
1.5	45.80	36.10	0.82	0.36	50.24	39.80	0.69	0.37
1.6	48.27	40.16	0.74	0.35	50.56	41.15	0.74	0.36
1.7	50.50	43.47	0.69	0.35	49.84	42.77	0.80	0.37
1.8	50.36	46.24	0.73	0.36	49.73	44.27	0.81	0.37
2.0	50.47	46.44	0.77	0.36	50.39	45.58	0.82	0.37
2.1	49.76	46.03	0.69	0.37	49.95	46.57	0.84	0.37
2.2	50.67	47.58	0.61	0.37	47.94	47.97	0.84	0.37
2.3	47.98	49.38	0.53	0.37	49.74	48.54	0.81	0.37
2.4	48.65	51.29	0.52	0.37	50.36	49.12	0.83	0.37
2.5	48.10	52.58	0.55	0.37	49.07	50.22	0.79	0.37
2.6	46.96	53.25	0.56	0.34	49.93	49.87	0.80	0.35

Table 4.2: Combinatorial and weighted set covering results for $n = 500$, $k = 75$, and varying bandwidth. The data represents the average over all instances with that particular bandwidth.

similarly as before. In this case we note that restricted BDDs provide better solutions when k is smaller. The intuition behind this behavior is due to the fact that, when the matrix is sparser, a variable affects a smaller number of constraints. Hence, the possible number of BDD node states is smaller, and less information is lost by restricting the BDD width. Moreover, we notice again that the relative gap for the weighted set covering is usually better than the combinatorial case. Finally, we observe that the restricted BDD time is relatively constant for all instances, outperforming the IP solver when k is small.

4.6.4 Set Packing

We extend the same experimental analysis of the previous section to set packing instances. Namely, we generated two classes of instances: the fist considers variations of the bandwidth, while the second considers variations of the density of the constraint matrix A. We observed that for most values of (n, k, b_w), both restricted BDD and IP had a very similar performance with respect to the final generated gap, since set packing problems with the studied structure

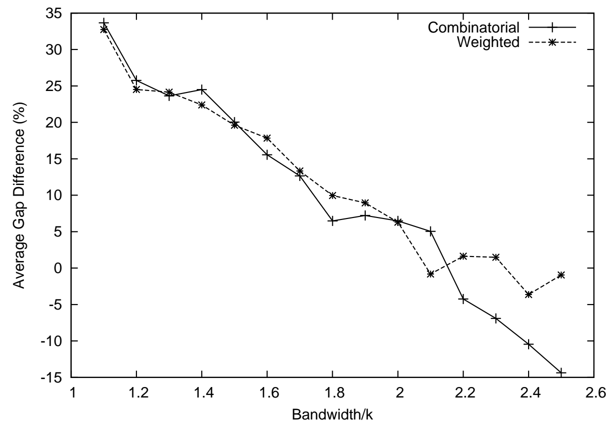


Figure 4.6: Average gap difference for the combinatorial and weighted set covering instances with $n = 500$, $k = 75$, and varying bandwidth.

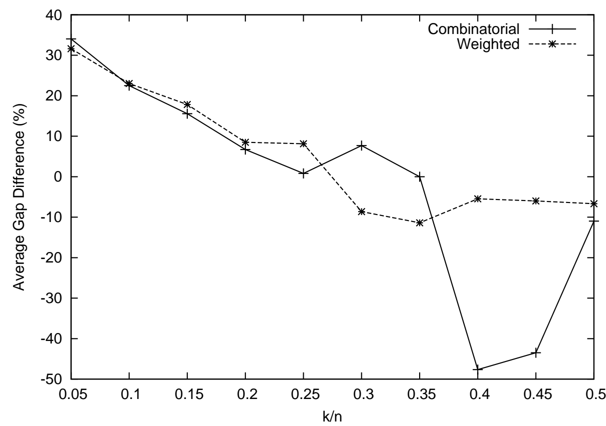


Figure 4.7: Average gap difference for the combinatorial and weighted set covering instances with $n = 500$, varying k, and $b_w = 1.6k$.

	Combinatorial				Weighted			
k/n	**IP-G**	**RB-G**	**IP-T(s)**	**RB-T(s)**	**IP-G**	**RB-G**	**IP-T(s)**	**RB-T(s)**
0.05	42.42	27.84	0.66	0.28	40.61	27.45	0.55	0.29
0.10	44.75	34.34	0.89	0.33	47.59	36.27	0.74	0.35
0.15	48.27	40.16	0.74	0.35	50.56	41.15	0.74	0.36
0.20	49.72	44.90	0.50	0.38	50.43	45.60	0.68	0.38
0.25	49.70	48.35	0.54	0.37	52.36	47.49	0.59	0.38
0.30	49.52	44.52	0.26	0.39	47.79	51.19	0.50	0.36
0.35	56.88	56.88	0.25	0.36	46.39	50.74	0.41	0.34
0.40	42.84	59.51	0.27	0.33	46.62	48.45	0.31	0.33
0.45	40.88	56.71	0.23	0.33	44.08	45.97	0.23	0.32
0.50	48.30	51.08	0.16	0.30	40.31	42.15	0.16	0.28

Table 4.3: Combinatorial and weighted set covering results for $n = 500$, varying k, and $b_w = 1.6k$. The data represents the average over all instances with that particular k.

are usually easy for all techniques. The only noticeable exception occurred when A was sparse. Hence, for the first instance class we fixed $n = 1000$, $k = 50$, and took b_w again as a multiple of k, namely $b_w \in \{\lfloor 1.1k \rfloor, \lfloor 1.2k \rfloor, \ldots, \lfloor 2.5k \rfloor\}$. In the second class we considered $n = 1000$, $k \in \{25, 50, \ldots, 250\}$, and fixed $b_w = 1.8k$. In all classes, we generated 30 instances for each triple (n, k, b_w) and fixed 500 as the restricted BDD maximum width.

Similar to set covering, experiments were performed with two types of objective function coefficients. The first, $c = \mathbf{1}$, yields the *combinatorial* set packing problem. For the second cost function, let nc_j again be the number of constraints that include variable x_j, $j = 1, \ldots, n$. Moreover, let $p = \max_j \{nc_j\}$. We chose the j-th cost of variable x_j uniformly at random from the range $[0.75(p - nc_j + 1), 1.25(p - nc_j + 1)]$. As a result, variables that participate in fewer constraints have a higher cost, thereby yielding harder set packing problems to solve. This cost vector yields the *weighted* set packing problem.

The results for the first instance class are presented in Table 4.4 and Figure 4.8, similarly as before. We observed that, for all tested instances, the solution obtained from the BDD restriction was at least as good as the IP solution for all cost functions. As the bandwidth increases, the gap increases for both techniques (as suggested by Table 4.4). Nonetheless, the bounds derived from the IP technique increase faster than the BDD bound, which explains why the relative gap for the BDD is progressively better. The same reasoning can be applied

to the weighted set packing version. Also note that the BDD time is an order of magnitude faster than the IP root node.

	Combinatorial				**Weighted**			
$\mathbf{b_w/k}$	**IP-G**	**RB-G**	**IP-T(s)**	**RB-T(s)**	**IP-G**	**RB-G**	**IP-T(s)**	**RB-T(s)**
1.1	0.96	0.49	1.87	0.03	0.36	0.12	0.30	0.03
1.2	0.17	0.17	2.16	0.03	0.52	0.11	0.51	0.03
1.3	0.83	0.00	2.28	0.04	1.06	0.10	0.62	0.03
1.4	0.90	0.21	2.45	0.04	1.36	0.17	0.72	0.04
1.5	0.91	0.18	2.66	0.04	1.40	0.22	0.78	0.04
1.6	1.62	0.23	2.82	0.05	1.89	0.25	0.81	0.05
1.7	3.21	0.52	3.08	0.05	1.53	0.29	0.85	0.05
1.8	2.14	0.63	3.28	0.06	2.05	0.44	0.86	0.06
1.9	3.67	0.90	3.41	0.06	1.50	0.37	0.91	0.06
2.0	3.76	1.61	3.39	0.07	2.36	0.37	0.90	0.07
2.1	3.91	1.35	3.55	0.07	1.90	0.26	0.93	0.07
2.2	3.36	1.62	3.57	0.09	1.77	0.27	0.94	0.09
2.3	4.05	1.26	3.73	0.09	1.47	0.26	0.99	0.09
2.4	3.65	1.55	3.64	0.11	1.64	0.39	1.01	0.10
2.5	4.24	1.74	3.59	0.11	1.91	0.48	1.01	0.12

Table 4.4: Combinatorial and weighted set packing results for $n = 1000$, $k = 50$, and varying bandwidth. The data represents the average over all instances with that particular bandwidth.

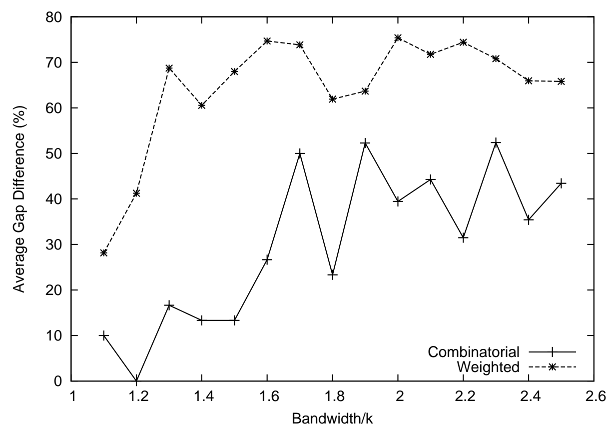


Figure 4.8: Average gap difference for the combinatorial and weighted set packing instances with $n = 1000$, $k = 50$, and varying bandwidth.

The results for the second instance class are presented in Table 4.5 and Figure 4.9, similarly as before. For all instances tested, the BDD bound was at least as good as the bound obtained with IP. The analysis is similar to the set covering case, in which restricted BDDs provide better solutions when k is smaller. This is due to the fact that, since A is sparser, fewer BDD node states are possible in each layer, which implies that less information is lost by restricting the BDD width. Finally, we observe that for small $\frac{k}{n}$ the restricted BDD approach is again an order of magnitude faster than IP, although this changes as $\frac{k}{n}$ increases.

	Combinatorial				Weighted			
k/n	**IP-G**	**RB-G**	**IP-T(s)**	**RB-T(s)**	**IP-G**	**RB-G**	**IP-T(s)**	**RB-T(s)**
0.05	2.14	0.63	3.28	0.06	2.05	0.44	0.86	0.06
0.10	0.64	0.12	3.02	0.12	1.09	0.19	1.00	0.12
0.15	5.82	5.55	2.02	0.18	4.02	3.96	0.92	0.18
0.20	12.27	11.88	1.23	0.22	3.61	3.24	0.86	0.22
0.25	8.46	8.46	0.84	0.21	4.96	4.43	0.76	0.21
0.30	8.08	8.08	0.72	0.20	5.80	5.04	0.71	0.21
0.35	6.25	6.25	0.68	0.19	5.43	4.56	0.68	0.19
0.40	8.24	7.86	0.59	0.18	5.90	4.35	0.59	0.19
0.45	7.10	6.32	0.44	0.16	5.11	4.51	0.45	0.16
0.50	6.80	6.80	0.19	0.15	4.52	3.91	0.19	0.16

Table 4.5: Combinatorial and weighted set packing results for $n = 1000$, varying k, and $b_w = 1.8k$. The data represents the average over all instances with that particular k.

4.7 Conclusion

We introduce a new structure, restricted BDDs, and describe how they can be used to develop a new class of general-purpose heuristics for binary optimization problems. A restricted BDD is a limited-size directed acyclic multigraph that represents an under-approximation of the feasible set. We apply this technique to randomly generated set covering and set packing instances and find that it can yield substantially better solutions than state-of-the-art integer programming technology when the constraint matrix has a small bandwidth.

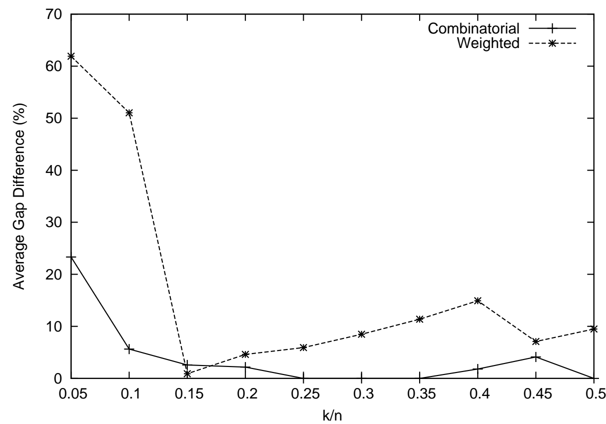


Figure 4.9: Average gap difference for the combinatorial and weighted set packing instances with $n = 1000$, varying k, and $b_w = 1.8k$.

The results indicate that restricted BDDs could become a useful addition to the existing library of heuristics for binary optimization problems. Several aspects of the algorithm may still need to be further investigated, including the application to broader classes of problems and how these structures can be incorporated into existing complete or heuristic methods. For example, they could be used as an additional primal heuristic during a branch-and-bound search. Moreover, restricted BDDs could also be applied to problems for which no strong linear programming relaxation is known, since they can accommodate constraints of arbitrary form.

Chapter 5

Decision Diagram-Based Branch and Bound

5.1 Introduction

Discrete optimization problems model a wide-range of problems arising in a wide-range of industries, including business analytics, process improvement, and health care operations, to name a few. Due to the computational difficulty in solving these problems, one typically relies on branch-and-bound algorithms to solve these problems.

Branch-and-bound algorithms proceed by iteratively defining and approaching smaller subproblems, calculating relaxation bounds and heuristic solutions for each subproblem, with the goal of finding an improving solution, or proving that one cannot exists in that area of the search space.

Traditionally, continuous relaxations are used for obtaining relaxation bounds, due in part to the vast research effort in this area. Examples of these include linear programming (LP) relaxations and semi-definite programming (SDP) relaxations. Primal heuristics are employed in conjunction with these continuous relaxations, which are used to find feasible/improving solutions.

Although these techniques have been successfully applied across a broad range of appli-

cation areas, research has been limited in searching for new solution methodologies for this class of optimization problems. With the exponential explosion of data and computational power, businesses, more than ever, are seeking to apply optimization methods to better the way they operate. As the problems they seek to solve grow larger and more complex, it is crucial to improve on existing technologies on this class of optimization problems.

In this chapter, we explore the idea of using decision diagrams (DDs) for solving discrete optimization problems. We discuss how DDs can be used for relaxation bounds, searching for improving feasible solutions, and defining subproblems. This approach differs substantially from the traditional methods in several important ways:

- We employ *discrete* relaxations in the form of *relaxed* DDs (Chapters 2,chap:tighten). Continuous relaxations have been successful, but often time problems do not admit tight, or even easily defined, continuous relaxations. As an alternative, we suggest using relaxed DDs for this purpose.

- We use relaxed DDs to define subproblems. The approach discussed differs substantially from standard branch-and-bound algorithms in a number of important ways. First, the algorithm relies on approximate DDs for relaxation bounds, a discrete structure, which differs from the typical continuous relaxations that are employed for this class of problems. In addition, search is performed not on single value assignments to variables, but rather on pools of partial solutions, thereby eliminating certain symmetries.

- We use DDs as the primal heuristic in the branch-and-bound procedure. These *restricted* DDs (Chapter 4) are generated at search tree nodes using the same information that is necessary to build relaxed DDs and so are easily integrated with the rest of the procedure.

The remainder of the chapter is organized as follows. We first describe binary optimization problems and the application of binary decision diagrams for representing the feasible set for this class of problems. We then describe *approximate decision diagrams* and then discuss how to define subproblems using relaxed DDs. We then describe the branch-and-bound algorithm in detail and discuss computational results on the application of the algorithm to

the maximum independent set problem. These experiments shows promise as the algorithms outperforms state-of-the-art integer programming technology. We then conclude with some final remarks and suggestion of future research.

5.2 Binary Optimization Problems

Binary optimization problems (BOPs) are specified by a set of binary variables $X = \{x_1, \ldots, x_n\}$, an objective function $f : \{0,1\}^n \to \mathbb{R}$ to be minimized, and a set of m constraints $C = \{C_1, \ldots, C_m\}$, which define relations among the problem variables. A *solution* to a BOP P is an assignment of values 0 or 1 to each of the variables in X. A solution is *feasible* if it satisfies all the constraints in C. The set of feasible solutions of P is denoted by $\mathrm{Sol}(P)$. A solution x^* is *optimal* for P if it is feasible and satisfies $f(x^*) \leq f(\tilde{x})$ for all $\tilde{x} \in \mathrm{Sol}(P)$. We denote by $x^*(P)$ an optimal solution and let $z^*(P)$ be the value of the optimal solution.

We will be discussing the application of the methodology presented here in the context of several classical BOPs. We define these problems here.

Knapsack Problem (KP): Given a set of items $1, \ldots, n$, each with size $s_j \geq 0$ and profit p_j, and a knapsack of size S, find the set of items K of maximum total value with total size not exceeding S.

The KP can be formulated as a binary optimization problem. Associating a binary variable x_j with each item, we write

$$
\begin{aligned}
\text{maximize} \quad & \sum_{j=1}^{n} p_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{n} s_j x_j \leq S \\
& x_j \in \{0,1\}, \quad j = 1, \ldots, n
\end{aligned}
$$

Inverse Knapsack Problem (IKP): Given a set of items $1, \ldots, n$, each with size $s_j \geq 0$ and cost c_j, and a minimum size of S, find the set of items K of minimum total cost with total size at least S.

The IKP can be formulated as a binary optimization problem. Associating a binary

variable x_j with each item, we write

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} s_j x_j \geq S \\ & x_j \in \{0,1\}, \quad j = 1, \dots, n \end{aligned}$$

Weighted Maximum Independent Set Problem (WMISP): Given a graph $G = (V, E)$ and weights w_v for each vertex $v \in V$, find the subset of variables I, all mutually non-adjacent, of largest total weight. When $w_v = 1$ for all vertices, the problem is referred to as the Maximum Independent Set Problem (MISP).

We can cast the WMISP as a BOP by associating a binary variables x_v with each vertex:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} w_v x_v \\ \text{subject to} \quad & x_v + x_{v'} \leq 1, \quad \forall (v, v') \in E \\ & x_v \in \{0,1\}, \quad \forall v \in V \end{aligned}$$

Weighted Maximum Clique Problem (WMCP): Given a graph $G = (V, E)$ and weights w_v for each vertex $v \in V$, find the subset of variables I, all mutually adjacent, of largest total weight. When $w_v = 1$ for all vertices, the problem is referred to as the Maximum Clique Problem (MCP).

We can cast the WMCP as a BOP by associating a binary variables x_v with each vertex:

$$\begin{aligned} \text{maximize} \quad & \sum_{j=1}^{n} w_v x_v \\ \text{subject to} \quad & x_v + x_{v'} \leq 1, \quad \forall (v, v') \notin E \\ & x_v \in \{0,1\}, \quad \forall v \in V \end{aligned}$$

Set Covering Problem (SCP):

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \geq e \\ & x_j \in \{0,1\}, \quad j = 1, \dots, n \end{aligned}$$

where c is an n-dimensional real-valued vector, A is a 0–1 $m \times n$ matrix, and e is the m-dimensional unit vector. Let $a_{i,j}$ be the element in the i-th row and j-th column of A, and define $A_j = \{i \mid a_{i,j} = 1\}$ for $j = 1, \ldots, n$. The SCP asks for a minimum-cost subset $V \subseteq \{1, \ldots, n\}$ of the sets A_j such that for all $i, a_{i,j} = 1$ for some $j \in V$, i.e. V *covers* $\{1, \ldots, m\}$.

Set Packing Problem (SPP):

$$\begin{aligned} \text{maximize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq e \\ & x_j \in \{0, 1\}, \quad j = 1, \ldots, n \end{aligned}$$

where c is an n-dimensional real-valued vector, A is a 0–1 $m \times n$ matrix, and e is the m-dimensional unit vector. Let $a_{i,j}$ be the element in the i-th row and j-th column of A, and define $A_j = \{i \mid a_{i,j} = 1\}$ for $j = 1, \ldots, n$. The SPP asks for a maximum-cost subset $V \subseteq \{1, \ldots, n\}$ of the sets A_j such that for all $i, a_{i,j} = 1$ for at most one $j \in V$.

Binary Integer Programming (IP): Given a real-valued matrix A, vector c, and vector b:

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x_j \in \{0, 1\}, \quad j = 1, \ldots, n \end{aligned}$$

The other problems listed above are all special cases of IP but it will be beneficial to refer to them individually as well.

5.3 Binary Decision Diagrams

Here we describe binary decision diagrams and how they are used to represent a set of solutions to a given BOP. We first describe standard binary decision diagrams and then discuss various forms of compressed binary decision diagrams that may be useful for particular applications.

5.3.1 Standard Binary Decision Diagrams

A *binary decision diagram* (BDD) $B = (U, A)$ for a BOP P is a layered directed acyclic multi-graph that encodes a set of solutions of P. The nodes U are partitioned into $n+1$ layers, $L_1, L_2, \ldots, L_{n+1}$, where we let $\ell(u)$ be the layer index of node u. Layers L_1 and L_{n+1} consist of single nodes; the root r and the terminal t, respectively. The *width* of layer j is given by $\omega_j = |L_j|$, and the *width* of B is $\omega(B) = \max_{j \in \{1,2,\ldots,n\}} \omega_j$. The *size* of B, denoted by $|B|$, is the number of nodes in B.

Each arc $a \in A$ is directed from a node in some layer j to a node in the adjacent layer $j+1$, and has an associated *arc-domain* $d_a \in \{0, 1\}$. The arc a is called a *1-arc* when $d_a = 1$ and a *0-arc* when $d_a = 0$. For any two arcs a, a' directed out of a node u, $d_a \neq d_{a'}$, so that the maximum out-degree ($d^+(u)$) of a node u in a BDD is 2, with each arc having a unique arc-domain. Given a node u, we let $a_0(u)$ be the 0-arc directed out of u (if it exists) and $b_0(u)$ be the node in $L_{\ell(u)+1}$ at its opposite end, and similarly for $a_1(u)$ and $b_1(u)$.

A BDD B represents a set of solutions to P in the following way. An arc a directed out of a node u represents the assignment $x_{\ell(u)} = d_a$. Hence, for two nodes u, u' with $\ell(u) < \ell(u')$, a directed path p from u to u' along arcs $a_{\ell(u)}, a_{\ell(u)+1}, \ldots, a_{\ell(u')-1}$ corresponds to the assignment $x_j = d_{a_j}$, $j = \ell(u), \ell(u)+1, \ldots, \ell(u')-1$. In particular, an r–t path $p = (a_1, \ldots, a_n)$ corresponds to a solution x^p, where $x^p_j = d_{a_j}$ for $j = 1, \ldots, n$. The set of solutions represented by a BDD B is denoted by $\mathrm{Sol}(B) = \{x^p \,|\, p \text{ is an } r\text{–}t \text{ path}\}$.

Let B be a BDD and P a BOP. B is an *exact* BDD for P if $\mathrm{Sol}(B) = \mathrm{Sol}(P)$. B is a *relaxed* BDD for P if $\mathrm{Sol}(B) \supseteq \mathrm{Sol}(P)$ and is a *restricted* BDD for P if $\mathrm{Sol}(B) \subseteq \mathrm{Sol}(P)$

For two nodes $u, u' \in U$ with $\ell(u) < \ell(u')$, let $B_{u,u'}$ be the BDD induced by the nodes that belong to some directed path between u and u'. In particular, $B_{r,t} = B$. A BDD is called *reduced* if $\mathrm{Sol}(B_{u,u'})$ is unique for any two nodes u, u' of B. The reduced BDD B is unique when the variable ordering is fixed, and therefore the most compact representation in terms of size for that ordering [68].

Finally, for a large class of objective functions (e.g., for additively separable functions), optimizing over the solutions represented by a BDD B can be reduced to finding a longest path in B. For example, given a real cost vector c and a linear objective function $c^T x$, we can associate an *arc-cost* $c(u, v) = c_{\ell(u)} d_{u,v}$ with each arc $a = (u, v)$ in the BDD. This way,

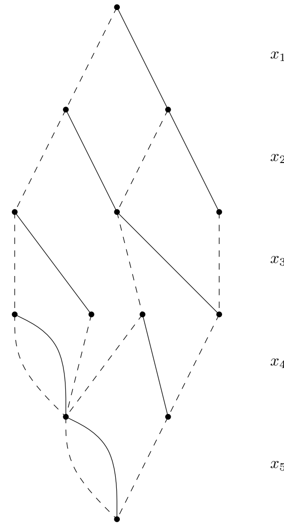


Figure 5.1: Reduced BDD for the BOP in Example 10.

a longest r–t path corresponds to a maximum cost solution in Sol(B). If B is exact, then this longest path corresponds to an optimal solution for P. We denote by $z^*(B)$ the length of the longest path in B, $p^*(B)$ the longest path in B, and $x^*(B)$ the solution corresponding to $p^*(B)$.

Example 10 Consider the following KP.

$$\begin{aligned} \text{maximize } & 2x_1 + 3x_2 + 5x_3 + x_4 + 4x_5 \\ \text{subject to } & 2x_1 + 2x_2 + 3x_3 + 3x_4 + 2x_5 \leq 5 \\ & x_j \in \{0,1\}, \;\; j = 1, \ldots, 5 \end{aligned}$$

In Figure 5.1 we show an exact reduced BDD for P. The 0-arcs are represented by dashed lines, while the 1-arcs are represented by solid lines. There are 15 paths in the BDD, which correspond to the 15 feasible solutions of this BOP. Assigning arc costs of 0 to all of the 0-arcs and the cost coefficient of x_j to the 1-arcs on layer j, $j = 1, \ldots, 5$, the longest path in the BDD correspond to the solution $(0, 0, 1, 0, 1)$, the optimal solutions for P. $\square$

5.3.2 Compressed Binary Decision Diagrams

For different applications, it may be useful to have arcs that can skip layers in order to reduce the size of the BDD by eliminating intermediate nodes. Depending on the particular BOP, the useful form of the compression may change. In this section we will describe three different types of compression and describe problem classes that may benefit from them.

0-BDDs

Zero-compressed BDDs were have been previously introduced. We use the notation 0-BDDs in this chapter for clarity but note that they correspond to the previously introduced structures.

Unlike standard BDDs, 0-BDDs may have arcs that skip layers. In a 0-BDD, an arc $a = (u, v)$ with $u \in L_j$ and $v \in L_{j'}$, $j < j'$, will have an arc-domain $d_a \in \{0, 1\}$ as before. However, in a 0-BDD, the arc represents the set of assignments $x_j = d_a, x_k = 0$, for $k = j+1, \ldots, j'-1$. In particular, in a 0-BDD, an $r-t$ path p on nodes $r = u_1, u_2, \ldots, u_k = t$ (which may now be on fewer than $n + 1$ nodes) corresponds to the solution x with

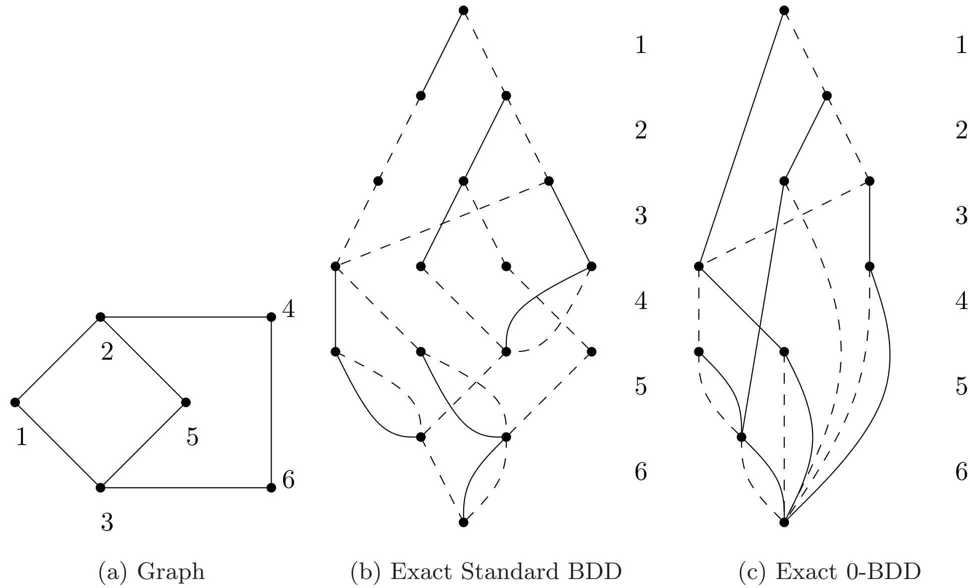
$$x_j = \begin{cases} d_{(u_i,u_{i+1})} & , \text{ for } j = \ell(u_i), i = 1, \ldots, k-1 \\ 0 & , \text{ otherwise} \end{cases}$$

As with standard BDDs, a solution in $\text{Sol}(B)$ that maximizes (minimizes) an additively separable objective function can be identified by a simple longest (shortest) path calculation. Let $z = \sum_{j=1}^{n} c_j(x_j)$. For each arc $a = (u, v)$ we let

$$c(a) = c_{\ell(u)}(d_a) + \sum_{j=\ell(u)+1}^{\ell(v)-1} c_j(0).$$

With these arc costs, a maximum (minimum) length $r - t$ path will correspond to a solution that maximizes (minimizes) z.

Example 11 Consider the graph in Figure 5.2a where we have labeled the vertices with numbers. Depicted in Figures 5.2b and 5.2c are a standard BDD and 0-BDD, respectively, for the independent sets in the graph. Notice that using a 0-BDD allows for a reduction in the size of the BDD. The standard BDD has a total of 17 nodes, as opposed to the 10 nodes



(a) Graph (b) Exact Standard BDD (c) Exact 0-BDD

Figure 5.2: BDD representation of independent sets in a graph using standard BDDs and 0-BDDs

in the 0-BDD, and the width of the standard BDD is 4, while it is only 2 for the 0-BDD. As both structures represent the same set of solutions, it is apparent from this example that there is a benefit to using a 0-BBD to represent the collection of independent sets in a graph.

□

1-BDDs

One-compressed BDDs, 1-BDDs, may have arcs that skip layers. In a 1-BDD, an arc $a = (u, v)$ with $u \in L_j$ and $v \in L_{j'}$, $j < j'$, will have an arc-domain $d_a \in \{0, 1\}$ as before. However, in a 1-BDD, the arc represents the set of assignments $x_j = d_a, x_k = 1$, for $k = j+1, \ldots, j'-1$. In particular, in a 1-BDD, an $r-t$ path p on nodes $r = u_1, u_2, \ldots, u_k = t$ (which may now be on fewer than $n+1$ nodes) corresponds to the solution x with

$$x_j = \begin{cases} d_{(u_i, u_{i+1})} & , \text{ for } j = \ell(u_i), i = 1, \ldots, k-1 \\ 1 & , \text{ otherwise} \end{cases}$$

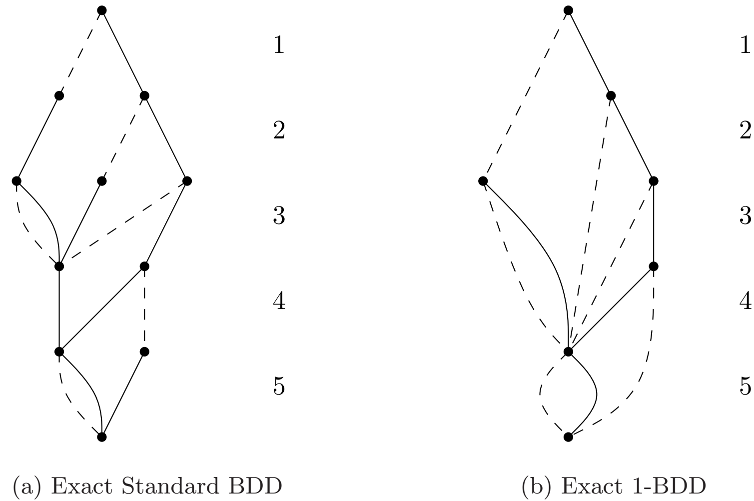


Figure 5.3: BDD representation of the feasible set for the BOP in Example 12 using standard BDDs and 1-BDDs

Again, as with standard BDDs, a solution in $\mathrm{Sol}(B)$ that maximizes (minimizes) an additively separable objective function can be identified by a simple longest (shortest) path calculation. Let $z = \sum_{j=1}^{n} c_j(x_j)$. For each arc $a = (u, v)$ we let

$$c(a) = c_{\ell(u)}(d_a) + \sum_{j=\ell(u)+1}^{\ell(v)-1} c_j(1).$$

With these arc costs, a maximum (minimum) length $r-t$ path will correspond to a solution that maximizes (minimizes) z.

Example 12 Consider the feasible set to the IKP with the following constraint:

$$x_1 + 3x_2 + x_3 + 3x_4 + x_5 \geq 6$$

Depicted in Figure 5.3a is a standard BDD for the feasible set and depicted in Figure 5.3b is a 1-BDD. As in Example 11, both BDDs represent the same set of solutions, but in this case the 1-BDD has 7 nodes as opposed to the 11 nodes in the standard BDD. Note that a 0-BDD would not have compressed any nodes, exemplifying that the useful type of compressed BDD is indeed problem specific. □

0/1-BDDs

Zero/One-compressed compressed BDDs, 0/1-BDDs, may also have arcs that skip layers, but these arcs represent multiples sets of assignments to variables as opposed to standard BDDs, 0-BDDs, and 1-BDDs. In a 0/1-BDD, an arc $a = (u, v)$ with $u \in L_j$ and $v \in L_{j'}$, $j < j'$, will have an arc-domain $d_a \in \{0, 1\}$ as before. However, in a 0/1-BDD, the arc represents the set of assignments $x_j = d_a, x_k \in \{0, 1\}$, for $k = j+1, \ldots, j'-1$. In particular, in a 0/1-BDD, an $r - t$ path p on nodes $r = u_1, u_2, \ldots, u_k = t$ (which may now be on fewer than $n + 1$ nodes) corresponds to the set of solutions for which

$$x_j = d_{(u_i,u_{i+1})}, \text{ for } j = \ell(u_i)i = 1, \ldots, k-1,$$

and all other variables are set to 0 or 1.

Once again, one can optimize over the solutions in $\text{Sol}(B)$. Here, however, arc costs $c(a)$ depend on whether we are maximizing or minimizing an objective function. Let $z = \sum_{j=1}^{n} c_j(x_j)$.

If we seek to find an $x \in \text{Sol}(B)$ that maximizes z, we let

$$c(a) = c_{\ell(u)}(d_a) + \sum_{j=\ell(u)+1}^{\ell(v)-1} \max\{c_j(0), c_j(1)\},$$

and a longest $r - t$ path will correspond to a maximum cost solution.

If we seek to find an $x \in \text{Sol}(B)$ that minimizes z, we let

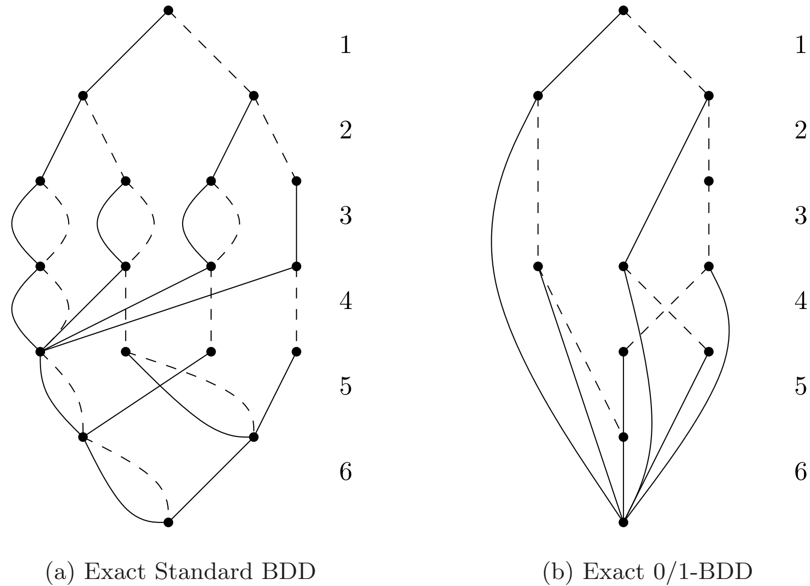
$$c(a) = c_{\ell(u)}(d_a) + \sum_{j=\ell(u)+1}^{\ell(v)-1} \min\{c_j(0), c_j(1)\},$$

and a shortest length $r - t$ path will correspond to a minimum cost solution.

Example 13 For the SCP, a 0/1-BDD representation can yield a lot of compression. Consider a set covering problem on 6 variables with the following constraint matrix:

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

An exact standard BDD has 18 nodes (Figure 5.4a) while a 0/1-BDD has 11 nodes (Figure 5.4b) □



(a) Exact Standard BDD (b) Exact 0/1-BDD

Figure 5.4: BDD representation of feasible set for BOP in Example 13 using standard BDDs and 0/1-BDDs

5.4 Exact BDDs

5.4.1 Exact BDD Compilation

We now describe an algorithm that, given a BOP, produces an exact reduced standard BDD. A simple modification can be used to generate a 0-BDD, 1-BDD, or 0/1-BDD.

An exact reduced BDD $B = (U, A)$ for a BOP P can be interpreted as a compact search tree for P, where infeasible leaf nodes are removed, isomorphic subtrees are superimposed, and the feasible leaf nodes are merged into t. In principle, B can be obtained by first constructing the branching tree for P and reducing it accordingly, which is impractical for our purposes.

We present here an efficient top-down algorithm for constructing an exact BDD B for P. It relies on problem-dependent information for merging BDD nodes and thus reducing its size. If this information satisfies certain conditions, the resulting BDD is reduced. The algorithm is a *top-down* procedure since it proceeds by compiling the layers of B one-by-one, where layer L_{j+1} is constructed only after layers $L_1, \ldots, L_j$ are completed.

We first introduce some additional definitions. Let $x' = (x'_1, \dots, x'_k)$, $k < n$, be a *partial solution* that assigns a value to variables $x_1, \dots, x_k$. We define

$$F(x') = \{x'' \in \{0,1\}^{n-k} \mid x = (x', x'') \text{ is feasible for } P\}$$

as the set of *feasible completions* of x'. We say that two distinct partial solutions x^1, x^2 on variables $x_1, \dots, x_k$ are *equivalent* if $F(x^1) = F(x^2)$.

The algorithm requires a method for establishing when two partial solutions are necessarily equivalent. If this is possible, then the last nodes u, u' of the BDD paths corresponding to these partial solutions can be merged into a single node, since $B_{u,t}$ and $B_{u',t}$ are the same. To this end, with each partial solution x' of dimension k we associate a *state function* $\mathtt{s} : \{0,1\}^k \to S$, where S is a problem-dependent *state space*. The state of x' corresponds to the information necessary to determine if x' is equivalent to any other partial solution on the same set of variables.

Formally, let x^1, x^2 be partial solutions on the same set of variables. We say that the function $\mathtt{s}(x)$ is *sound* if $\mathtt{s}(x^1) = \mathtt{s}(x^2)$ implies that $F(x^1) = F(x^2)$, and we say that $\mathtt{s}$ is *complete* if the converse is also true. The algorithm requires only a sound state function, but if $\mathtt{s}$ is complete, the resulting BDD will be reduced.

For simplicity of exposition, we further assume that it is possible to identify when a partial solution x' cannot be completed to a feasible solution, i.e. $F(x') = \emptyset$. It can be shown that this assumption is not restrictive, but rather makes for an easier exposition of the algorithm. We write $\mathtt{s}(x') = \hat{0}$ to indicate that x' cannot be completed into a feasible solution. If x is a solution to P, we write $\mathtt{s}(x) = \emptyset$ if x is feasible and $\mathtt{s}(x) = \hat{0}$ otherwise.

We now extend the definition of state functions to nodes of the BDD B. Suppose that $\mathtt{s}$ is a complete state function and B is an exact (but not necessarily reduced) BDD. For any node u, the fact that B is exact implies that any two partial solutions $x^1, x^2 \in \text{Sol}(B_{r,u})$ have the same feasible completions, i.e. $F(x^1) = F(x^2)$. Since $\mathtt{s}$ is complete, we must have $\mathtt{s}(x^1) = \mathtt{s}(x^2)$. We henceforth, for exact BDDs, define the state of a node u as $\mathtt{s}(u) = \mathtt{s}(x)$ for any $x \in \text{Sol}(B_{r,u})$, which is therefore uniquely defined for a complete function $\mathtt{s}$.

We also introduce a function $\mathtt{update} : S \times \{0,1\} \to S$. Given a partial solution x' on variables $x_1, \dots, x_k$, $k < n$, and a domain value $d \in \{0,1\}$, the function $\mathtt{update}(\mathtt{s}(x'), d)$ maps the state of x' to the state of the partial solution obtained when x' is appended with

d, $\texttt{s}((x', d))$. This function is similarly extended to nodes: $\texttt{update}(s(u), d)$ represents the state of all partial solutions in $\text{Sol}(B_{r,u})$ extended with value d for a node u.

The top-down compilation procedure is presented in Algorithm 8. We start by setting $L_1 = \{r\}$ and $\texttt{s}(r) = s_0$, where s_0 is an initial state appropriately defined for the problem. Now, having constructed layers $L_1, \ldots, L_j$, we create layer L_{j+1} in the following way. For each node $u \in L_j$ and for $d \in \{0, 1\}$, let $\texttt{s}_{\text{new}} = \texttt{update}(\texttt{s}(u), d)$. If $\texttt{s}_{\text{new}} = \hat{0}$ we do not create arc $a_d(u)$. Otherwise, if there exists some $u' \in L_{j+1}$ with $\texttt{s}(u') = \texttt{s}_{\text{new}}$, we set $b_d(u) = u'$; if such a node does not exist, we create node u_{new} with $\texttt{s}(u_{\text{new}}) = \texttt{s}_{\text{new}}$ and set $b_d(u) = u_{\text{new}}$.

The proof of correctness and running time of the above algorithm is described in previous chapters.

Example 14 Consider the following simple BOP:

$$\begin{aligned} \text{maximize} \quad & 5x_1 + 4x_2 + 3x_3 \\ \text{subject to} \quad & x_1 + x_2 + x_3 \leq 1 \\ & x_j \in \{0, 1\}, \quad j = 1, 2, 3 \end{aligned}$$

We can define $\texttt{s}(x)$ to equal the number of variables set to 1 in x. In this way, whenever $\texttt{s}(x^1) = \texttt{s}(x^2)$ for two partial solutions we have $F(x^1) = F(x^2)$. For example, $\texttt{s}((1, 0)) = 1$ and $\texttt{s}((0, 1)) = 1$, with the only feasible completion being (0).

An update function can be given as follows:

$$\texttt{update}(\texttt{s}(u), d) = \begin{cases} \hat{0} & , d = 1 \text{ and } \texttt{s}(u) = 1 \\ 1 & , d = 1 \text{ and } \texttt{s}(u) = 0 \\ s(u) & , d = 0 \end{cases}$$

With this update function, if in a partial solution there is already one variable set to 1, the update operation will assign $\hat{0}$ to the node on the 1-arc (signifying that the solution cannot be completed to a feasible solution) and 1 to the node on the 1-arc (to signify that still only one variable is set to 1). On the other hand, if a partial solution has no variable set to 1, the 1-arc will now be directed to a node that has state 1 and the 0-arc will be directed to a node with state 0.

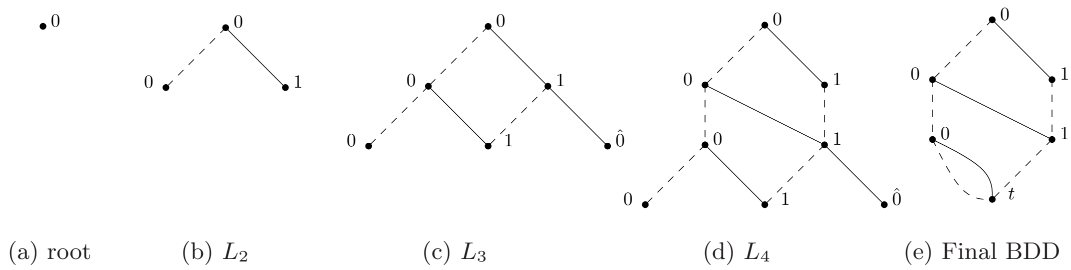
Algorithm 8 Exact BDD Compilation: build_exact($X, s_0,$ `update`)

```
1: INPUT: X, s_0, update // X is the set of variables, s_0 the starting state, and update is
                          // the function used to update the states
2: Create node r with s(r) = s_0
3: L_1 = {r}
4: U = ∅, A = ∅
5: for j = 1 to |X| do
6:   U = U ∪ L_j
7:   L_{j+1} = ∅
8:   for all u ∈ L_j do
9:     for all d ∈ {0,1} do
10:      s_new := update(s(u), d)
11:      if s_new ≠ 0̂ then
12:        if ∃u' ∈ L_{j+1} with s(u') = s_new then
13:          b_d(u) = u'
14:          A = A ∪ {(u, u')}
15:        else
16:          Create node u_new with s(u_new) = s_new
17:          b_d(u) = u_new
18:          L_{j+1} ← L_{j+1} ∪ u_new
19:          A = A ∪ {(u, u_new)}
20: create terminal t
21: merge(L_{|X|+1}, t); L_{|X|+1} = {t}
22: U = U ∪ L_{|X|+1}
23: return B = (U, A)
```

The progression of the algorithm is depicted in Figure 5.5 where after the construction of each layer L_j we show the partially constructed BDD along with the state of each node. Note that the state definition (and update function) for the BOP is a complete state function. The only caveat is that the feasible solution $(0, 0, 0)$ has state 0, while all other feasible

Algorithm 9 merge(M, u')

1: **for all** $u \in M$ **do**
2:     **for all** arcs $a_0(w)$ with $b_0(w) = u$ **do**
3:         $b_0(w) \leftarrow u'$
4:     **for all** arcs $a_1(w)$ with $b_1(w) = u$ **do**
5:         $b_1(w) \leftarrow u'$



(a) root (b) L_2 (c) L_3 (d) L_4 (e) Final BDD

Figure 5.5: Depiction of exact BDD compilation for the BOP in Example 14

solutions have state 1. However, this is corrected in the final line of Algorithm 8 since all nodes in the final layer are merged with the terminal t.

□

5.4.2 Examples

Here we describe state functions for a variety of BOPs. We let x' be a partial solution on the first k variables and give a state for the partial solution and an associated update function.

- Knapsack Problem (KP)

  For the KP, we have $S = \mathbb{R}^+ \cup \hat{0}$, $s_0 = 0$, and

$$\mathtt{s}(x) = \begin{cases} \sum_{j=1}^{k} s_j x'_j & , \sum_{j=1}^{k} s_j x'_j \leq S \\ \hat{0} & , \text{otherwise} \end{cases}$$

$$\mathtt{update}(\mathtt{s}, d) = \begin{cases} \mathtt{s} + s_{k+1} \cdot d & , \sum_{j=1}^{k} s_j x'_j + s_{k+1} \cdot d \leq S \\ \hat{0} & , \text{otherwise} \end{cases}$$

Proposition 1 $\mathtt{s}$ *is a sound state function for the KP and* `update` *calculates* $\mathtt{s}$.

Proof. Proof For any partial solution x' on the first k variables,

$$F(x) = \left\{ (x_{k+1}, \ldots, x_n) : \sum_{j=k+1}^{n} x_j s_j \leq S - \sum_{j=1}^{k} x'_j s_j \right\}.$$

Therefore, if any two partial solutions x^1, x^2 agree on $\sum_{j=1}^{k} x^i_j s_j$ they must have the same feasible completions. In addition, if $\sum_{j=1}^{k} s_j x'_j > S, x'$ cannot be completed to a feasible solution.

`update` exactly calculates the marginal effect of setting $x_k = d$, as desired. □

We note here that developing a complete state function for the KP is NP-hard but it can be done in pseudo-polynomial time/space.

- Inverse Knapsack Problem (IKP)

For the IKP, we also have $S = \mathbb{R}^+ \cup \hat{0}, s_0 = 0$, but in this case

$$\mathtt{s}(x) \quad = \quad \begin{cases} \min\left\{\sum_{j=1}^{k} s_j x'_j, S\right\} & , \sum_{j=1}^{k} s_j x'_j + \sum_{j=k+1}^{n} s_j \geq S \\ \hat{0} & , \text{otherwise} \end{cases}$$

$$\mathtt{update}(\mathtt{s}, d) \quad = \quad \begin{cases} \min\left\{\mathtt{s} + s_{k+1} \cdot d, S\right\} & , \mathtt{s} + s_{k+1} \cdot d + \sum_{j=k+2}^{n} s_j \geq S \\ \hat{0} & , \text{otherwise} \end{cases}$$

Proposition 2 $\mathtt{s}$ *is a sound state function for the IKP and* `update` *calculates* $\mathtt{s}$.

Proof. Proof For any partial solution x' on the first k variables,

$$F(x) = \left\{ (x_{k+1}, \ldots, x_n) : \sum_{j=k+1}^{n} x_j s_j \geq S - \sum_{j=1}^{k} x'_j s_j \right\}.$$

Therefore, if any two partial solutions x^1, x^2 have $\sum_{j=1}^{k} x^i_j s_j \geq S$ they must have the same feasible completions (namely any assignment to the remaining variables) and so assigning a state of S to any solution of this form allows the solutions to be identifying as having the same feasible completions. In addition, if $\sum_{j=1}^{k} x^1_j s_j = \sum_{j=1}^{k} x^2_j s_j < S$, the solution will also have the same set of feasible completions, and so

assigning this common value as the state of a node is a valid assignment. In addition, if $\sum_{j=1}^{k} s_j x'_j + \sum_{j=k+1}^{n} < S, x'$ cannot be completed to a feasible solution so assigning a state of $\hat{0}$ is correct.

`update` exactly calculates the marginal effect of setting $x_k = d$, as desired. □

As in the case of the KP problem, a complete state function for the IKP is NP-hard but it can be done in pseudo-polynomial time/space.

- WMISP

Let $G = (V, E), V = \{v_1, \ldots, v_n\}$ be a graph. For any partial solution x we associate a set of vertices $V(x) = \{v_j : x_j = 1\}$. Let $I(G)$ be the family of all independent sets in a graph. For any vertex v_j, let $N(v_j)$ be the *neighborhood* of v_j (where by convention $v_j \in N(v)$).

Let x be a partial solution on variables $x_1, \ldots, x_k$. $S = 2^V \cup \hat{0}, s_0 = V$, and

$$\mathtt{s}(x) = \begin{cases} \{v_i | i \geq k+1, v_i \cup V(x) \in I(G)\} & , V(x) \in I(G) \\ \hat{0} & , \text{otherwise} \end{cases}$$

$$\mathtt{update}(\mathtt{s}, d) = \begin{cases} \mathtt{s} \setminus \{v_{k+1}\} & , d = 0 \\ \mathtt{s} \setminus N(v_{k+1}) & , d = 1 \text{ and } v_{k+1} \in \mathtt{s} \\ \hat{0} & , d = 1 \text{ and } v_{k+1} \notin \mathtt{s} \end{cases}$$

Proposition 3 `s` *is a complete state function for the WMISP and* `update` *calculates* `s`.

The proof of Proposition 3 appears in Chapter 2 and is based upon interpreting the state as the set of vertices that are non-adjacent to all vertices in the partial solution. If this set is the same for two partial solutions, then they must have the same feasible completions. Interestingly this is also a complete state function so that two solutions have the same feasible completions if and only if they agree on this set.

- WMCP

Let $C(G)$ be the set of cliques in G. Similarly to the WMISP, we let $S = 2^V \cup \hat{0}, s_0 = V$, but now

$$\mathtt{s}(x) = \begin{cases} \{v_i | i \geq k+1, v_i \cup V(x) \in C(G)\} & , V(x) \in C(G) \\ \hat{0} & , \text{otherwise} \end{cases}$$

$$\mathtt{update}(\mathtt{s}, d) = \begin{cases} \mathtt{s} \setminus \{v_{k+1}\} & , d = 0 \\ \mathtt{s} \setminus N(v_{k+1}) & , d = 1 \text{ and } v_{k+1} \in \mathtt{s} \\ \hat{0} & , d = 1 \text{ and } v_{k+1} \notin \mathtt{s} \end{cases}$$

Proposition 4 `s` *is a complete state function for the WMCP and* `update` *calculates* `s`*.*

Proof. Proof Let $G = (V, E)$ be a graph for which we are creating an exact BDD to represent the set of cliques for the WMCP. Consider the complement graph $\bar{G}$ of G. $I(\bar{G}) = C(G)$. Therefore, a complete state function for $I(\bar{G})$ must be a complete state function for $C(G)$. □

- SCP

Let C_i be the set of indices of the variables that participate in constraint i, $C_i = \{j \mid a_{i,j} = 1\}$, and let $\mathtt{last}(C_i) = \max\{j \mid j \in C_i\}$ be the largest index of C_i. We consider the state space $S = 2^{\{1,\ldots,m\}} \cup \{\hat{0}\}$. For a partial solution x' on variables $x_1, \ldots, x_k$, we can write the following state and update functions:

$$\mathtt{s}(x) = \begin{cases} \hat{0}, & \text{if } \exists\, i : \sum_{j=1}^{k} a_{i,j} x'_j = 0 \text{ and } k \geq \mathtt{last}(C_i), \\ \left\{ i : \sum_{j=1}^{k} a_{i,j} x'_j = 0 \right\}, & \text{otherwise.} \end{cases}$$

$$\mathtt{update}(\mathtt{s}, d) = \begin{cases} \mathtt{s}(x') \setminus \{i \mid a_{i,k+1} = 1\}, & d = 1 \\ \mathtt{s}(x'), & d = 0, \forall\, i^* \in \mathtt{s}(x') : \mathtt{last}(C_{i^*}) > k+1 \\ \hat{0}, & d = 0, \exists\, i^* \in \mathtt{s}(x') : \mathtt{last}(C_{i^*}) = k+1 \end{cases}$$

Proposition 5 `s` *is a sound state function for the SCP and* `update` *calculates* `s`*.*

The proof of Proposition 5 has previously been shown. We think of the state as representing the set of constraints that still need some variable set to be set to 1 in order to satisfy that constraint. If two solutions agree upon this set of constraints, they must have the same feasible completions.

`s` (and `update`) can be extended into a complete state function and still run in polynomial time.

- SPP

  Let P be a SPP. One can define a similar state and update function to solutions for the SPP as the SCP. However, another way of defining a state is to note the following. Consider the graph $G = (V, E)$ where we associate a vertex v_j for each variable x_j and let

  $$E = \{(v_j, v_{j'}) | \exists\, i \text{ for which } a_{i,j} = a_{i,j'} = 1\}.$$

  Let P' be the WMISP on G. It is well-known that $\mathrm{Sol}(P) = \mathrm{Sol}(P')$. Therefore, a complete state function for P' is also a complete state function for P.

- IP

  Let P be a general IP (with only binary variables). Let $S = \mathbb{R}^m \cup \hat{0}$, where m is the number of linear inequalities in P. Starting with $s_0 = (0, \ldots 0)$, for a partial solution x', we can let $s(x)_i$ be the sum $\sum_{j=1}^{k} a_{i,j} x'_j$.

  It is clear that any solutions that agree on `s` must have the same feasible completions. $\texttt{update}(\texttt{s}, d)$ can be defined so as to represent the marginal effect of setting $x_{k+1} = d$ (namely either adding $a_{i,k+1}$ to `s` if $d = 1$ or keep `s` constant otherwise).

  This is only a sound state function.

5.5 Approximate BDDs

Exact BDDs are useful for a variety of purposes. As discussed above, they can be viewed as a compact encoding of the entire branching tree. Unfortunately, exact BDDs can be

exponential in size and so we use approximate BDDs to encode over-approximations and under-approximations of the feasible set.

Approximate BDDs come in two forms: relaxed or restricted. Relaxed BDD provide an over-approximation of the feasible set while restricted BDDs provide an over-approximation of the feasible set. These structures are useful for a variety of purposes; for this chapter, we use them to guide branching decisions and to provide optimization bounds. We concentrate on *limited-width* approximate BDDs which cap the size of each layer by a preset maximum width W, thereby enforcing a bound on the size of the entire BDD.

In this section, we describe how to modify Algorithm 8 in order to generate these structures and discuss how to obtain optimization bounds.

5.5.1 Relaxed BDDs

A BDD B is a width-W relaxed BDD for P if $\omega(B) \leq W$ and $\mathrm{Sol}(B) \supseteq \mathrm{Sol}(P)$. Such a structure has size $|B| \leq nW$ and so enforcing a maximum width of W controls the size of the BDD.

We use relaxed BDD for two purposes: generating relaxation bounds and branching. In this section we will describe how to generate relaxed BDDs and derive bounds from them, and in the following sections describe the branching procedure.

A relaxed BDD B can be used to provide an upper-bound on the objective function. As with exact BDDs, a solution in $\mathrm{Sol}(B)$ maximizing an additively separable objective function can be found via a simple longest path calculation. As B is a relaxed BDD, every feasible solution is a candidate, and therefore the value of this longest path must be an upper bound on the objective function.

Example 15 Consider the KP from Example 10. Figure 5.6 depicts a width-3 relaxed BDD for the problem. The relaxed BDD contains 17 solutions, including the 15 feasible solution in the exact BDD in Figure 5.1. The longest path in the exact BDD corresponds to the solution (0,0,1,0,1) which has objective function value 9 while the longest path in the relaxed BDD corresponds to the solution (0,0,1,1,1) which has objective function value 10, an upper bound on the optimal value. □

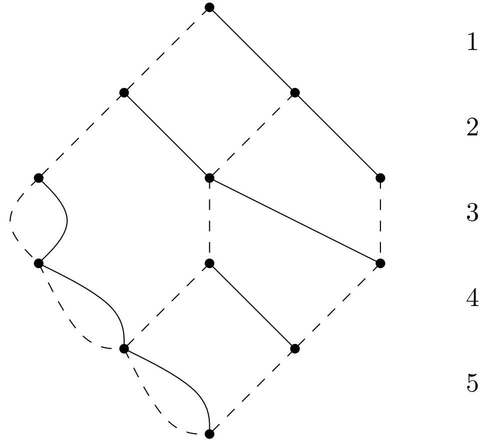


Figure 5.6: Width-3 Relaxed BDD for the BOP in Example 15.

Relaxed BDD Compilation

Before describing an algorithm that can be used to generate a relaxed BDDs, we first define additional notation.

Let P be a BOP with $z(x) = \sum_{j=1}^{n} c_j(x_j)$ (we suppose the objective is to maximize z). For any node $u \in A$, let $P|_u$ be the BOP with objective function $\sum_{j=\ell(u)}^{n} c_j(x_j)$ and feasible set as in P except that feasible solutions are required to have the tuple $(x_1, \dots, x_{\ell(u)-1})$ restricted to $\mathrm{Sol}(B_{r,u})$. In other words, $P|_u$ is the optimization problem only on solutions that can be extended into feasible solutions from the set of partial solutions represented by paths from the root of B to node u. Similarly, $P|_{x'}$ for any partial solution x' is the BOP resulting from fixing the variables in x'. In addition, let $z(u)$ be the longest $r-u$ path in B.

We prove a lemma now that describes how an exact BDD can be generated for a subproblem of a BOP P. It will be used later for generating relaxed BDD and for the branch-and-bound algorithm.

Lemma 8 *Let $x' = (x'_1, \dots, x'_k)$ be a partial solution (that can be extended into a feasible solution) on the first k variables in P and B an exact BDD for P. Let $\mathtt{s}$ be a sound state function for P. If we begin Algorithm 8 with $\mathtt{s}(r) = s_0 = \mathtt{s}(x')$ and run iterations from*

$j = k+1, \ldots, n$, the BDD generated, B', will be an exact BDD for $P|_u$, where $u \in L_{k+1}$ is the node in B for which $x' \in$ Sol$(B_{r,u})$; i.e., $\mathtt{s}$ is a sound state function for $P|_u$ when s_0 is set to $\mathtt{s}(x')$.

Proof. Proof B is an exact BDD for P so $\text{Sol}(B_{u,t}) = F(x)$ for any $x \in \text{Sol}(B_{r,u})$; in particular, $\text{Sol}(B_{u,t}) = F(x')$. In addition, it must be that $\text{Sol}(B_{u,t}) = \text{Sol}(P|_u)$. It therefore suffices to show that $\text{Sol}(B_{u,t}) = \text{Sol}(B')$. But, any path created that starts from u will always be created when Algorithm 8 is run with r having the same state as u, and vice-versa. □

Relaxed BDDs can be generated via a modification of Algorithm 8 and require the definition of a relaxation function $\oplus : S \times S \to S$. This function takes as argument two states s_1, s_2, both corresponding to partial solutions x^1, x^2 on the same set of variables (say $x_1, \ldots, x_k$), and returns a new state $s_{\text{new}} = s_1 \oplus s_2$. A valid relaxation function requires that running Algorithm 8 with state s_{new} as the state of the root node using variables $x_{k+1}, \ldots, x_n$ will generate a BDD B' for which $\text{Sol}(B') \supseteq \text{Sol}(P|_{x^i})$. We will give several examples of functions later in this section, but first describe how to generate a relaxed BDD given such a function.

Algorithm 10 describes a modification to Algorithm 8 that will generate a relaxed BDD. It proceeds exactly as in the case of generating an exact BDD, but whenever a layer L_j exceeds the preset maximum allotted width W, the algorithm selects two nodes in L_j and merges them (along with all arcs directed at either of them). The state of the newly created node is set via the relaxation function $\oplus$ so that no feasible solutions will be lost. Let build_relaxation$(X, s, \mathtt{update})$ be the function that returns a relaxed BDD via this modification.

The quality of the relaxed BDD generated hinges greatly on the selection of nodes to merge in line 2 in Algorithm 10. The authors have investigated several possible heuristics for this choice. In preliminary computational work, selecting the nodes u_1, u_2 that have the worst objective function value (i.e., have lowest $z(u)$ values in L_j) works well and we employ thus heuristic in the computational work described in Section 5.8.

Algorithm 10 Node merger for obtaining a relaxed BDD.

Insert immediately after line 5 of Algorithm 8.

1: **while** $|L_j| > W$ **do**
2: $\quad \{u_1, u_2\} := \text{node_select}(L_j)$
3: $\quad s_{\text{new}} := s(u_1) \oplus s(u_2)$
4: $\quad L_j \leftarrow L_j \backslash \{u_1, u_2\}$
5: $\quad$ **if** $\exists u' \in L_j$ with $s(u') = s_{\text{new}}$ **then**
6: $\qquad \text{merge}(\{u_1, u_2\}, u')$
7: $\quad$ **else**
8: $\qquad$ Create node $\hat{u}$ with $s(\hat{u}) = s_{\text{new}}$
9: $\qquad \text{merge}(\{u_1, u_2\}, \hat{u})$
10: $\qquad L_j = L_j \cup \{\hat{u}\}$

Examples

- Knapsack Problem (KP)

  Let P be a KP and let $\mathtt{s}$ be the state for the KP from Section 5.4.2.

  Proposition 6 *Let*

  $$B = \text{build_exact}(\{x_{k+1}, \dots, x_n\}, s, \texttt{update})$$

  and

  $$B' = \text{build_exact}(\{x_{k+1}, \dots, x_n\}, s', \texttt{update}),$$

  with $s < s'$. *Then* $Sol(B) \supseteq Sol(B')$

  Proof. Proof As S is a sound state function, the function $\text{build_exact}(X, s, \texttt{update})$ will return an exact BDD for $P|_{x'}$, for any partial solution with $\mathtt{s}(x') = s$. The feasible region for $P|_{x'}$ is those tuples $(x_{k+1}, \dots, x_n)$ for which $\sum_{j=k+1}^{n} s_j x_j \leq S - \sum_{j=1}^{k} s_j x'_j$. Now, take partial solutions x, x' having $\mathtt{s}(x) = s, \mathtt{s}(x') = s'$. $\text{Sol}(P|_x) \supseteq \text{Sol}(P|_{x'})$ and therefore the larger the value of the state assigned to the root of the BDD, the fewer solutions it will contain, and these solutions will be contained in the BDD created with any smaller value. □

From Proposition 6, when we combine nodes u_1 and u_2 into some node w, we can let

$$s_{\text{new}} = \min\{s_1, s_2\},$$

because decreasing s_i can only introduce more solutions to the BDD $B_{w,t}$ as compared to $B_{u_i,t}$.

- Inverse Knapsack Problem (IKP)

  Let P be a IKP and let $\mathtt{s}$ be the state for the IKP from Section 5.4.2. Much like for the KP, we can let

  $$s_{\text{new}} = \max\{s_1, s_2\}.$$

  We replace the min with a max because in this case, increasing s_i can only introduced more solutions.

- WMISP

  Let P be a WMISP and let $\mathtt{s}$ be the state for the WMISP from Section 5.4.2.

Proposition 7 *Let*

$$B = \text{build_exact}(\{x_{k+1}, \ldots, x_n\}, s, \texttt{update})$$

and

$$B' = \text{build_exact}(\{x_{k+1}, \ldots, x_n\}, s', \texttt{update}),$$

with $s \subseteq s'$. *Then* $Sol(B) \subseteq Sol(B')$

The proof of Proposition 7 can be found in previous chapters and is based on the idea that if we include more vertices in the state of a node, more vertices will be eligible to be added to the independent sets under this node, thereby increase the number of solutions in the BDD.

Therefore, we can use

$$s_{\text{new}} = s_1 \cup s_2$$

as the relaxation operation.

As the WMCP and the SPP can both be formulated as a WMISP, this relaxation function also works for these problems.

- SCP

  The compilation of relaxed BDD for the SCP is discussed above. In relies on the following proposition:

  Proposition 8 *Let*

  $$B = \text{build_exact}(\{x_{k+1}, \ldots, x_n\}, s, \texttt{update})$$

  and

  $$B' = \text{build_exact}(\{x_{k+1}, \ldots, x_n\}, s', \texttt{update})$$

  with $s \subseteq s'$*. Then* $Sol(B) \supseteq Sol(B')$

  The proof is based on the interpretation of the state of a solution as the set of constraints that still need to have some variable set to 1. If we eliminate constraints from this state, we can only increase the set of solutions in the BDD because the condition of feasibility becomes relaxed.

- IP

  Developing a relaxation operation for a general IP can be difficult. Assuming $a_{i,j} \geq 0$ one can let the relaxation operation take the coordinate-wise minimum because each constraint can be seen as a knapsack constraint.

5.5.2 Restricted BDDs

A BDD B is a width-W restricted BDD for P if $\omega(B) \leq W$ and $\text{Sol}(B) \subseteq \text{Sol}(P)$. Such a structure has size $|B| \leq nW$ and so enforcing a maximum width of W controls the size of the BDD.

Much like for relaxed BDDs, a restricted B can be used to provide a lower-bound on the objective function. As with exact BDDs, a solution in $\text{Sol}(B)$ maximizing an additively separable objective function can be found via a simple longest path calculation. As B is a restricted BDD, every solution will be feasible and so the longest path will return the best feasible solution in the restricted BDD.

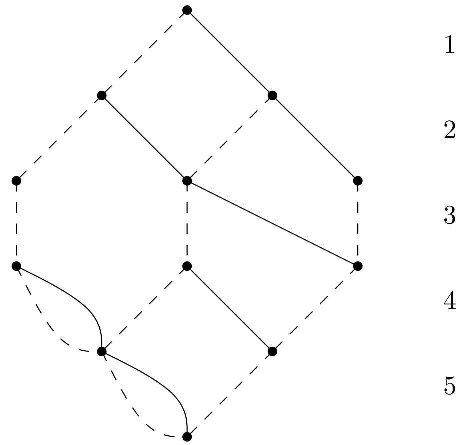


Figure 5.7: Width-3 Restricted BDD for the BOP in Example 16.

Algorithm 11 Node deletion for obtaining a restricted BDD.

Insert immediately after line 5 of Algorithm 8.

1: **while** $|L_j| > W$ **do**
2: $\quad u' :=$ node_select(L_j)
3: $\quad$ delete(u')

Example 16 Consider again the KP from Example 10. Depicted in Figure 5.7 is a width-3 restricted BDD. Each of the 13 solution represented by the BDD are feasible and the longest path (corresponding to (0,1,1,0,0)) has value 8, a lower bound on the optimal value. □

There are many ways to generate a restricted BDD. One can do a modification as in Section 5.5.1 and merge nodes whenever the width exceeds a given threshold by defining a *restriction operation*. However, a simpler modification works as well. During preliminary computational testing it was identified by the authors that the following modification to the exact BDD compilation algorithm produces higher quality solutions than defining a restriction operation, and so we present this method here.

Algorithm 11 can be used to generate a restricted BDD. It proceeds as in Algorithm 8, and during the construction, whenever a layer exceeds the preset maximum width, it selects some node in that layer and deletes it from the BDD. Such an operation only removes solutions and hence creates a BDD representing only feasible solutions.

Algorithm 12 delete(u')

1: **for all** $u \in L_{j-1}$ with arc directed at u' ($b_0(u)$ or $b_1(u) = u'$) **do**
2:     **if** $d^+(u) = 1$ **then**
3:         delete(u)
4: $L_j = L_j \setminus \{u'\}$

The operation that selects a node $u' \in L_j$ to delete can have a dramatic effect on the quality of the restricted BDD that is generated. In particular, even a width-1 BDD will contain an optimal solution if the correct nodes to delete are always selected. The authors have experimented with various methods and find that selecting the node u' with the worst objective function value (i.e., $z(u')$ is smallest amongst all nodes in L_j) yields restricted BDD with high quality solutions (as compared with other heuristics tested).

Let the function build_restriction($X, s,$ `update`) be Algorithm 8 supplemented with Algorithm 11.

5.6 Branching via BDDs

5.6.1 Partitioning the Solution Space

Let P be a BOP with $z = \sum_{j=1}^{n} c_j(x_j)$ (we suppose the objective is to maximize z) and $B = (U, A)$ be a relaxed BDD for P.

We begin by defining when a node in B is *exact* versus when it is *relaxed*. An node u is an exact node if for any two partial solutions $x^1, x^2 \in \text{Sol}(B_{r,u}), F(x^1) = F(x^2)$ and these sets are not empty (i.e., x^1 and x^2 can be completed to feasible solutions). A node is relaxed otherwise.

A set of nodes $C \subseteq U$ is an *exact* $r - t$ *cut* if there is no path in B from r to t that doesn't use a node from C and all nodes in C are exact.

Example 17 Consider the KP from Example 10. Figure 5.8 depicts a width-3 relaxed BDD for the problem. The nodes are named r, t or an associated integer and in parenthesis next to the name is a label indicating whether the node is exact or relaxed for the KP.

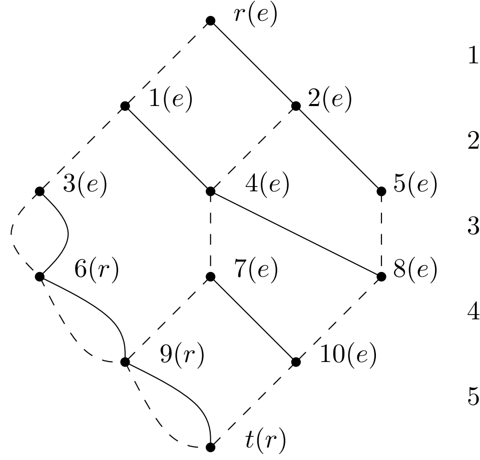


Figure 5.8: Width-3 Relaxed BDD for the BOP in Example 17.

Consider, for example, node 6. The two partial solutions ending at node 6 are (0,0,0) and (0,0,1). The completion (1,1) (i.e., assigning $x_4 = x_5 = 1$) is feasible for (0,0,0) but infeasible for (0,0,1). Therefore, 6 is relaxed. In contrast, consider node 8. The three partial solutions, (0,1,1), (1,0,1), (1,1,0), can all only be completed into feasible solutions by setting $x_4 = x_5 = 0$ thereby making 8 an exact node. □

Theorem 10 *Let C be an exact $r - t$ cut for B. Then,*

$$z^*(P) = \max_{u \in C}\{z(u) + z^*(P|_u)\}$$

Proof. Proof We begin by showing $z^*(P) \leq \max_{u \in C}\{z(u) + z^*(P|_u)\}$.

Let $x^*(P)$ be an optimal solution to P. As B is a relaxed BDD, $x \in \text{Sol}(B)$ so there exists some $r - t$ path p in B corresponding to $x^*(P)$. Since C is an $r - t$ cut, p goes through some node $u' \in C$. Let $x^*(P) = (x^1, x^2)$, where x^1 is $x^*(P)$ on variables $x_1, \ldots, x_{\ell(u)-1}$ and x^2 is $x^*(P)$ on the remaining variables.

We first claim that $z(u') = \sum_{j=1}^{\ell(u')-1} c_j(x_j^1)$. If not, then there is some other partial solution x' in $\text{Sol}(B_{r,u'})$ with $z(u') = \sum_{j=1}^{\ell(u')-1} c_j(x') > \sum_{j=1}^{\ell(u')-1} c_j(x_j^1)$. But, since u' is exact, $F(x') = F(x^1)$, making the solution $\tilde{x} = (x', x^2)$ feasible, and $\sum_{j=1}^{n} \tilde{x}_j > \sum_{j=1}^{n} x^*(P)_j$, contradicting that $x^*(P)$ is an optimal solution.

In addition, since u' is an exact node, the feasible set to $P|_{u'}$ coincides exactly with $F(x)$ for every $x \in \text{Sol}(B_{r,u'})$ and in particular to $F\left(x^1\right)$. Therefore, x^2 must be optimal to $P|_{u'}$, making $z^*(P|_{u'}) = \sum_{j=\ell(u)}^{n} x_j^2$.

Therefore,

$$z^*(P) = \sum_{j=1}^{n} x^*(P)_j = z(u') + z^*(P|_{u'}) \leq \max_{u \in C}\{z(u) + z^*(P|_u)\},$$

as desired.

We now show the other side of the inequality. Suppose, by way of contradiction, that

$$z^*(P) < z(u') + z^*(P|_{u'})$$

for some $u' \in C$. Let x' be the partial solution in $\text{Sol}(B_{r,u})$ achieving $z(u')$; i.e., $\sum_{j=1}^{\ell(u)-1} x'_j = z(u')$. As u is exact, x' can be completed into a feasible solution, and the completion x'' of x' that maximizes z is the optimal solution to $P|_{u'}$. Then, the feasible solution $x = (x', x'')$ satisfies that $z(u') + z^*(P|_{u'}) = \sum_{j=1}^{n} c_j(x_j)$, contradicting that $z^*(P)$ is the optimal solution to P. □

Example 18 Consider again the relaxed BDD in Figure 5.8. Consider the cut $C = \{1, 2\}$. For node $1, z(1) = 0$ and the best possible completion with x_1 fixed to 0 is (0,1,0,1). Therefore, for this node, $z(1) + z^*(P|_1) = 0 + 9 = 9$. For node $2, z(2) = 2$ because the one partial solution ending at 2 is (1) and $c_1 = 2$ so that $c_2 \cdot 1 = 2$. In addition, the best possible completion with x_1 set to 1 is (0,1,0,0) which has objective function value 5. Therefore, for this node, $z(2) + z^*(P|_2) = 2 + 5 = 7$. The optimal value is $\max\{z(1) + z^*(P|_1), z(2) + z^*(P|_2)\} = 9$. □

Theorem 10 gives us a method of branching so as to create a complete search of the feasible set. Given a BOP P, we can build a relaxed BDD B and, for any exact $r - t$ cut $C = \{u_1, \ldots, u_k\}$, create subproblems $P|_{u_i}$. Solving each subproblem yields optimal values $z^*\left(P|_{u_1}\right), \ldots, z^*\left(P|_{u_k}\right)$. Adding the longest path lengths $z(u_1), \ldots, z(u_k)$ in the relaxed BDD to each of these values, and taking the maximum of the values $z(u_i) + z^*\left(P|_{u_i}\right)$ will therefore yield the optimal value for P.

One can also uncover an optimal solution. When creating the subproblems $P|_{u_k}$ we also save any partial solution $x'_k \in B_{r,u_k}$ that has objective function value $z(u_k)$. Let $x^*\left(P|_{u_k}\right)$

be the optimal solution for each of the subproblems. Then, for the k' that maximizes $z(u_{k'}) + z^* \left(P|_{u_{k'}}\right)$, the solution $\left(x'_{k'}, x^* \left(P|_{u_{k'}}\right)\right)$ will be an optimal solution for P.

5.6.2 Labeling Nodes as Exact/Relaxed

Identifying which nodes in a relaxed BDD B are exact is computationally difficult. However, adding the following to the top-down relaxed BDD construction algorithm will identify a subset of the nodes that are indeed exact. Such a procedure can then be used to identify an exact $r - t$ cut.

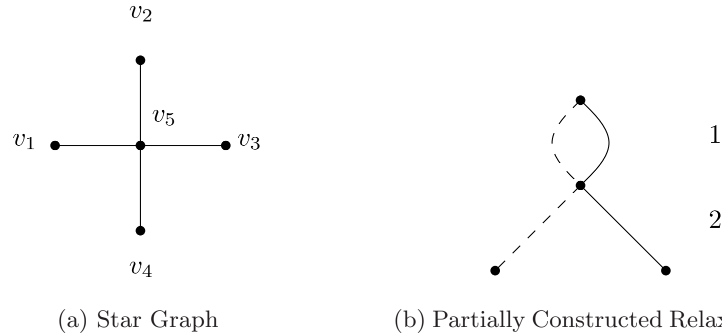
Initialize the root node r to have label $e(r) = 1$, where $e(u) = 1$ indicates that the node u is exact. Now, having constructed layers $L_1, \dots L_j$ and labeled nodes in L_j as exact or not, consider a node $u \in L_j$. If when considering appending the BDD with an arc and s_{new} doesn't exists in layer L_{j+1}, a new node u_{new} is created (as described in Algorithm 8) and set $e(u_{\text{new}}) = e(u)$. If otherwise s_{new} does exist at some node $u' \in L_{j+1}$, we set $e(u') = \min\{e(u'), e(u)\}$ (i.e., $e(u')$ will be 0 if either it is currently 0 or if $e(u)$ is 0). Lastly, whenever the modification in Algorithm 10 is employed due to a layer growing beyond the preset maximum width W, we set $e(u_1) = e(u_2) = 0$ for the two nodes chosen to be merged.

Theorem 11 *If $e(u) = 1$ then u is exact.*

Proof. Proof $e(u) = 1$ if any only if no node in $B_{r,u}$ is ever forcibly merged via the modification in Algorithm 10. Now, suppose we build an exact BDD via Algorithm 8 without the modification of Algorithm 10 and let B' be the exact BDD generated. There must be a node u' in B' with $\text{Sol}(B'_{r',u'}) = \text{Sol}(B_{r,u})$ because the same operations leading to u must occur in the generation of B' as well. As B' is an exact BDD, each node in B' is exact. Therefore, u must be exact. □

The converse of the statement in Theorem 11 is not necessary true as shown by the following example.

Example 19 Consider the partially constructed relaxed BDD in Figure 5.9b for the independent sets in the star graph in Figure 5.9a. The partial solutions $(0), (1)$ have different feasible completions. For example, the solution $(0, 0, 0, 0, 1)$, which corresponds to the set



(a) Star Graph (b) Partially Constructed Relaxed BDD

Figure 5.9: A star graph and a partially constructed relaxed BDD for the independent sets in B

$\{v_5\}$ is an independent set while $(1, 0, 0, 0, 1)$, corresponding to the set $\{v_1, v_5\}$, is not an independent set. Therefore, the single node in L_2 is a relaxed node and during the compilation algorithm it would have resulted from an instantiation of Algorithm 10. Therefore, the right-most node in L_3 would have been designated as relaxed, because there is an incoming arc directed from a relaxed node. However, the two solutions represented by paths to this node, $(0, 1)$ and $(1, 1)$, have the same set of feasible completions: x_3 and x_4 assigned arbitrarily, but $x_5 = 0$.

□

5.6.3 Selecting an Exact $r - t$ Cut

As described in Theorem 10 any exact $r - t$ cut suffices for partitioning the solution space. For example, as long as the maximum width of a relaxed BDD is greater than or equal to 2, we can use the second layer (i.e., $C = L_2$) which in effect creates subproblems defined by setting x_1 equal to 0 or 1. However, having nodes in deeper layers of the BDD creates subproblems defined by several value assignments to variables taken all at once. In addition, the deeper the nodes in the cut, the more likely that paths have merged above the nodes and therefore creating subproblems may remove symmetry as well.

We propose three methods for selecting cuts.

- **Traditional Branching**

  Here, we set $C = L_2$. As discussed above, this has the effect of branching on the designation of the variable associated with the first layer of the BDD. This is the shallowest cut possible but mimics traditional branching schemes.

- **Last-Exact-Layer**

  We define, for a *proper* relaxed BDD B (i.e., a BDD for which $\mathrm{Sol}(B) \supset \mathrm{Sol}(P)$), the *last-exact-layer* of B as the set of nodes $\mathrm{LEL}(B) = L_{j'}$, where j' is the maximum value of j for which each node in L_j is exact.

  Although identifying precisely which nodes in a relaxed BDD are exact can be hard, identifying $\mathrm{LEL}(B)$ can be done during the construction of a relaxed BDD, as proved in Theorem 12, if a complete state function (and associated update function) is known.

Theorem 12 *Let u be the first node for which $e(u) = 0$ and suppose Algorithm 8 is run with a complete state function. Then* $\mathrm{LEL}(B) = L_{\ell(u)-1}$.

Proof. Proof We first show that if L_j contains a relaxed node then L_{j+1} must also contain a relaxed node. Let u be a relaxed node in L_j. Then, either there exists a solution $x' \in \mathrm{Sol}(B_{r,u})$ which has no feasible completion or there exists solutions $x^1, x^2 \in \mathrm{Sol}(B_{r,u})$ for which $F(x^1) \neq F(x^2)$.

In the former case, for any arc a directed out of u, (x', d_a) will also have no feasible completion.

For the latter case, suppose, without loss of generality, that there exists a $y \in F(x^1)$ which is not in $F(x^2)$. Let $\tilde{x}^1 = (x^1, d)$, where d is the value assigned to variable $x_{\ell(u)}$ in y. Since B is a relaxed BDD, there must be a node $u' \in L_{j+1}$ with $\tilde{x}' \in \mathrm{Sol}(B_{r,u'})$. Consider $\tilde{x}^2 = (x^2, d)$. This solution is also in $\mathrm{Sol}(B_{r,u'})$. Then, u' must be relaxed, because, since $y \notin F(x^2)$, the partial solution y', which is obtained by removing the value assigned to variable x_j in y, will not be in $F(\tilde{x}^2)$, although it is in $F(\tilde{x}^1)$.

We need only show that if we use a complete state function in Algorithm 8, the first node u to have $e(u) = 0$ must be relaxed. The first node designated relaxed must

be done so because of the instantiation of Algorithm 10. Since the state function is complete, we know that the two node u_1 and u_2 that are combined have solutions $x^1 \in \mathrm{Sol}(B_{r,u_1})$ and $x^2 \in \mathrm{Sol}(B_{r,u_2})$ for which $F(x^1) \neq F(x^2)$ (this may not be the case if the state function is only sound and not complete). When the nodes are merged into u' or $\hat{u}$, these two solutions will be a part of the solution space of the same node, and therefore u will be relaxed. □

Therefore, given a complete state function, we can easily identify what $\mathrm{LEL}(B)$ is. If only a sound state function is employed, we can still use the layer above the first node that is designated as an exact $r - t$ cut, but it may not be the actual last-exact-layer of B.

- **Transition-Cut**

  We define, for a *proper* relaxed BDD B, the *transition-cut* of B as the set of nodes

$$\mathrm{TC}(B) = \{u : e(u) = 1,\ \exists (u, u') \in A \text{ with } e(u') = 0\},$$

  where $e(u)$ is set as described in the beginning of Section 5.6.2. We show now that $\mathrm{TC}(B)$ is an exact $r - t$ cut.

  Theorem 13 *Let $B = (U, A)$ be a proper relaxed BDD.* $\mathrm{TC}(B)$ *is an exact $r - t$ cut.*

  Proof. Proof By the definition of a transition-cut, each node is exact. We need only show that each solution goes through some node in $\mathrm{TC}(B)$.

  Take any $x \in \mathrm{Sol}(B)$. The path p corresponding to this solution ends at t. Since B is a proper relaxation, $e(t) = 0$. Since $e(r) = 1$ there must be a first node $u \in p$ for which $e(u) = 0$. The node immediately preceding this node in p will be in $\mathrm{TC}(B)$, as desired. □

Example 20 Consider again the relaxed BDD in Figure 5.8. The three examples of cuts discussed above are:

$$\begin{aligned} L_2 &= \{1, 2\} \\ \mathrm{LEL}(B) = L_3 &= \{3, 4, 5\} \\ \mathrm{TC}(B) &= \{3, 7, 10\} \end{aligned}$$

□

5.7 Branch and Bound

5.7.1 The Algorithm

We now present the main algorithm of the chapter. It utilizes relaxed BDD for upper-bounds (assuming a maximization problem) and restricted BDDs for lower-bounds. In addition, the algorithm uses exact $r-t$ cuts in relaxed BDDs to create subproblems.

The algorithm iteratively search over subproblems, creating relaxed and restricted BDDs. Each subproblem is defined by a search tree node u that contains information necessary to build an exact (and hence relaxed or restricted) BDD of a subproblem that needs to be explored. This consists of a set of variables $X(u)$ in the subproblem, a state $s(u)$ which will be assigned as the state of the root, and a value $z(u)$ which is the largest value of the sum of a partial solution on the variables not in $X(u)$ that leads to this subproblem.

Starting with P, we create a search tree node u for the entire problem on variables $x_1, \ldots, x_n$. We initialize this node to have state s_0, the initial state necessary to create an exact BDD. In addition, the `update` function is supplied. We let this search node be u, the initial search tree node. In addition, as in typical branch-and-bound algorithms, the optimal value z_{opt} is initialized to $-\infty$. This search tree node is put in a queue of nodes to explore, Q.

Now, while there are search tree nodes left to explore, we select a node $u \in Q$ and remove u' from Q. We first build a restricted BDD B for the subproblem defined by the state $s(u)$ on variables $X(u)$. If the longest path value in B plus the value $z(u)$ is larger than the current value of z_{opt}, we update z_{opt} to equal this sum. We then delete B from memory.

Now, we build a relaxed BDD B, again for the subproblem defined by the state $s(u)$ on variables $X(u)$. This BDD supplies an upper-bound for the subproblem. If the upper-bound $z^*(B)$ plus the value $z(u)$ is less than or equal to z_{opt} we need not explore this subproblem further, and B is deleted without creating any further search tree nodes.

Otherwise, $z^*(B) + z(u) > z_{\text{opt}}$ and we need to explore this subproblem further. From

Theorem 10 we know that any exact $r - t$ cut in B will allow us to generate subproblems that will explore the entire feasible set remaining in this subproblem. We can do this by any method, or specifically via the labeling e described in Section 5.6.2. From the cut chosen, C, we create a search tree node $u(w)$ for each $w \in C$. The state of this node $s(u(w))$ is set to the state of w. By Lemma 8, this state is the initial state necessary to generate an exact BDD for $P|_w$ and so also a relaxed or restricted BDD. The variables in $X(u(w))$ are the variables in B that are below the layer of node u, which are the variables in $P|_{u(w)}$. Finally, $z(u(w))$ is the value of the partial solution with the best possible objective value that when set yields subproblem $P|_{u(w)}$.

The algorithm follows a standard branch-and-bound scheme. The primal heuristic employed is restricted BDDs and relaxed BDD provide relaxation bounds. The two BDDs need exactly the same information as input; namely the longest path value to the node that creates the subproblem, the set of variables remaining to explore, and the initial state for this subproblem. Search nodes are pruned based on objective function values in the subproblems.

We note here that, as presented, the algorithm only returns on optimal value. A simple modification allows one to recover the optimal solution as well. Namely, we save an additional piece of information at each search tree node u. This information will be the values assigned to the variables that yield a partial solution of value $z(u)$ in $X \setminus X(u)$. This can be determined and saved when creating each search tree node.

5.7.2 Comparison with Traditional Branch-and-Bound Algorithms

There are many algorithms designed to solve BOPs. Many of these rely on a variant of the following branch-and-bound scheme. Begin by solving a relaxation for the original problem P, and then select a variable x_k to create subproblems $P|_{x_k=0}, P|_{x_k=1}$. As

$$z^*(P) = \max\{c_1(0) + z^*(P|_{x_k=0}), c_2(1) + z^*(P|_{x_k=1})\},$$

the algorithm proceed by recursively solving each subproblem, yielding a complete algorithm.

We highlight the major differences, contributions, and benefits of the BDD-based branch-

Algorithm 13 solve(P, X, s_0, `update`)

1: create search node u'

2: $X(u') = X$

3: $z(u') = 0$

4: $s(u') = s_0$

5: $Q = \{u'\}$

6: $z_{\text{opt}} = -\infty$

7: **while** $Q \neq \emptyset$ **do**

8: $u = \text{select_node}(Q)$

9: $Q = Q \setminus \{u\}$

10: $B = \text{build_restriction}(X(u), s(u))$

11: **if** $(z(u) + z^*(B) > z_{\text{opt}})$ **then**

12: $z_{\text{opt}} = z(u) + z^*(B)$

13: delete B

14: $B = \text{build_relaxation}(X(u), s(u))$

15: **if** $(z(u) + z^*(B) > z_{\text{opt}})$ **then**

16: let C be an exact $r - t$ cut in B

17: **for all** $w \in C$ **do**

18: create search node $u(w)$

19: $X(u(w)) = X(u) \setminus \{x_j : j = |X| - |X(u)| + 1, \ldots, |X| - |X(u)| + \ell(w)\}$

20: $z(u(w)) = z(u) + z(w)$

21: $s(u(w)) = s(w)$

22: $Q = Q \cup \{u'\}$

23: delete B

24: return z_{opt}

and-bound algorithm.

1. Relaxed and restricted BDDs are a novel technique for providing bounds to BOPs. They can be used as a standalone bounding method or be employed in conjunction with other relaxations and heuristics. The only information necessary (in addition to

a sound state function and `update` function) to create a relaxed of restricted BDD for a subproblem is either a state or a partial solution that yields that subproblem. In addition, the quality of the relaxation (heuristic) at each search tree node is flexible in that the maximum width determines how tight the approximation generated will be.

2. The BDD-based algorithm requires little memory consumption. The amount of information stored at each search tree node is minimal, and at most one BDD is saved at any point during the execution of the algorithm.

3. Search tree nodes are created, in general, based on several value assignments to variables, all taken at once. This is because any exact $r - t$ cut can be used to create subproblems and ensure that the entire feasible region is explored. The deeper in the BDD this cut is taken, the more variables are assigned.

4. The subproblems generated are based on potentially a set of solutions, not just one. When a node u in a relaxed BDD is part of an exact $r - t$ cut that is used to create subproblems, there may be many solutions from the root of the relaxed BDD to u. In a standard branching scheme, each of these partial solutions could potentially lead to different search tree nodes.

5.8 Computational Results

In this section we present results comparing the run time and solution quality of the BDD-based branch-and-bound algorithm with a leading general-purpose solver for BOPs, CPLEX. We apply the technique to the MISP and run the comparison on complements of the graphs in the well-known DIMACS benchmark set (http://cs.hbg.psu.edu/txn131/clique.html) for the Maximum Clique Problem.

The tests were run on an Intex Xeon E5345 with 8GB RAM. The BDD-based algorithm was implemented in C++ and CPLEX 12.4 was used. The settings used in CPLEX were set to default.

Many inputs to the BDD-based algorithm are necessary. There include the frequency of heuristics, the order of the variables used, the exact $r - t$ cut used, the selection of

subproblems, and the maximum width allowed for the relaxed and restricted BDDs. In our implementation, a width-100 restricted BDD was generated at each search tree node, to name a few. The order of the variables is given by the min-state ordering from the Chapter 2. We employed the TC(B) cut. Subproblems were selected based on the best relaxation bound known for each node. Namely, the node with the worst $z(u)$ value amongst the nodes in Q is always selected. The original BDD generated has width-100, and all subsequent BDDs have width-1. This mimics the approach often taken in IP; a reasonable amount of time is spent at the root node finding cutting planes and tightening the relaxation while less time is devoted to each subsequent search tree node.

We use the IP model for the MISP in Section 5.2 and note that there are different IP models for the MISP. For the purposes of computational testing we used this traditional model. Although other choices may yield different results, the computational results presented here and to give a basis for the proposed algorithm. The authors by no means claim that the BDD-based branch-and-bound algorithm outperforms state-of-the-art IP technology. We compare with IP (and in particular CPLEX) to test whether or not the proposed algorithm has an basis for further development and if it has the capability to solve problems of at least similar difficulty to other general-purpose methods for BOP. The results suggest this.

The results are given in Table 5.1 where both algorithm were given a maximum of half an hour. We begin with the name of the instances and the number of vertices. We then, for both methods, give the time (in seconds), the lower-bound, upper-bound, and the percent gap ($\frac{\mathrm{UB}-\mathrm{LB}}{\mathrm{LB}} \cdot 100\%$) proved in the time given.

The algorithms were tested on 66 instances. 33 instances were solved by the BDD-based method, as opposed to 30 instances by the IP-method. For 28 instances, the gap provided by the BDD-based algorithm was better than the IP gap, as opposed to only 14 instances where IP-based bound provided a tighter gap. In addition, a statistical test for the differences in the means yields a p-value of 0.03 that the percent gap provided by the BDD algorithm is stronger than the percent gap provided by CPLEX.

In addition, we provide two plots of the data points. In Figure 5.10 we depict the number of instances solved, per method, versus time. We see that at any given point in time, more

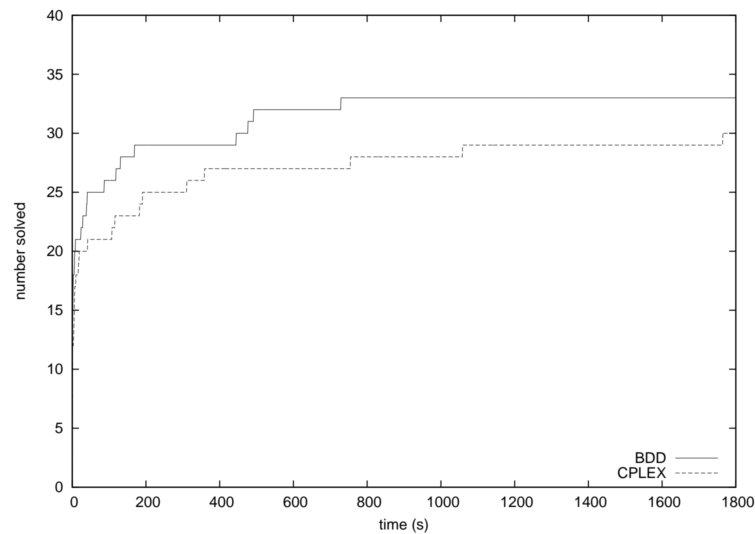


Figure 5.10: Number of instances solved versus time.

instances are solved by the BDD-based algorithm than CPLEX. Furthermore, in Figure 5.11 we depict the percent gap, after the allotted 1800 seconds, for both methods. From this plot we can deduce that the BDD-based algorithm for the MISP on this data set is more robust than the IP-based method used.

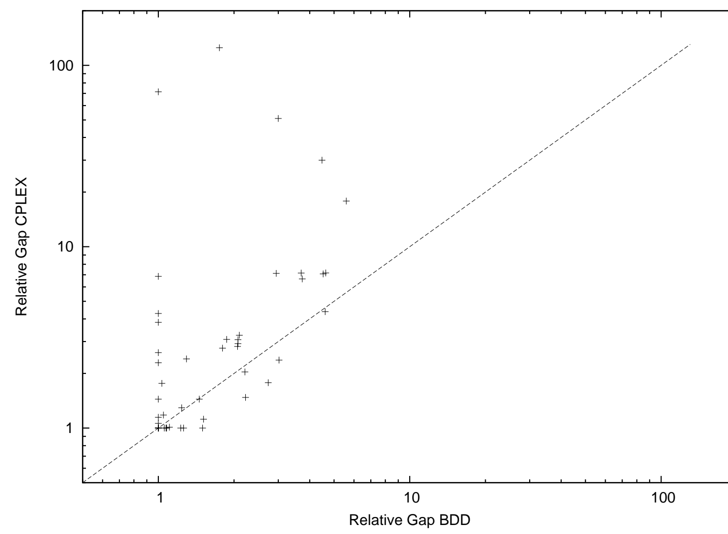


Figure 5.11: Gap comparison.

Table 5.1: Comparison of BDD-Based branch-and-bound with IP solver CPLEX

Instance		BDD				IP			
name	n	t_{BDD}	LB_{BDD}	UB_{BDD}	$\%\text{Gap}_{\text{BDD}}$	t_{IP}	LB_{IP}	UB_{IP}	$\%\text{Gap}_{\text{IP}}$
brock200_1.clq	200	1800	21	22	4.76	1800	21	24.78	18.00
brock200_2.clq	200	6.24	12	12	0.00	754.06	12	12.00	0.00
brock200_3.clq	200	38.36	15	15	0.00	1058.24	15	15.00	0.00
brock200_4.clq	200	118.92	17	17	0.00	1800	17	18.06	6.22
brock400_1.clq	400	1800	27	56	107.41	1800	22	64.20	191.83
brock400_2.clq	400	1800	27	56	107.41	1800	21	64.49	207.12
brock400_3.clq	400	1800	30	62	106.67	1800	23	64.87	182.05
brock400_4.clq	400	1800	31	58	87.10	1800	21	64.72	208.17
brock800_1.clq	800	1800	19	86	352.63	1800	13	92.00	607.69
brock800_2.clq	800	1800	20	74	270.00	1800	13	93.00	615.38
brock800_3.clq	800	1800	19	71	273.68	1800	14	93.00	564.29
brock800_4.clq	800	1800	19	88	363.16	1800	13	93.10	616.19
c-fat200-1.clq	200	0.01	12	12	0.00	16.17	12	12.00	0.00
c-fat200-2.clq	200	0.01	24	24	0.00	6.93	24	24.00	0.00
c-fat200-5.clq	200	0.01	58	58	0.00	41.6	58	58.00	0.00
c-fat500-10.clq	500	0.04	126	126	0.00	107.77	126	126.00	0.00
c-fat500-1.clq	500	0.04	14	14	0.00	310.38	14	14.00	0.00
c-fat500-2.clq	500	0.04	26	26	0.00	358.31	26	26.00	0.00
c-fat500-5.clq	500	0.05	64	64	0.00	190.9	64	64.00	0.00
hamming10-2.clq	1024	1800	512	541	5.66	0.06	512	512.00	0.00
hamming10-4.clq	1024	1800	38	104	173.68	1800	27	48.00	77.78
hamming6-2.clq	64	0.25	32	32	0.00	0	32	32.00	0.00
hamming6-4.clq	64	0	4	4	0.00	0.5	4	4.00	0.00
hamming8-2.clq	256	476.3	128	128	0.00	0	128	128.00	0.00
hamming8-4.clq	256	130.55	16	16	0.00	5.25	16	16.00	0.00

Continued on next page

Instance		BDD				IP			
name	n	t_{BDD}	LB_{BDD}	UB_{BDD}	$\%\text{Gap}_{\text{BDD}}$	t_{IP}	LB_{IP}	UB_{IP}	$\%\text{Gap}_{\text{IP}}$
johnson16-2-4.clq	120	0.35	8	8	0.00	0.08	8	8.00	0.00
johnson32-2-4.clq	496	1800	16	24	50.00	4.01	16	16.00	0.00
johnson8-2-4.clq	28	0	4	4	0.00	0	4	4.00	0.00
johnson8-4-4.clq	70	0.2	14	14	0.00	0.02	14	14.00	0.00
keller4.clq	171	28.27	11	11	0.00	9.45	11	11.00	0.00
keller5.clq	776	1800	27	60	122.22	1800	21	31.00	47.62
keller6.clq	3361	1800	57	171	200.00	1800	33	1680.50	4992.42
MANN_a27.clq	378	1800	126	136	7.94	5.15	126	126.00	0.00
MANN_a45.clq	1035	1800	342	367	7.31	182.42	345	345.00	0.00
MANN_a81.clq	3321	1800	1097	1215	10.76	1800	1100	1112.50	1.14
MANN_a9.clq	45	0.01	16	16	0.00	0.03	16	16.00	0.00
p_hat1000-1.clq	1000	728.8	10	10	0.00	1800	7	500.00	7042.86
p_hat1000-2.clq	1000	1800	36	106	194.44	1800	30	213.89	612.95
p_hat1000-3.clq	1000	1800	46	212	360.87	1800	48	210.49	338.51
p_hat1500-1.clq	1500	1800	12	21	75.00	1800	6	750.00	12400.00
p_hat1500-2.clq	1500	1800	44	197	347.73	1800	25	750.00	2900.00
p_hat1500-3.clq	1500	1800	52	291	459.62	1800	42	750.00	1685.71
p_hat300-1.clq	300	1.56	8	8	0.00	1799.93	7	16.03	129.02
p_hat300-2.clq	300	168.25	25	25	0.00	1764.15	25	25.00	0.00
p_hat300-3.clq	300	1800	33	48	45.45	1800	33	47.65	44.39
p_hat500-1.clq	500	23.5	9	9	0.00	1800	6	25.71	328.51
p_hat500-2.clq	500	1800	34	44	29.41	1800	23	55.34	140.62
p_hat500-3.clq	500	1800	43	95	120.93	1800	42	85.61	103.83
p_hat700-1.clq	700	86.81	11	11	0.00	1800	5	34.32	586.35
p_hat700-2.clq	700	1800	40	72	80.00	1800	27	74.49	175.90
p_hat700-3.clq	70	1800	47	142	202.13	1800	48	113.84	137.16
san1000.clq	1000	40.59	15	15	0.00	1800	7	18.23	160.48

Continued on next page

Instance		BDD				IP			
name	n	t_{BDD}	$\mathrm{LB}_{\mathrm{BDD}}$	$\mathrm{UB}_{\mathrm{BDD}}$	$\%\mathrm{Gap}_{\mathrm{BDD}}$	t_{IP}	$\mathrm{LB}_{\mathrm{IP}}$	$\mathrm{UB}_{\mathrm{IP}}$	$\%\mathrm{Gap}_{\mathrm{IP}}$
san200_0.7_1.clq	200	1.82	30	30	0.00	0.9	30	30.00	0.00
san200_0.7_2.clq	200	1.65	18	18	0.00	18.05	18	18.00	0.00
san200_0.9_1.clq	200	2.5	70	70	0.00	0.13	70	70.00	0.00
san200_0.9_2.clq	200	2.7	60	60	0.00	0.2	60	60.00	0.00
san200_0.9_3.clq	200	1800	44	54	22.73	0.98	44	44.00	0.00
san400_0.5_1.clq	400	5.28	13	13	0.00	1800	9	13.00	44.44
san400_0.7_1.clq	400	8.21	40	40	0.00	115.99	40	40.00	0.00
san400_0.7_2.clq	400	1800	30	31	3.33	1800	17	30.00	76.47
san400_0.7_3.clq	400	1800	21	26	23.81	1800	17	22.00	29.41
san400_0.9_1.clq	400	1800	100	126	26.00	3.25	100	100.00	0.00
sanr200_0.7.clq	200	444.27	18	18	0.00	1800	18	20.65	14.71
sanr200_0.9.clq	200	1800	39	59	51.28	1800	41	45.87	11.87
sanr400_0.5.clq	400	491.74	13	13	0.00	1800	10	38.28	282.78
sanr400_0.7.clq	400	1800	20	42	110.00	1800	18	58.47	224.85

5.9 Conclusions and Future Work

In this chapter we present a novel branch-and-bound algorithm for binary optimization problems (BOPs) that utilizes only binary decision diagrams (BDDs). The BDDs are used for generating both relaxation bounds and heuristic solutions. In addition, the relaxed BDDs also guide the branching. This technique allows for making branching decisions on pools of partial solutions, as opposed to branching on single value assignments to variables, as is typically done in algorithms designed to solve BOPs.

Computational results suggest to that proposed algorithm is well-suited for the maximum independent set problem as the algorithm is competitive with state-of-the-art integer programming software on a standard benchmark set for this problem class.

Many aspects of the algorithm needs to be investigated, as with any general purpose branch-and-bound algorithm. This includes looking at bettering the techniques for using approximate BDDs for bounds, and also looking into how to use the BDD to guide the search.

Chapter 6

Finite-Domain Cuts for Graph Coloring

6.1 Introduction

In integer programming models, a choice from several alternatives is typically encoded by a set of binary variables. For example, the job assigned to a particular worker might be represented by 0-1 variables y_{ij}, where $\sum_j y_{ij} = 1$ for each worker i, and $y_{ij} = 1$ indicates that job j is assigned to worker i. Valid inequalities can then be generated in terms of the 0-1 variables, so as to strengthen the continuous relaxation of the model.

An alternative approach is to formulate such a choice directly in terms of finite-domain variables. For example, variable x_i might indicate which job is assigned to worker i. The value of x_i need not be a number, but if we choose to denote jobs by numbers, we can analyze the convex hull of feasible solutions and write valid inequalities in terms of the variables x_i. These inequalities can then be mapped into a 0-1 model of the problem using a simple change of variable. The resulting 0-1 inequalities may be different from and more effective than known cutting planes for the 0-1 model.

This chapter explores the idea of using a finite-domain formulation of a problem as a source of new valid inequalities for the 0-1 model. We will refer to such inequalities as

finite-domain cuts. We apply the idea to the vertex coloring problem on graphs, which has a natural finite-domain formulation in terms of *all-different* constraints. Such "global" constraints frequently appear in constraint programming models, where finite-domain variables are often used rather than 0-1 variables to encode discrete choices.

We employ a common strategy for generating problem-specific cuts: the identification of facet-defining cuts for special types of induced subgraphs, such as odd holes, webs, and paths. We identify cuts that bound the objective function (which we call *z-cuts*) as well as cuts that exclude infeasible solutions (*x-cuts*).

We find that for coloring problems, finite-domain cuts for several subgraph structures (when mapped into 0-1 space) provide tighter bounds than known 0-1 cuts for those subgraphs. Furthermore, we identify more general structures for which finite-domain cuts are substantially more effective than known cuts, or for which no known cuts exist.

We present here our results for webs, odd cycles, paths, and intersecting systems, because they illustrate four possible outcomes:

- Finite-domain *comb cuts*, which differ from previously identified comb inequalities, that exist depending on the size of the domain set.
- Finite-domain *web cuts*, when mapped into the 0-1 model, yield tighter bounds than standard web cuts. This means, in particular, that if an existing algorithm identifies separating web cuts, we can replace them with more effective finite-domain web cuts at no additional computational cost.
- *Odd cycles* are a generalization of odd holes. We show that in the special case of odd holes, finite-domain cuts provide tighter bounds than standard odd hole and clique cuts. We can therefore replace known separating odd hole cuts with more effective cuts, at no additional cost. In the general case of odd cycles, only two finite-domain cuts for a given cycle provide a substantially tighter bound than hundreds or thousands of odd hole and clique cuts that can be generated for that cycle. We provide a polynomial-time algorithm that identifies all separating finite-domain cuts for a given odd cycle.
- By contrast, finite-domain *path cuts* do not improve existing bounds. When mapped into 0-1 space, they have no effect on the bound provided by the standard 0-1 model.

- *Intersecting systems* illustrate how a finite-domain perspective can yield facet-defining cuts for novel structures. To our knowledge, no 0-1 cuts have previously been identified for this general class of subgraphs. We also present a polynomial-time separation algorithm.

Mapping finite-domain cuts into 0-1 space has the advantage that finite-domain cuts can be combined with standard 0-1 constraints as well as previously known families of 0-1 cuts. However, bounds can also be obtained directly from the finite-domain model by solving its relaxation, which is much smaller than the 0-1 model. We investigate both approaches computationally.

We begin below with a problem statement and brief literature review. We then describe the mapping of finite-domain cuts into 0-1 space and prove some of its elementary properties. We next describe some general properties of the finite-domain polytope and then derive facet-defining inequalities for combs, odd cycles, webs, paths, and intersecting systems, and study their properties when mapped into 0-1 space. In particular, we show that a family of facet-defining x-cuts gives rise to a family of facet-defining z-cuts in a canonical way, a result that is crucial for obtaining good bounds. A section on computational results compares the strength of finite-domain cuts and known 0-1 cuts on odd cycles and webs. It also demonstrates the advantages of odd cycle cuts on a set of benchmark instances. The chapter concludes with a summary and suggestions for future research.

6.2 The Problem

Given an undirected graph G with vertex set V and edge set E, the vertex coloring problem is to assign a color x_i to each vertex $i \in V$ so that $x_i \neq x_j$ for each $(i, j) \in E$. We seek a solution with the minimum number of colors; that is, a solution that minimizes $|\{x_i \mid i \in V\}|$.

The vertex coloring problem can be formulated as a system of all-different constraints. An all-different constraint alldiff(X) requires that the variables in set X take pairwise distinct values. Let $\{V_k \mid k \in K\}$ be the vertex sets of the maximal cliques of G, and let X_k be the set of variables x_i with $i \in V_k$. Let the colors be denoted by distinct nonnegative numbers v_j for $j \in J$, so that each variable x_i has the finite-domain $D = \{v_j \mid j \in J\}$. Then

the problem of minimizing the number of colors is

$$
\begin{aligned}
& \min \ z \\
& z \geq x_i, \ i \in V \\
& \text{alldiff}(X_k), \ k = 1, \dots, K \\
& x_i \in D = \{v_j \mid j \in J\}, \ i \in V
\end{aligned} \tag{6.1}
$$

Here we use maximal cliques V_k, but any clique cover $\{V_k \mid k \in K\}$ suffices to formulate the coloring problem.

It is convenient assume that $|V| = n$ colors $v_0, \dots, v_{n-1}$ are available. We also assume $v_0 < \cdots < v_{n-1}$. An initial question is how to select numerical domain values $v_0, \dots, v_n$, and how polyhedral structure depends on the selection. We note that this same question arises in 0-1 programming, because the numerical domain of a Boolean variable need not be $\{0, 1\}$. In the Boolean case, polyhedral results are valid for any binary domain, modulo appropriate adjustments in the coefficients and right-hand sides of valid inequalities. The issue is more complicated for general finite-domains, but we find that the x-cuts identified here are valid for arbitrary nonnegative domain values, while z-cuts are valid for any domain of the form $D_\delta = \{0, \delta, 2\delta \dots, (n-1)\delta\}$, where $\delta > 0$. In practice, it is convenient to use domain D_1, because in this case the minimum color number z is one less than the chromatic number.

A standard 0-1 model for the coloring problem uses binary variables y_{ij} to denote whether vertex i receives color j, and binary variables w_j that indicate whether color j is used. The model is

$$
\begin{aligned}
& \min \sum_{j \in J} w_j \\
& \sum_{j \in J} y_{ij} = 1, \quad i \in V \qquad && (a) \\
& \sum_{i \in V_k} y_{ij} \leq w_j, \quad j \in J, \ k \in K \qquad && (b) \\
& y_{ij} \in \{0, 1\}, \quad i \in V, \ j \in J
\end{aligned} \tag{6.2}
$$

6.3 Previous Work

All facets for a single all-different constraint alldiff(X) are given in [40, 69]. If $X = \{x_1, \ldots, x_m\}$ and each x_i has domain $\{v_1, \ldots, v_m\}$ with $n \le m$, they are

$$\sum_{j=1}^{|J|} v_j \le \sum_{i \in J} x_i \le \sum_{j=m-|J|+1}^{m} v_j, \quad \text{all nonempty } J \subseteq \{1, \ldots, n\} \tag{6.3}$$

where again $v_1 < \cdots < v_m$. If $m = n$, (6.3) defines the affine hull when $J = \{1, \ldots, n\}$. The facial structure of a system of two all-different constraints is studied in [4, 5].

In [53] the dimension of all-different systems is considered and it was found that the polytope is full-dimensional if and only if the number of domain values exceeds the chromatic number. If the number of domain values equals the chromatic number, the polytope is not full-dimensional, and it's dimension depends on the structure of the constraints in the problem.

Facets for general all-different systems are derived for combs in [47, 48, 53] and for odd holes and webs in [52]. To our knowledge, the cuts we describe here for cycles, paths, and intersecting systems have not been previously identified. We also generalize the web cuts in [52] and introduce z-cuts for webs.

It is natural to ask when all facets of an all-different system are facets of individual constraints in the system. It is shown in [53] that this occurs if and only if the all-different system has an *inclusion* property, which means that pairwise intersections of sets V_k in the alldiff constraints are ordered by inclusion. The structures studied here lack the inclusion property and therefore generate new classes of facets.

Known facets for the 0-1 graph coloring model are discussed in [20, 56, 57, 59]. These include cuts based on odd holes, webs, anti-webs, cliques, and paths.

Finite-domain cuts have been developed for a few global constraints other than alldiff systems. These include the element constraint [40], the circuit constraint [27], the cardinality constraint [42], cardinality rules [70], the sum constraint [71], and disjunctive and cumulative constraints [42].

In a conference paper [11], we presented the cycle cuts described here and mapped them into 0-1 space. The present chapter extends the computational tests to benchmark instances, introduces additional families of cuts, and studies the properties of the mapping. Aside from

[11], the strategy of mapping finite-domain cuts into 0-1 space has, to our knowledge, not been previously investigated.

6.4 Mapping into 0-1 Space

We now specify what it means to map a valid finite-domain cut into 0-1 space. We first discuss cuts for (6.1) involving only the variables x_i, which we call *x-cuts*. We then consider bounds on the largest color number z, which we call *z-cuts*.

We convert a valid x-cut $ax \geq b$ to a 0-1 inequality simply by replacing each x_i with $\sum_j v_j y_{ij}$. The inequality $ax \geq b$ therefore becomes

$$\sum_{i=1}^{n} a_i \sum_{j=0}^{n-1} v_j y_{ij} \geq b \tag{6.4}$$

We refer to this as a *0-1 x-cut*. It is important to analyze this conversion carefully, to ensure that valid cuts are mapped to valid cuts and to study their strength in the 0-1 model.

The domain D^n of the coloring problem is the set of all tuples $(x_1, \ldots, x_n)$ with each $x_i \in D$. A bijection ϕ maps each $x \in D^n$ to a point $y = \phi(x)$ given by

$$y_{ij} = \begin{cases} 1 & \text{if } x_i = v_j \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

For $S \subset D^n$, let $\phi(S) = \{\phi(x) \mid x \in S\}$. All $y \in \phi(S)$ satisfy

$$\sum_{j=0}^{n-1} y_{ij} = 1, \quad i = 1, \ldots, n \tag{6.6}$$

An inequality $ax \geq b$ is *valid* for $S \subset D^n$ if $ax \geq b$ for all $x \in S$. The inequality (6.4) is valid for $\phi(S)$ if it is satisfied by all $y \in \phi(S)$. Valid cuts map to valid cuts:

Lemma 9 *If $ax \geq b$ is valid for $S \subset D^n$, then (6.4) is valid for $\phi(S)$.*

Proof. Supposing $\bar{y} \in \phi(S)$, we wish to show that $\bar{y}$ satisfies (6.4). Because $\bar{y} \in \phi(S)$, we have $\bar{y} = \phi(\bar{x})$ for some $\bar{x} \in S$. Thus $a\bar{x} \geq b$, which implies that $\bar{y}$ satisfies (6.4) because $\bar{x} = \sum_j v_j \bar{y}_{ij}$ from the definition of ϕ. □

An important issue is the strength of cuts mapped into 0-1 space. In particular, we may wish to know whether a 0-1 x-cut (6.4) is redundant of a system $Ay \geq c$ of known

0-1 cuts. To make this precise, we will say that (6.4) is *redundant* of system $Ay \geq c$ if all $y \in [0,1]^{n\times n}$ satisfying (6.6) and $Ay \geq c$ also satisfy (6.4). Cut (6.4) is simply *redundant* if all $y \in [0,1]^{n\times n}$ satisfying (6.6) also satisfy (6.4).

We now consider z-cuts, or bounds $z \geq ax + b$ on the largest color number z. If we suppose that the color numbers are $0, \ldots, n-1$, minimizing z is equivalent to minimizing the number of colors minus 1. Because the number of colors is $\sum_j w_j$, we map $z \geq ax + b$ to the 0-1 inequality

$$\sum_{j=0}^{n-1} w_j - 1 \geq \sum_{i=1}^{n} a_i \sum_{j=0}^{n-1} j\, y_{ij} + b \tag{6.7}$$

which we call a 0-1 z-cut. This inequality may be added to the 0-1 model because some optimal solution satisfies it. Yet (6.7) is not valid, because it can be violated by solutions that use larger color numbers but the same number of colors. Thus 0-1 z-cuts have the advantage of excluding symmetric solutions. We can ensure that (6.7) is formally valid by adding symmetry breaking constraints

$$w_j \geq w_{j+1},\ j = 0, \ldots, n-2 \tag{6.8}$$

to the 0-1 model (6.2).

We can now define validity as follows. Given $S \subset D_1^n$, we say that the z-cut $z \geq ax + b$ is valid for S if $\max_i\{x_i\} \geq ax + b$ for all $x \in S$. Inequality (6.7) is valid for $\phi(S)$ when it is satisfied by all $y \in \phi(S)$ and $w \in \{0,1\}^n$ that satisfy (6.6), (6.8), and

$$w_j \geq y_{ij}, \ \text{ all } i, j \tag{6.9}$$

Then valid bounds map to valid bounds:

Lemma 10 *If $z \geq ax + b$ is valid for $S \in D_1^n$, then (6.7) is valid for $\phi(S)$.*

Proof. Suppose that $\bar{y} \in \phi(S)$ and $\bar{w} \in \{0,1\}^n$ satisfy (6.6), (6.8) and (6.9). We wish to show that $(\bar{y}, \bar{w})$ satisfies (6.7). Because $\bar{y} = \phi(\bar{x})$ for some $\bar{x} \in S$, we have

$$\max_i\{\bar{x}_i\} \geq a\bar{x} + b = \sum_{i=1}^{n} a_i \sum_{j=0}^{n-1} j\, \bar{y}_{ij} + b$$

It therefore suffices to show that

$$\sum_{j=0}^{n-1} \bar{w}_j - 1 \geq \max_i\{\bar{x}_i\} \tag{6.10}$$

Due to (6.8), we can suppose $\bar{w}_j = 1$ for $j \leq k$ and $\bar{w}_j = 0$ for $j > k$. Then from (6.9) we have $\bar{y}_{ij} = 0$ for all i and all $j > k$. Thus (6.6) implies $\sum_{j=0}^{k} \bar{y}_{ij} = 1$ for all i, which implies

$$\sum_{j=0}^{k} j\bar{y}_{ij} \leq k, \quad \text{all } i \tag{6.11}$$

Now we have

$$\sum_{j=0}^{n-1} \bar{w}_j - 1 = (k+1) - 1 \geq \max_i \left\{ \sum_{j=0}^{k} j\bar{y}_{ij} \right\} = \max_i \left\{ \sum_{j=0}^{n-1} j\bar{y}_{ij} \right\} = \max_i \{\bar{x}_i\}$$

where the inequality is due to (6.11). This establishes (6.10), as desired. □

It is an interesting question whether facets map to facets. In general, they do not. Consider the feasible set S for the single constraint alldiff(x_1, x_2, x_3) with $x_i \in \{0, 1, 2\}$. Then $x_1 + x_2 \geq 1$ is one of the facet-defining inequalities (6.3). It maps to

$$y_{11} + 2y_{12} + y_{21} + 2y_{22} \geq 1 \tag{6.12}$$

This is not facet-defining because the convex hull of $\phi(S)$ has dimension 4, while only 2 points satisfy (6.12) at equality:

$$\begin{bmatrix} y_{10} & y_{11} & y_{12} \\ y_{20} & y_{21} & y_{22} \\ y_{30} & y_{31} & y_{32} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The finite-domain cuts we obtain below do not in general map to facet-defining cuts in 0-1 space. They can nonetheless provide substantially tighter bounds on the chromatic number than known cuts, which themselves may not be facet-defining, as in the case of odd hole cuts.

6.5 General Properties of the Polytope

In this section we describe two general properties of the finite-domain polytope in (6.1). The first proves that facet-defining inequalities appear in pairs and the second proves that for large enough domain values, all facet-defining inequalities contain coefficients with the same sign. This is demonstrated by the comb inequalities in Section 6.7.

For this section, we suppose that our domain set is D_1 but note that the result in Theorem 14 extends all-different problems with domain set D_δ.

6.5.1 Complement Inequalities

Given a point x, the *complement* of x , denoted by $C(x)$, is the n-dimensional vector $(n-1)e$, where e is the n-dimensional vector with 1 in each coordinate.

Lemma 11 *If $\tilde{x}$ is feasible then so is $C(\tilde{x})$.*

Proof. Let $\tilde{x}$ be a feasible solution. Then, $\tilde{x}_j \in D_1$. Hence $k - \tilde{x}_j \in D_1$.

Furthermore, consider any two variables x_j and $x_{j'}$. Since $\tilde{x}$ is feasible, if for some $k, x_j, x_{j'} \in X_k$ then $\tilde{x}_j \neq \tilde{x}_{j'}$. Therefore, $(n-1) - \tilde{x}_j \neq (n-1) - \tilde{x}_{j'}$ for any variables which share a constraint. □

Theorem 14 *Inequality $ax \geq \delta_1$ is facet-defining if and only if there exists a δ_2 such that $ax \leq \delta_2$ is facet defining.*

Proof. $(\rightarrow)$ Let $ax \geq \delta_1$ be facet defining. Take any point $\hat{x}$ satisfying $a\hat{x} = \delta_1$. Define $\delta_2 = aC(\hat{x})$.

Suppose $ax \leq \delta_2$ is not valid. Then there exits some feasible point $\tilde{x}$ such that $a\tilde{x} > \delta_2$. Hence

$$a\tilde{x} > \delta_2 = aC(\hat{x}) = a(n-1) - a\hat{x}.$$

Rearranging terms, we get

$$a\hat{x} > a(n-1) - a\tilde{x} = a((n-1) - \tilde{x}) = aC(\tilde{x}).$$

$\tilde{x}$ is feasible, therefore, by lemma 11, $C(\tilde{x})$ is feasible, so $aC(\tilde{x}) \geq \delta_1$. Hence,

$$a\hat{x} > aC(\tilde{x}) \geq \delta_1,$$

contradicting that $\hat{x}$ satisfies $ax \geq \delta_1$ at equality.

In order to finish the proof that $ax \leq \delta_2$ is facet defining, we display n affinely independent points, each satisfying $ax \leq \delta_2$ at equality. We know $ax \geq \delta_1$ is facet defining, so there exists n affinely independent points satisfying $ax \geq \delta_1$ at equality. Let $x^1, \ldots, x^n$ be these points. We show that their complements, $C(x^j)$, are n affinely independent points satisfying $aC(x^j) = \delta_2$.

For all $j, ax^j = \delta_1$ so that $aC(x^j) = a(n-1) - ax^j = \delta_2$.

Now, suppose $\sum_{j=1}^{n} \lambda_j C(x^j) = 0$ and $\sum_{j=1}^{n} \lambda_n = 0$. Then,

$$0 = \sum_{j=1}^{n} \lambda_j C(x^j) = \sum_{j=1}^{n} \lambda_j ((n-1) - x^j) = 0 - \sum_{j=1}^{n} \lambda_j x^j.$$

And, since $x^1, \ldots, x^n$ are affinely independent,

$$\sum_{j=1}^{n} \lambda_j x^j = 0 \text{ and } \sum_{j=1}^{n} \lambda_j = 0 \rightarrow \lambda_j = 0, \text{ for all j.}$$

($\leftarrow$) Dividing $ax \leq \delta_2$ by -1 produces $(-a)x \geq -\delta_2$. Applying this first part of the proof, we get that $(-a)x \leq \tilde{\delta_2}$ is facet defining for some $\tilde{\delta_2}$. And hence, setting $\delta_1 = -\tilde{\delta_2}$ we get that $ax \geq \delta_1$ is facet defining. $\square$

6.5.2 Signs of Coefficients in Facet-Defining Inequalities

In it interesting to consider what happens to the facial structure of the finite-domain polytope when we consider variations in D. In this, and the next, section, we will explore what happens to the polytope under changes to the domain set.

For this section, suppose we are given domain set $D^k = \{0, 1, \ldots, k\}$. We address the problem of characterizing the facet-defining inequalities as k varies.

Before providing the main result for this section, we first prove a lemma concerning optimal graph colorings which is used to prove the main result.

Lemma 12 *Let $G = (V, E)$ be a graph and consider the* Sum Coloring Problem*:*

$$\begin{aligned} &\min \ cx \\ &\mathit{alldiff}(X_k), \ k = 1, \ldots, K \\ &x_i \in D^k, \end{aligned} \tag{6.13}$$

where $c \geq 0$ and $D^k = \{0, 1, \ldots, k\}$.

There exists an optimal solution $\tilde{x}$ to the sum coloring problem (6.13) for which $\tilde{x}_i \leq \Delta$, where Δ is the maximum degree of any vertex in G.

Proof. Suppose by contradiction that there exists a graph G' such that in any optimal solution x^*, there exists a variable with $x_i^* \geq \Delta + 1$. Take such an optimal solution and consider variable x_j with $x_j^* \geq \Delta + 1$. Let $N(x_j)$ be the neighborhood in G' of x_j:

$$N(x_j) = \{x_i : (x_i, x_j) \in E(G')\}$$

Δ is the size of the maximum neighborhood in G', therefore $|N(x_j)| \leq \Delta$. Hence, by the pigeonhole principle, there exists a $d \in \{0, 1, \ldots, \Delta\}$ such that $d \neq x_i^*$ for all $x_i \in N(x_j)$. Consider the feasible point $\tilde{x}$ defined by:

$$\tilde{x}_i = \begin{cases} x_i^* & : i \neq j \\ d & : i = j \end{cases}$$

$c \geq 0$, therefore

$$a\tilde{x} = \sum_{i \in [v]} a_i \tilde{x}_i = \sum_{i \in [v], i \neq j} a_i \tilde{x}_i + a_j d \leq \sum_{i \in [v], i \neq j} a_i \tilde{x}_i + a_j (\Delta + 1) \leq ax^*$$

We can do the same for all i with $x_i^* \geq \Delta + 1$, contradicting that the optimal solution cannot have all variables with value less than or equal to Δ. $\Box$

We note that here that the above results is tight: there are graphs and vectors c for which in any optimal solution $x_j^* = \Delta$ for some j.

Let $G = (V, E)$ be the bipartite graph defined by

$$V = \{x_1, x_2, x_3\} \cup \{x_4, x_5, x_6\}$$

$$E = \{(x_1, x_4), (x_1, x_5), (x_1, x_6), (x_2, x_4), (x_2, x_6), (x_3, x_4), (x_3, x_5)\}$$

Consider

$$1x_1 + 100x_2 + 1{,}000x_3 + 10x_4 + 100x_5 + 1{,}000x_6$$

It is not hard to see that the unique optimal solution is:

$$x^* = (3, 1, 0, 2, 1, 0),$$

with $\tilde{x}_1 = 3 = \Delta$.

The above example can be extended to arbitrary Δ.

By symmetry we get the following lemma as well.

Lemma 13 *Then there exists an optimal solution x^* to (6.13) for which, for all $j, x_j^* \geq (n-1) - \Delta$.*

We use Lemmas 12,13 to show that when the domain set is sufficiently large, all facet-defining inequalities must have the coefficients of the variables take on the same sign.

Theorem 15 *If $|D^k| \geq 2\Delta + 2$ (i.e., k is chosen so that $k \geq 2\Delta + 1$), then any facet defining inequality must have all coefficients of the same sign.*

Proof. Suppose we have an alldifferent system with $|D^k| \geq 2\Delta + 2$. By contradiction, suppose there is a facet defining inequality, $\alpha x \geq \delta$, where some of the α_i's are positive and some are negative. (Without loss of generality, we can assume the inequality is a greater than or equal to and $\alpha \geq 0$.)

Letting $\alpha = [\alpha^+ : \alpha^-]$ with $\alpha^+ = \{\alpha_i : \alpha_i \geq 0\}$ and $\alpha^- = \{\alpha_i : \alpha_i < 0\}$, we define:

$$\alpha^1 = [\alpha^+ : 0] \text{ and } \alpha^2 = [0 : \alpha^-].$$

Consider the sum coloring problems P^1, P^2, with the objective functions α^1, α^2, respectively.

By lemma 12, there exists an optimal solution x^1 for P^1 with $x^1_j \leq \Delta, \forall j$ and by lemma 13, there exists an optimal solution x^2 for P^2 with $x^2_j \geq (n-1) - \Delta \geq \Delta + 1, \forall j$. Letting δ_1, δ_2 be defined by $\delta_1 = \alpha^1 x^1, -\delta_2 = -\alpha^2 x^2$, we get the following two valid inequality for P_I:

$$\alpha^1 x \geq \delta_1 \text{ and } \alpha^2 x \geq \delta_2.$$

Let F be the set of feasible colorings of a graph: i.e., $F = \{x : x_j \neq x_{j'}, \forall (j, j') \in E\}$.

Now, since $\alpha x \geq \delta$ is facet defining, it must be the case that

$$\delta = \min_{x \in F} \alpha x,$$

and since for any two functions f, g with common support S,

$$\min_{x \in S}\{f(x) + g(x)\} \geq \min_{x \in S}\{f(x)\} + \min_{x \in S}\{g(x)\},$$

we get that $\delta \geq \delta_1 + \delta_2$. Now suppose that $\delta > \delta_1 + \delta_2$; i.e.,

$$\min_{x \in F} \alpha x > \delta_1 + \delta_2.$$

Consider the point $\tilde{x}$ defined by:

$$\tilde{x}_i = \begin{cases} x^1_i & : \alpha_i \geq 0 \\ x^2_i & : \alpha_i < 0 \end{cases}$$

We show now that $\tilde{x}$ is feasible.

Suppose by contradiction that $\tilde{x} \notin F$. As $\tilde{x}_j \in D^k$, there must exists a pair of variables x_i, x_j in the same constraint for which $\tilde{x}_i = \tilde{x}_j$. If $\alpha_i, \alpha_j \geq 0$ then $\tilde{x}_i \neq \tilde{x}_j$ because x^1 is feasible. Similarly if $\alpha_i, \alpha_j < 0$. Therefore, either α_i or α_j are greater than or equal to 0, but not both. Without loss of generality, suppose $\alpha_i \geq 0$ and $\alpha_j < 0$. But then $\tilde{x}_i = x_i^1 \leq \Delta$ and $\tilde{x}_j = x_j^2 \geq \Delta + 1$ contradicting that $\tilde{x}_i = \tilde{x}_j$.

Hence, $\tilde{x}$ is feasible and $\alpha\tilde{x} = \alpha^1 x^1 + \alpha^2 x^2 = \delta_1 + \delta_2$ contradicting that

$$\min_{x \in F} \alpha x = \delta > \delta_1 + \delta_2.$$

Therefore, $\delta = \delta_1 + \delta_2$. But then we can write $\alpha x \geq \delta$ as a positive combination of two valid inequalities, $\alpha^1 x \geq \delta_1$ and $\alpha^2 x \geq \delta_2$, contradicting that $\alpha x \geq \delta$ is facet defining. □

6.5.3 Facet Invariance to Affine Transformations in the Domain Set

In this section we show that there is a one-to-one correspondence between the facet-defining inequalities under affine mappings of the domain set. In particular, this shows that facet-defining inequalities for all-different systems with domain sets D_δ and $D_{\delta'}$ are unique up to changes in the right-hand side.

For the rest of this section, suppose that we have a fixed all-different system composed of K all-different constraints, as in the definition of the problem in (6.1).

Let D, D' be domain sets of equal cardinality, for which there exists constants $a', b' \neq 0$ for which for all $d \in D$ there exists a $d' \in D'$ with $d' = a'd + b'$. Let P, P' be the graph coloring problem defined in (6.1) with domain set D, D', respectively. Let $T(x) = a'x + b'$. The following lemma follows immediately.

Lemma 14 *x is feasible to P if and only if $\tilde{x}$ is feasible to P'.*

Theorem 16 *Let $ax \geq \delta_1$ be facet-defining for P. If $a' > 0$, then there exists δ_2 for which $ax \geq \delta_2$ is facet-defining for P'. If $a' < 0$, then there exists a δ_3 for which $ax \leq \delta_3$ is facet-defining for P'.*

Proof. We begin with the case when $a' > 0$.

$ax \geq \delta_1$ is facet defining for P, therefore there exists $\hat{x}$ satisfying $a\hat{x} = \delta_1$. Let $\delta_2 = aT(\hat{x})$.

First we show that $ax \geq \delta_2$ is valid for P'. Suppose by contradiction that there exists $\tilde{x}$ feasible to P' for which $a\tilde{x} < \delta_2$. Consider $T^{-1}(x) = \frac{1}{a'}x - \frac{b'}{a'}$. By lemma (14) since $a' \neq 0, T^{-1}(\tilde{x})$ is feasible to P, so that

$$\begin{array}{rcl} aT^{-1}(\tilde{x}) & \geq & \delta_1 \\ a(\frac{1}{a'}\tilde{x} - \frac{b'}{a'}) & \geq & \delta_1 \\ a\tilde{x} - ab' & \geq & a'\delta_1 \\ a\tilde{x} & \geq & a'\delta_1 + ab'. \end{array}$$

δ_2 is defined to equal $aT(\hat{x})$, which by the assumption is larger than $a\tilde{x}$. Therefore,

$$\begin{array}{rcl} aT(\hat{x}) & > & a'\delta_1 + ab' \\ a(a'\hat{x} + b') & > & a'\delta_1 + ab' \\ a(a'\hat{x}) + ab' & > & a'\delta_1 + ab' \\ a'(a\hat{x}) & > & a'\delta_1 \\ a\hat{x} & > & \delta_1 \end{array}$$

,

contradicting that $\delta_1 = a\hat{x}$. Therefore, $ax \geq \delta_2$ is valid for P'.

Now, since $ax \geq \delta_1$ is facet defining for P, there exists, n affinely independent points, feasible the P satisfying $ax \geq \delta_1$ at equality. Let $x^1, x^2, \ldots, x^n$ be such a set points. We show that the points $\{T(x^i)\}_{i=1}^n$ are n affinely independent points satisfying $ax \geq \delta_2$ at equality.

First, $\delta_2 = aT(\hat{x}) = a'\delta_1 + b'a = a'ax^i + b'a = aT(x^i)$ for each i, so that all points $T(x^i)$ satisfy $ax \geq \delta_2$ at equality.

Now, suppose $\sum_{i=1}^n \lambda^i T(x^i) = 0$ and $\sum_{i=1}^n \lambda^i = 0$. Then

$$0 = \sum_{i=1}^n \lambda^i T(x^i) = a' \sum_{i=1}^n \lambda^i (x^i) + b' \sum_{i=1}^n \lambda^i$$

We know $b' \sum_{i=1}^n \lambda^i = 0$ and since $a' \neq 0$ the above implies that

$$\sum_{i=1}^n \lambda^i (x^i) = 0.$$

And since $\{x^i\}_{i=1}^n$ are affinely independent, we get that $\lambda^i = 0$, for all j, finishing the proof that when $a' > 0, ax \geq \delta_2$ is facet defining for P'.

The proof for $a' < 0$ follows similarly. □

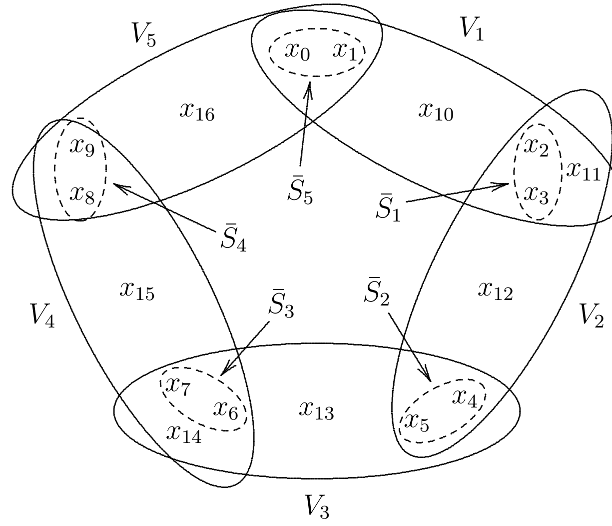


Figure 6.1: A 5-cycle. The solid ovals correspond to constraints alldiff(X_k) for $k = 1, \ldots, 5$. The sets $\bar{S}_1, \ldots, \bar{S}_2$ provide the basis for one possible valid cut with $s = 2$.

6.6 Cycles

We first investigate valid inequalities that correspond to odd cycles. We define a cycle in graph G to be a subgraph of G induced by the vertices in $V_1, \ldots, V_q \in V$ (for $q \geq 3$), where the subgraph induced by each V_k is a clique, and the only overlapping V_k's are adjacent ones in the cycle $V_1, \ldots, V_q, V_1$. Thus,

$$V_k \cap V_\ell = \begin{cases} S_k & \text{if } k+1 = \ell \text{ or } (k, \ell) = (q, 1) \\ \emptyset & \text{otherwise} \end{cases}$$

where $S_k \neq \emptyset$. A feasible vertex coloring on G must therefore satisfy

$$\text{alldiff}(X_k), \quad k = 1, \ldots, q \tag{6.14}$$

where again $X_k = \{x_i \mid i \in V_k\}$. The cycle is *odd* if q is odd. If $|V_k| = 2$ for each k, an odd cycle is an *odd hole*.

Figure 6.1 illustrates an odd cycle with $q = 5$. Each solid oval corresponds to a constraint alldiff(X_k). Thus $V_1 = \{0, 1, 2, 3, 10, 11\}$, and similarly for $V_2, \ldots, V_5$. All the vertices in a given V_k are connected by edges in G.

6.6.1 Valid Inequalities

We first identify valid inequalities that correspond to a given cycle. In the next section, we show that they are facet-defining.

Lemma 15 *Let $V_1, \ldots, V_q$ induce a cycle, and let $\bar{S}_k \subseteq S_k$ and $|\bar{S}_k| = s \geq 1$ for $k = 1, \ldots, q$. If q is odd and $\bar{S} = \bar{S}_1 \cup \cdots \cup \bar{S}_q$, the following inequality is valid for (6.1):*

$$\sum_{i \in \bar{S}} x_i \geq \beta(q, s) \tag{6.15}$$

where

$$\beta(q, s) = \frac{q-1}{2} \sum_{j=0}^{L-2} v_j + \left(sq - \frac{q-1}{2}(L-2) \right) v_{L-1}$$

and

$$L = \left\lceil \frac{sq}{(q-1)/2} \right\rceil$$

Proof. Because q is odd, each color can be assigned to at most $(q-1)/2$ vertices in the cycle. This means that the vertices must receive at least L distinct colors, and the variables in (6.14) must take at least L different values. Because $v_0 < \cdots < v_{n-1}$, we have

$$\sum_{i \in \bar{S}} x_i \geq \frac{q-1}{2}(v_0 + v_1 + \cdots + v_{L-2}) + \left(sq - \frac{q-1}{2}(L-2) \right) v_{L-1} = \beta(q, s)$$

where the coefficient of v_{L-1} is the number of vertices remaining to receive color v_L after colors $v_0, \ldots, v_{L-2}$ are assigned to $(q-1)/2$ vertices each. □

If the cycle is an odd hole, each $|S_k| = 1$ and $L = 3$. So (6.15) becomes

$$\sum_{i \in \bar{S}} x_i \geq \frac{q-1}{2}(v_0 + v_1) + v_2 \tag{6.16}$$

If the domain $\{v_0, \ldots, v_{n-1}\}$ of each x_i is $D_\delta = \{0, \delta, 2\delta, \ldots, (n-1)\delta\}$ for some $\delta > 0$, inequality (6.15) becomes

$$\sum_{i \in \bar{S}} x_i \geq \left(sq - \frac{q-1}{4} L \right) (L-1)\delta \tag{6.17}$$

for a general cycle and

$$\sum_{i \in \bar{S}} x_i \geq \frac{q+3}{2} \delta$$

for an odd hole.

An example with $q = 5$ appears in Fig. 6.1. By setting $s = 2$ we can obtain 9 valid inequalities by selecting 2-element subsets $\bar{S}_2$ and $\bar{S}_4$ of S_2 and S_4, respectively. Here $L = 5$, and if the colors are $0, \ldots, 9$, the right-hand side of the cut is $\beta(5, 2) = 20$. The sets $\bar{S}_1, \ldots, \bar{S}_5$ illustrated in the figure give rise to the valid inequality

$$x_0 + \cdots + x_9 \geq 20 \tag{6.18}$$

6.6.2 Facet-defining Inequalities

We now show that the valid inequalities identified in Lemma 15 are facet-defining. Let the variables x_i for $i \in \bar{S}$ be indexed $x_0, \ldots, x_{qs-1}$. We will say that a partial solution

$$(x_0, x_1, \ldots, x_{qs-1}) = (\bar{x}_0, \bar{x}_1 \ldots, \bar{x}_{qs-1}) \tag{6.19}$$

is feasible for (6.1) if it can be extended to a feasible solution of (6.1). That is, there is a complete solution $(x_1, \ldots, x_n)$ that is feasible in (6.1) and that satisfies (6.19). Because $|V|$ colors are available, any partial solution (6.19) that satisfies (6.14) can be extended to a feasible solution simply by assigning the remaining vertices distinct unused colors. That is, assign vertices in $V \setminus \{0, \ldots, sq-1\}$ distinct colors from the set $J \setminus \{\bar{x}_0, \ldots, \bar{x}_{sq-1}\}$.

Theorem 17 *If the graph coloring problem (6.1) is defined on a graph in which vertex sets $V_1, \ldots, V_q$ induce a cycle, where q is odd, then inequality (6.15) is facet defining for (6.1).*

Proof. Define

$$F = \{x \text{ feasible for (6.1)} \mid (x_0, \ldots, x_{qs-1}) \text{ satisfies (6.15) at equality}\}$$

It suffices to show that if $\mu x \geq \mu_{n+1}$ holds for all $x \in F$, then there is a scalar $\lambda > 0$ such that

$$\mu_i = \begin{cases} \lambda & \text{for } i = 0, \ldots, qs-1 \\ \beta(q,s)\lambda & \text{for } i = n+1 \\ 0 & \text{otherwise} \end{cases} \tag{6.20}$$

We will construct a partial solution $(\bar{x}_0, \ldots, \bar{x}_{qs-1})$ that is feasible for (6.1) as follows. Domain values $v_0, \ldots, v_{L-2}$ will occur $(q-1)/2$ times in the solution, and domain value

v_{L-1} will occur r times, where

$$r = qs - \frac{q-1}{2}(L-1)$$

This will ensure that (6.15) is satisfied at equality. We form the partial solution by first cycling r times through the values $v_0, \ldots, v_{L-1}$, and then by cycling through the values $v_0, \ldots, v_{L-2}$. Thus

$$\bar{x}_i = \begin{cases} v_{i \bmod L} & \text{for } i = 0, \ldots, rL-1 \\ v_{(i-rL) \bmod (L-1)} & \text{for } i = rL, \ldots, rs-1 \end{cases} \tag{6.21}$$

To show that this partial solution is feasible for the odd cycle, we must show

$$\begin{array}{ll} \text{alldiff}\{\bar{x}_i,\ i \in \bar{S}_k \cup \bar{S}_{k+1}\}, \text{ for } k = 1, \ldots, q-1 & (a) \\ \text{alldiff}\{\bar{x}_i,\ i \in \bar{S}_1 \cup \bar{S}_q\} & (b) \end{array}$$

To show (a), we note that the definition of L implies $L - 1 \geq 2s$. Therefore, any sequence of $2s$ consecutive $\bar{x}_i$'s are distinct, and (a) is satisfied. To show (b), we note that the number of values $\bar{x}_{rL}, \ldots, \bar{x}_{rs-1}$ is

$$(rs-1) - rL + 1 = (L-1)\left(\frac{q-1}{2}L - qs\right)$$

from the definition of r. Because the number of values is a multiple of $L-1$, the values $\bar{x}_i$ for $i \in \bar{S}_q$ are $(\bar{x}_{(q-1)s}, \ldots, \bar{x}_{qs-1}) = (v_{L-s-1}, \ldots, v_{L-2})$, and they are all distinct. The values $\bar{x}_i$ for $i \in \bar{S}_1$ are $(\bar{x}_0, \ldots, \bar{x}_{s-1}) = (v_0, \ldots, v_{s-1})$ and are all distinct. But $L - 1 \geq 2s$ implies $L - s > s$, and (b) follows.

We now construct a partial solution $(\tilde{x}_0, \ldots, \tilde{x}_{qs-1})$ from the partial solution in (6.21) by swapping any two values $\bar{x}_\ell, \bar{x}_{\ell'}$ for $\ell, \ell' \in \bar{S}_k \cup \bar{S}_{k+1}$, for any $k \in \{1, \ldots, q-1\}$. That is,

$$\tilde{x}_i = \begin{cases} \bar{x}_{\ell'} & \text{if } i = \ell \\ \bar{x}_\ell & \text{if } i = \ell' \\ \bar{x}_i & \text{otherwise} \end{cases} \tag{6.22}$$

Extend the partial solutions (6.21) and (6.22) to complete solutions $\bar{x}$ and $\tilde{x}$, respectively, by assigning values with

$$\bar{x}_i = \tilde{x}_i \text{ for } i \notin \{0, \ldots, qs-1\}$$

such that the values assigned to $\bar{x}_i$ for $i \notin \{0, \ldots, qs-1\}$ are all distinct and do not belong to $\{v_0, \ldots, v_{L-1}\}$. Because $\bar{x}$ and $\tilde{x}$ are feasible and satisfy (6.15) at equality, they satisfy $\mu x = \mu_{n+1}$. So we have $\mu\bar{x} = \mu\tilde{x}$, which implies $\mu_\ell = \mu_{\ell'}$ for $\ell, \ell' \in \bar{S}_k \cup \bar{S}_{k+1}$ for any pair $\ell, \ell' \in \bar{S}_k \cup \bar{S}_{k+1}$ and any $k \in \{1, \ldots, q-1\}$. This implies

$$\mu_\ell = \mu_{\ell'} \text{ for any } \ell, \ell' \in \bar{S} \tag{6.23}$$

Define $\bar{x}'$ by letting $\bar{x}' = \bar{x}$ except that for an arbitrary $\ell \notin \{0, \ldots, qs-1\}$, $\bar{x}'_\ell$ is assigned a value that does not appear in the tuple $\bar{x}$. Since $\bar{x}$ and $\bar{x}'$ are feasible and satisfy (6.15) at equality, we have $\mu\bar{x} = \mu\bar{x}'$. This and $\bar{x}_\ell \neq \bar{x}'_\ell$ imply

$$\mu_i = 0, \;\; i \in V \setminus \{0, \ldots, qs-1\} \tag{6.24}$$

Finally, (6.23) implies that for some $\lambda > 0$,

$$\mu_i = \lambda, \;\; i = 0, \ldots, qs-1 \tag{6.25}$$

Because $\mu\bar{x} = \mu_{n+1}$, we have from (6.25) that $\mu_{n+1} = \beta(q,s)\lambda$. This, (6.24), and (6.25) imply (6.20). □

In the example of Fig. 6.1, suppose that the vertices in $V_1, \ldots, V_5$ induce a cycle of G. That is, all vertices in each V_k are connected by edges, and there are no other edges of G between vertices in $V_1 \cup \cdots \cup V_5$. Then (6.18) is facet-defining for (6.1).

6.6.3 Bounds on the Chromatic Number

We can write a facet-defining inequality involving the objective function variable z if the domain of each x_i is D_δ for $\delta > 0$. To do so we rely on the following:

Theorem 18 *If $ax \geq \beta$ is facet-defining for a graph coloring problem (6.1) in which each x_i has domain D_δ for $\delta > 0$, then*

$$aez \geq ax + \beta \tag{6.26}$$

is also facet defining, where $e = (1, \ldots, 1)$.

Proof. To show that (6.26) is valid, note that for any $x \in D_\delta^n$, $z - x_i \in D_\delta$ for all i, where $z = \max_i\{x_i\}$. Because $ax \geq \beta$ is valid for all $x \in D_\delta^n$ and $z - x_i \in D_\delta$, $ax \geq \beta$ holds when

$z - x_i$ is substituted for each x_i. This implies (6.26) because z in (6.1) satisfies $z \geq x_i$ for each i.

To show that the z-cut (6.26) is facet-defining, let

$$F = \{(z, x) \text{ feasible for (6.1)} \mid aez = ax + \beta\}$$

It suffices to show that if $\mu_z z = \mu x + \mu_0$ is satisfied by all $(z, x) \in F$, then there is a $\lambda > 0$ with

$$\begin{aligned} \mu_z &= \lambda ae \\ \mu &= \lambda a \\ \mu_0 &= \lambda\beta \end{aligned} \tag{6.27}$$

Let $F' = \{x \text{ feasible for (6.1)} \mid ax = \beta\}$. F' is nonempty because $ax \geq \beta$ is facet defining. F is therefore nonempty, because for any $x \in F'$, we have $(\bar{z}, \bar{x}) \in F$ where $\bar{z} = \max_i\{x_i\}$ and $\bar{x} = ze - x$. But for any point $(z, x) \in F$, we also have $(z+\delta, x+\delta e) \in F$. So $\mu_z z = \mu x + \mu_0$ and $\mu_z(z + \delta) = \mu(x + \delta e) + \mu_0$. Subtracting one equation from the other, we get $\mu_z = \mu e$. We now claim that any $(ez - x) \in F'$ satisfies $\mu(ez - x) = \mu_0$. This is because $(ez - x) \in F'$ implies $(z, x) \in F$, which implies $\mu ez = \mu x + \mu_0$, which implies $\mu(ex - x) = \mu_0$. But because $ax \geq \beta$ is facet defining, there is a $\lambda > 0$ for which $\mu = \lambda a$ and $\mu_0 = \lambda\beta$. Because $\mu_z = \mu e$, this same λ satisfies (6.27). □

Inequality (6.15) and Theorem 18 imply

Corollary 1 *If the graph coloring problem (6.1) is defined on a graph in which vertex sets $V_1, \ldots, V_q$ induce a cycle, where q is odd and each x_i has domain D_δ with $\delta > 0$, then*

$$z \geq \frac{1}{qs} \sum_{i \in \bar{S}} x_i + \frac{\beta(q, s)}{qs} \tag{6.28}$$

is facet defining for (6.1), where

$$\frac{\beta(q, s)}{qs} = \left(1 - \frac{q-1}{4qs} L\right)(L - 1)\delta$$

In the case of an odd hole ($s = 1$), the z-cut is

$$z \geq \frac{1}{q} \sum_{i \in \bar{S}} x_i + \frac{q+3}{2q}\delta$$

In the example of Fig. 6.1, the z-cut is

$$z \geq \tfrac{1}{10}(x_0 + \cdots + x_9) + 2 \tag{6.29}$$

6.6.4 Mapping to 0-1 Cuts

The 0-1 model for a coloring problem on a cycle has the following continuous relaxation:

$$\begin{aligned} &\sum_{j\in J} y_{ij} = 1, \ \ i = 1,\ldots,q && (a)\\ &\sum_{i\in V_k} y_{ij} \le w_j, \ \ j\in J, \ k=1,\ldots,q && (b)\\ &0 \le y_{ij}, w_j \le 1, \ \ \text{all } i,j && (c) \end{aligned} \tag{6.30}$$

Because constraints (b) appear for each maximal clique, the relaxation implies all clique inequalities $\sum_{i\in V_k} y_{ij} \le 1$. Nonetheless, we will see that two finite-domain cuts strengthen the relaxation more than the collection of all odd hole cuts.

To simplify discussion, let each x_i have domain $D_1 = \{0,1,\ldots,n-1\}$. The x-cut (6.17) maps into the cut

$$\sum_{i\in\bar{S}}\sum_{j=1}^{n-1} j y_{ij} \ge \left(sq - \frac{q-1}{4}L\right)(L-1) \tag{6.31}$$

which is valid by Lemma 9. The z-cut (6.28) maps into

$$\sum_{j=0}^{n-1} w_j - 1 \ge \frac{1}{q}\sum_{i\in\bar{S}}\sum_{j=1}^{n-1} j y_{ij} + \frac{q+3}{2q} \tag{6.32}$$

which is valid by Lemma 10.

We will compare cuts (6.31)–(6.32) with classical odd hole cuts, which have the form

$$\sum_{i\in H} y_{ij} \le \frac{q-1}{2} w_j, \ \ j = 0,\ldots,n-1 \tag{6.33}$$

where H is the vertex set for an odd hole. The cut (6.33) is not facet defining in general, although it is facet defining when H contains all vertices of G. This is in contrast with the finite-domain cut (6.15), which is facet defining in the x-space for any odd hole in G (and more generally, any odd cycle in G).

We first note that when $s = 1$, the 0-1 x-cut (6.31) is redundant of odd hole cuts.

Lemma 16 *If $s = 1$, the 0-1 x-cut (6.31) is implied by the 0-1 model (6.30) with odd hole cuts (6.33).*

Proof. When $s = 1$, the cut (6.31) becomes

$$\sum_{i\in\bar{S}}\sum_{j=0}^{n-1} j y_{ij} \ge \frac{q+3}{2} \tag{6.34}$$

It suffices to show that (6.34) is dominated by a nonnegative linear combination of (6.30) and (6.33), where $H = \bar{S}$ in (6.33). Assign multiplier 2 to each constraint in (6.30a); multipliers 2 and 1, respectively, to constraints (6.33) with $j = 0, 1$; and multipliers $q-1$ and $(q-1)/2$, respectively, to the constraints $w_0 \leq 1$ and $w_1 \leq 1$. The resulting linear combination is

$$\sum_{i\in\bar{S}} y_{i1} + 2\sum_{j=2}^{n-1}\sum_{i\in\bar{S}} y_{ij} \geq 2q - \frac{q-1}{2} - (q-1) = \frac{q+3}{2}$$

This dominates (6.34) because the left-hand side coefficients are less than or equal to the corresponding coefficients in (6.34). □

However, the two finite-domain cuts (6.31) and (6.32), when combined, provide a tighter bound than the n odd hole cuts (6.33) even when $s = 1$. For example, when $q = 5$, the 10 odd hole cuts provide a lower bound of 2.5 on the chromatic number, while the two finite-domain cuts provide a bound of 2.6. The improvement is modest, but 10 cuts are replaced by only two cuts. Comparisons for larger q appear in the next section.

Furthermore, when $s > 1$, the single 0-1 z-cut (6.32) provides a tighter bound than the collection of all odd hole cuts, which have no effect in this case. There are s^q odd hole cuts (6.33) for each color j, one for every H that selects one element from each S_k, $k = 1, \ldots, q$. For example, when $q = 5$ and $s = 2$, there are $ns^q = 320$ odd hole cuts. The lower bound on the chromatic number is 4.0 with or without them. However, the one finite-domain cut (6.32) yields a bound of 4.5. Addition of the 0-1 x-cut (6.31) strengthens the bound further, raising it to 5.0. This bound is actually sharp in the present instance, because the chromatic number is 5. Thus two finite-domain cuts significantly improve the bound, while 320 odd hole cuts have no effect on the bound. Further comparisons appear in Section 6.11.

6.6.5 Separation

Separating cuts can be identified in either the x-space or the y-space. When a continuous relaxation of the 0-1 model is solved, the resulting values of the y_{ij}s can be used to identify a separating cut directly in 0-1 space. Alternatively, these values can be mapped to values of the x_js using the transformation $x_i = \sum_j v_j y_{ij}$, and a separation algorithm applied in x-space.

In practice, a solver may apply existing algorithms to identify separating odd hole cuts. The odd holes that give rise to these cuts can trigger the generation of an x-cut and a z-cut. These superior cuts can then replace the odd hole cuts.

If odd cycle cuts for $s > 1$ are desired, a separation algorithm can be applied to the x_i-values by heuristically seeking a cycle that gives rise to separating cuts. We show here that a simple polynomial-time algorithm identifies a separating x-cut and a separating z-cut for a given cycle if such cuts exist.

The algorithm is as follows. We again suppose the colors are $0, 1, \ldots, n-1$. Let (6.14) be an odd q-cycle for which we wish to find a separating cut. Let $\bar{y}, \bar{w}$ be a solution of the continuous relaxation of the 0-1 model, and let

$$\bar{x}_i = \sum_{j=1}^{n-1} j\bar{y}_{ij}, \quad i \in \bigcup_{k=1}^{q} V_k \qquad\qquad \bar{z} = \sum_{j=0}^{n-1} \bar{w}_j - 1$$

For each $k = 1, \ldots, q$, define the bijection $\pi_k : \{1, \ldots, |S_k|\} \rightarrow S_k$ such that $\bar{x}_{\pi_k(\ell)} \leq \bar{x}_{\pi_k(\ell')}$ whenever $\ell < \ell'$. Then for $s = 1, \ldots, \min_k |S_k|$, generate a separating x-cut

$$\sum_{k=1}^{q} \sum_{\ell=1}^{s} x_{\pi_k(\ell)} \geq \beta(q, s)$$

whenever $\bar{x}$ violates this inequality, and generate a separating z-cut

$$z \geq \frac{1}{qs} \sum_{k=1}^{q} \sum_{\ell=1}^{s} x_{\pi_k(|S_k|-\ell+1)} + \frac{\beta(q, s)}{qs}$$

whenever $(\bar{x}, \bar{z})$ violates this inequality. The running time of the algorithm is $\mathcal{O}(q\bar{s}\log\bar{s})$, where $\bar{s} = \max_k |S_k|$ and $\bar{s}\log\bar{s}$ is the sort time for $\bar{s}$ values.

Lemma 17 *The above algorithms find a separating x-cut and separating z-cut for a given odd q-cycle if such cuts exist.*

Proof. Suppose there is a separating x-cut with $\bar{S}_k \subset S_k$ and $s^* = |\bar{S}_k|$ for $k = 1, \ldots, q$. Then

$$\sum_{i \in \bar{S}} \bar{x}_i < \beta(q, s^*) \tag{6.35}$$

where $\bar{S} = \bigcup_k \bar{S}_k$. Because π_k orders the elements of S_k by size,

$$\sum_{\ell=1}^{s^*} \bar{x}_{\pi_k(\ell)} \leq \sum_{i \in \bar{S}_k} \bar{x}_i, \quad k = 1, \ldots, q$$

Summing this over $k = 1, \ldots, q$, we get

$$\sum_{k=1}^{q}\sum_{\ell=1}^{s}\bar{x}_{\pi_k(\ell)} \leq \sum_{i\in\bar{S}}\bar{x}_i < \beta(q,s)$$

where the strict inequality is due to (6.35). This means that the algorithm generates the separating cut for $s = s^*$. The proof is similar for z-cuts. □

6.6.6 Additional Cycle Facts

In this section we present an additional facet-defining inequality in (6.36) for a cycle graph that is not of the form (6.15). The inequality displays that a graph may not have $|V_i \cap V_{i+1}|$ be a uniform value for each of the consecutive cliques in the cycle, but nonetheless, may still admit a facet-defining inequality. In addition, several other inequalities can be derived from (6.36), including a complement inequality (as described in Theorem 14), a $z-$cut (as described in Theorem 18), and all of these inequalities can be mapped into 0-1 space to strengthen the linear relaxation, similar to the discussion in Section 6.6.4.

Theorem 19 *Consider the cycle all-different system in Figure 6.2, and let $D = D_1$. The inequality*

$$3\cdot(x_1 + x_2 + x_7 + x_8) + 4\cdot(x_3 + x_4 + x_5 + x_6) \geq 42 \tag{6.36}$$

is facet-defining.

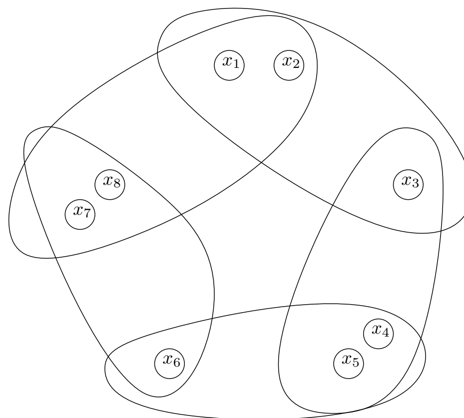


Figure 6.2: Cycle graph with non-uniformly sized intersections

Proof. Let $ax \geq b$ be inequality (6.36). We being with a proof validity.

Let $\tilde{x}$ be a feasible solution. We first show that if there are no repeated values among the values $\{\tilde{x}_3, \tilde{x}_4, \tilde{x}_5, \tilde{x}_6\}$ then inequality (6.36) is satisfied.

If there are no repeated values in $\{\tilde{x}_3, \tilde{x}_4, \tilde{x}_5, \tilde{x}_6\}$ then

$$\tilde{x}_3 + \tilde{x}_4 + \tilde{x}_5 + \tilde{x}_6 \geq 0 + 1 + 2 + 3 = 6 \tag{6.37}$$

The remaining values $\{\tilde{x}_1, \tilde{x}_2, \tilde{x}_7, \tilde{x}_8\}$ all belong to a single alldiff constraint so that

$$\tilde{x}_1 + \tilde{x}_2 + \tilde{x}_7 + \tilde{x}_8 \geq 0 + 1 + 2 + 3 = 6 \tag{6.38}$$

And so, in the case that all of the values in $\{\tilde{x}_3, \tilde{x}_4, \tilde{x}_5, \tilde{x}_6\}$ are unique, we get

$$a\tilde{x} \geq 3 \cdot 6 + 4 \cdot 6 = 42, \tag{6.39}$$

as desired.

We now suppose that there is some repeated value in the set $\{\tilde{x}_3, \tilde{x}_4, \tilde{x}_5, \tilde{x}_6\}$ implying that $\tilde{x}_3 = \tilde{x}_6$. Let this common value be k. We now condition on the value of k, noting that no other variables can be assigned the value k since the union of the neighborhoods of the vertices corresponding to x_3 and x_6 span all of the vertices in the graph.

- $k = 0$

  In this case, we have

$$\tilde{x_4} + \tilde{x_5} \geq 1 + 2 = 3 \tag{6.40}$$

  and

$$\tilde{x}_1 + \tilde{x}_2 + \tilde{x}_7 + \tilde{x}_8 \geq 1 + 2 + 3 + 4 = 10 \tag{6.41}$$

  Therefore,

$$a\tilde{x} \geq 3 \cdot 10 + 4 \cdot (0 + 1 + 2 + 0) = 42, \tag{6.42}$$

  as desired.

- $k = 1$

  In this case, we have

$$\tilde{x_4} + \tilde{x_5} \geq 0 + 2 = 2 \tag{6.43}$$

  and

$$\tilde{x}_1 + \tilde{x}_2 + \tilde{x}_7 + \tilde{x}_8 \geq 0 + 2 + 3 + 4 = 9 \tag{6.44}$$

  Therefore,

$$a\tilde{x} \geq 3 \cdot 9 + 4 \cdot (1 + 0 + 2 + 1) = 43 > b, \tag{6.45}$$

  as desired.

- $k = 2$

  In this case, we have

$$\tilde{x_4} + \tilde{x_5} \geq 0 + 1 = 1 \tag{6.46}$$

  and

$$\tilde{x}_1 + \tilde{x}_2 + \tilde{x}_7 + \tilde{x}_8 \geq 0 + 1 + 3 + 4 = 8 \tag{6.47}$$

  Therefore,

$$a\tilde{x} \geq 3 \cdot 8 + 4 \cdot (2 + 0 + 1 + 2) = 44 > b, \tag{6.48}$$

  as desired.

- $k = 3$

  In this case, we still have

$$\tilde{x_4} + \tilde{x_5} \geq 0 + 1 = 1 \tag{6.49}$$

  and now

$$\tilde{x}_1 + \tilde{x}_2 + \tilde{x}_7 + \tilde{x}_8 \geq 0 + 1 + 2 + 4 = 7 \tag{6.50}$$

  Therefore,

$$a\tilde{x} \geq 3 \cdot 7 + 4 \cdot (3 + 0 + 1 + 3) = 49 > b, \tag{6.51}$$

as desired.

- $k \geq 4$

  In this case, we still have

$$\tilde{x_4} + \tilde{x_5} \geq 0 + 1 = 1 \tag{6.52}$$

and as always

$$\tilde{x}_1 + \tilde{x}_2 + \tilde{x}_7 + \tilde{x}_8 \geq 0 + 1 + 2 + 3 = 6 \tag{6.53}$$

Therefore,

$$a\tilde{x} \geq 3 \cdot 6 + 4 \cdot (k + 0 + 1 + k) = 18 + 4 \cdot (2k + 1) >= 54 > b, \tag{6.54}$$

as desired, and finishing the proof that $ax \geq b$ is valid for all feasible solutions.

We now prove that $ax \geq b$ is facet-defining. Let

$$F = \{x : x \text{ is feasible and } ax = b\}$$

We show that if (μ, μ_0) are such that $\mu x = \mu_0$ for every $x \in F$, then for some λ, $\mu_i = \lambda \cdot a_i$ for each i and $\mu_0 = \lambda \cdot b$.

Let $\tilde{x} = (0, 1, 3, 1, 2, 0, 2, 3)$. It is readily seen that $\tilde{x} \in F$.

Lemma 18 $\mu_1 = \mu_2 = \mu_7 = \mu_8$

Proof. Switching the values assigned to x_1 and x_2 in $\tilde{x}$ yields another feasible solution also in F. Therefore, $\mu_1 = \mu_2$. This is also true for x_7 and x_8 so that $\mu_7 = \mu_8$.

Now, consider the solution $\hat{x}$ for which $\hat{x}_2 = 2$ and $\hat{x}_7 = 1$ with all other values $\hat{x}_i = \tilde{x}_i$; i.e., we switch the values assigned to x_2 and x_7. $\hat{x} \in F$ and therefore, we have $\mu\hat{x} = \mu\tilde{x}$. Simplifying yields $\mu_2 = \mu_7$. Therefore, $\mu_1 = \mu_2 = \mu_7 = \mu_8$, as desired.

Let λ_A be the common value of $\mu_1, \mu_2, \mu_7, \mu_8$.

Lemma 19 $\mu_3 = \mu_4 = \mu_5 = \mu_6$

Proof. Switching the values assigned to x_4 and x_5 in $\tilde{x}$ yields another feasible solution also in F. Therefore, $\mu_4 = \mu_5$.

Now, consider the solution $\hat{x}$ for which $\hat{x}_3 = 2$ and $\hat{x}_5 = 3$ with all other values $\hat{x}_i = \tilde{x}_i$; i.e., we switch the values assigned to x_3 and x_5. $\hat{x} \in F$ and therefore, we have $\mu\hat{x} = \mu\tilde{x}$. Simplifying yields $\mu_3 = \mu_5$.

Now, consider the solution $\hat{x}$ for which $\hat{x}_4 = 0$ and $\hat{x}_6 = 1$ with all other values $\hat{x}_i = \tilde{x}_i$; i.e., we switch the values assigned to x_4 and x_6. $\hat{x} \in F$ and therefore, we have $\mu\hat{x} = \mu\tilde{x}$. Simplifying yields $\mu_4 = \mu_6$.

Therefore, $\mu_3 = \mu_4 = \mu_5 = \mu_6$, as desired.

Let λ_B be the common value of $\mu_3, \mu_4, \mu_5, \mu_6$. We can then rewrite $\mu x = \mu_0$ as

$$\lambda_A \cdot (x_1 + x_2 + x_7 + x_8) + \lambda_B \cdot (x_3 + x_4 + x_5 + x_6) = \mu_0. \tag{6.55}$$

Lemma 20 $\frac{4}{3} \cdot \lambda_A = \lambda_B$

Proof. Consider the solution $\hat{x}$ for which $\hat{x}_3 = 0$ and $\hat{x}_1 = 4$ with all other values $\hat{x}_i = \tilde{x}_i$.

We first show that $\hat{x} \in F$. $\hat{x}$ is feasible since in each alldiff constraint all of the variables take pairwise different values. It remains to show that $ax = b$. Plugging in yields

$$a\hat{x} = 3 \cdot (4 + 1 + 2 + 3) + 4 \cdot (0 + 1 + 2 + 0) = 42,$$

as desired.

Therefore, $\hat{x} \in F$ and we have $\mu\hat{x} = \mu\tilde{x}$. Simplifying yields $4\mu_1 = 3\mu_3$ and plugging in λ_A and λ_B yields $\frac{4}{3} \cdot \lambda_A = \lambda_B$ as desired.

We now write $\mu x = \mu_0$ as

$$\lambda_A \cdot (x_1 + x_2 + x_7 + x_8) + \frac{4}{3} \cdot \lambda_A \cdot (x_3 + x_4 + x_5 + x_6) = \mu_0.$$

Let $\lambda = \frac{1}{3}\lambda_A$. This yields

$$3\lambda \cdot (x_1 + x_2 + x_7 + x_8) + 4\lambda \cdot (x_3 + x_4 + x_5 + x_6) = \mu_0.$$

Finally, as $\tilde{x} \in F$, $\mu\tilde{x} = \mu_0$. Therefore

$$3\lambda \cdot (0 + 1 + 2 + 3) + 4\lambda \cdot (3 + 1 + 2 + 0) = \mu_0,$$

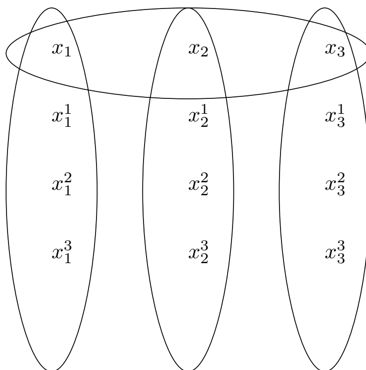


Figure 6.3: A comb with 3 variables in the handle and three teeth, each with 3 variables.

yielding $18\lambda + 24\lambda = \mu_0$. Therefore $\mu_0 = 42\lambda$, finishing the proof that $ax \geq b$ is facet-defining. □

6.7 Combs

In this section we study cuts that arise from combs. Comb structures often arise as structures from which cutting planes can be deduced, for example in the polyhedral analysis of the Traveling Salesman Problem [31, 32].

Comb structures, in the context of all-different systems, were studied in [47, 48, 53]. We study them here again, where we find inequalities that depend on the cardinality of D.

A *comb* is a subgraph of G induced by vertices in subsets $H, T_1, \ldots, T_K$, (H is the handle and the T_j's are teeth) where each subset is a clique, and

$$|H \cap T_k| = 1 \quad , \text{ for } k = 1, \ldots, K$$
$$|T_k \cap T_{k'}| = 0 \quad , \text{ for all } 1 \leq k < k' \leq K$$

A comb with $H = \{x_1, x_2, x_3\}$ and $T_k = \{x_k^1, x_k^2, x_k^3\}$ is depicted in Figure 6.3.

Let $X(\ell, u)$ be the sum of the integers from ℓ to u, inclusive. We will prove that the following inequalities are facet-defining for a comb.

For $A, B \subseteq H, A \cap B = \emptyset, |A| = a \geq 1, |B| = b \geq 1, s = a + b$, and for any $A_i \subseteq T_i, |A_i| =$

m for $i \in A$, and for any $B_i \subseteq Ti, |B_i| = m$ for $i \in B$,

$$a \sum_{i \in A} \left(x_i + s \sum_{j \in A_i} x_i^j \right) - b \sum_{i \in B} \left(x_i + s \sum_{j \in B_i} x_i^j \right) \geq b_1, \tag{6.56}$$

with $k = 2m + s - 2$ and

$$\begin{aligned} b_1 = \quad & a\{X(m, m+a-2) + (m+s-1) + s\{a \cdot X(0, m-1)\} \\ & -b\{X(m+a-1, m+s-2) + s\{b \cdot X(m+s-1, 2m+s-2)\}. \end{aligned}$$

6.7.1 Validity Proof

Theorem 20 *Inequality (6.56) is valid.*

First we state two results from [49].

Lemma 21 $\forall A \subseteq H, |A| = a \geq 1$ *and* $\forall A_i \subseteq T_i, |A_i| = m$ *for* $i \in A$,

$$\sum_{i \in A} (x_i + a \sum_{j \in A_i} x_i^j) \geq X(m, m+a-1) + a^2 \cdot X(0, m-1)$$

is a valid inequality.

Lemma 22 $\forall B \subseteq H, |B| = b \geq 1$ *and* $\forall B_i \subseteq T_i, |B_i| = m$ *for* $i \in B$,

$$\sum_{i \in B} (x_i + b \sum_{j \in B_i} x_i^j) \leq X(k-m-b+1, k-m) + b^2 \cdot X(k-m+1, k)$$

is a valid inequality.

Now, with the use of the above lemmas, we may proceed with the proof of the validity of the inequality (6.56). Let the inequality (6.56) be given by $cx \geq b_1$, and suppose by contradiction that there exists a feasible $\tilde{x}$ which minimizes cx for which

$$c\tilde{x} \leq b_1 - 1. \tag{6.57}$$

Lemma 23 $\forall i \in A, \tilde{x}_i \geq m.$

Proof. Suppose that for some $\tilde{x}_k, k \in A, \tilde{x}_k < m$. Because $\tilde{x}$ minimizes cx, the variables in tooth k must all be assigned to the set $\{0, 1, \ldots, \tilde{x}_{k-1}, \tilde{x}_{k+1}, \ldots, m\}$. Now, given that $\tilde{x}_k < m$, consider the expression

$$\sum_{i \in A, i \neq k} (x_i + s \sum_{j \in A_i} x_i^j) \tag{6.58}$$

From lemma 21 we know that the greedy assignment of values to the teeth variables will minimize

$$\sum_{i \in A, i \neq k} (x_i + (a-1) \sum_{j \in A_i} x_i^j),$$

and since $s > a$, the greedy assignment must also minimize (6.58). Hence, given that for some $k, \tilde{x}_k < m$, we have that

$$\begin{aligned} a\{\sum_{i \in A} \tilde{x}_i + s \sum_{j \in A_i} \tilde{x}_i^j\} &= a\{\tilde{x}_k + s \sum_{j \in S_k} \tilde{x}_k^j + \sum_{i \in A, i \neq k} \tilde{x}_i + s \sum_{j \in A_i} \tilde{x}_i^j\} \\ &\geq a\{\tilde{x}_k + s(X(0,m) - \tilde{x}_k) + X(m, m+a-2) \\ &\quad +s(a-1)(X(0, m-1))\} \end{aligned}$$

And hence,

$$\begin{aligned} a\{\sum_{i \in A} \tilde{x}_i + s \sum_{j \in A_i} \tilde{x}_i^j\} \geq \quad & a\{X(m, m+a-2)) + s\{aX(0, m-1)\}\} \\ & +a(\tilde{x}_k + sm - s\tilde{x}_k) \end{aligned} \tag{6.59}$$

Now, consider the expression

$$\sum_{i \in B} x_i + s \sum_{j \in B_i} x_i^j. \tag{6.60}$$

From lemma 22, we know that the greedy assignment (the maximization version) of values to variables will maximize

$$\sum_{i \in B} (x_i + b \sum_{j \in B_i} x_i^j)$$

and since $s > b$, the greedy assignment must maximize (6.60). Hence,

$$-b\{\sum_{i \in B} (\tilde{x}_i + s \sum_{j \in B_i} \tilde{x}_i^j)\} \geq -b\{X(k-m-b+1, k-m) + sb \cdot X(k-m+1, k)\}$$

Plugging in $k = 2m + s - 2$ yields

$$-b\{\sum_{i\in B}(\tilde{x}_i + s\sum_{j\in B_i}\tilde{x}_i^j)\} \geq \quad -b\{X(m+a-1, m+s-2) + sb\cdot X(m+s-1, 2m+s-2)\} \quad (6.61)$$

Combining (6.57), (6.59) and (6.61) we get

$$a(\tilde{x}_i + sm - s\tilde{x}_k) \leq a(m+s-1) - 1,$$

which can be written as

$$(s-1)(m - \tilde{x}_k - 1) \leq -1,$$

and since $s - 1 \geq 0$ and $m - \tilde{x}_k - 1 \geq 0$, we have a contradiction. Therefore, $\forall i \in A, \tilde{x}_i \geq m$. □

Lemma 24 $\forall i \in B, \tilde{x}_i \leq k - m = m + s - 2.$

Proof. Follows similarly to the proof of lemma 23. □

Lemma 25 $\forall i \in A, \forall j \in A_i, \tilde{x}_i^j \leq m - 1$. *Hence* $\forall i \in A$,

$$\sum_{j\in A_i}\tilde{x}_i^j = X(0, m-1).$$

Proof. Suppose some variable x_k^l is such that $\tilde{x}_k^l > m-1$ for some $k \in A, l \in A_k$. Then some value $d \in \{0, 1, \ldots, m-1\}$ is not taken by any of the variables $x_k \cup \{x_k^j\}_{j\in A_k}$. Therefore defining $\hat{x} = \tilde{x}$ except that $\hat{x}_k^l = d$ yields a feasible solution such that $c\hat{x} < c\tilde{x}$ contradicting that $\tilde{x}$ minimizes cx. □

Lemma 26 $\forall i \in B, \forall j \in B_i, \tilde{x}_i^j \geq m + s - 1$. *Hence* $\forall i \in B$,

$$\sum_{j\in B_i}\tilde{x}_i^j = X(m+s-1, 2m+s-2 = k).$$

Proof. Follows similarly to the proof of lemma 25. □

Lemma 26 and lemma 25, together with $c\tilde{x} \leq b_1 - 1$, implies

$$a \sum_{i \in A} \tilde{x}_i - b \sum_{i \in B} \tilde{x}_i \leq \quad a\{X(m, m+a-2) + (m+s-1))\} \qquad (6.62)$$
$$-b\{X(m+a-1, m+s-2)\} - 1.$$

Suppose now, without loss of generality, that the variables in A are given by $x_1, x_2, \ldots, x_a$ and that the variables are sorted in increasing order ($\tilde{x}_1 < \cdots < \tilde{x}_a$). Similarly, let the variables in B be given by $x_{a+1}, \ldots, x_s$ and that the variables are sorted in increasing order ($\tilde{x}_{a+1} < \cdots < \tilde{x}_s$).

Lemma 27 $\tilde{x}_{a-1} = m + a - 2$ *and therefore for* $i = 1, 2, \ldots, a-1, \tilde{x}_i = m + i - 1$.

Proof. Note that from lemma 23, if $\tilde{x}_i = m + a - 2$, then we must have $\tilde{x}_i = m + i - 1$ for $i = 1, 2, \ldots, a-1$. Therefore we need only show that $\tilde{x}_{a-1} = m + a - 2$.

First suppose that $\tilde{x}_{a-1} \geq m + s - 1$. Consider the set of domain values $\{m, m + 1, \ldots, m+s-2\}$ which has cardinality $s-1$. The number of variables in $A \cup B - \{x_{a-1}, x_a\}$ is $s-2$, and since $\tilde{x}_a \geq \tilde{x}_{a-1} \geq m+s-1$ there is some $d \in \{m, m+1, \ldots, m+s-2\}$ such that no variable in the entire alldifferent system is assigned to d. Therefore, defining $\hat{x} = \tilde{x}$ except that $\tilde{x}_{a-1} = d$ yields a feasible integer point satisfying $c\hat{x} < c\tilde{x}$ contradicting that $\tilde{x}$ minimizes cx.

Now suppose that $m + a - 1 \leq \tilde{x}_{a-1} \leq m + s - 2$. Then $\exists d \in \{m, m+1, \ldots, m+a-2\}$ such that $\tilde{x}_i \neq d, \forall i \in A$. If there also does not exists a $\tilde{x}_i = d$, for $i \in B$, then defining $\hat{x} = \tilde{x}$ except that $\hat{x}_{a-1} = d$ yields a feasible solution with $c\hat{x} < c\tilde{x}$ contradicting that $\tilde{x}$ minimizes cx. If there does exists some $k \in B$ such that $\tilde{x}_k = d$, switching the domain values assigned to x_{a-1} and x_k yields a feasible solution with an even smaller value of cx, again contradicting that $\tilde{x}$ minimizes cx. □

Lemma 28 $\tilde{x}_{a+2} = m + a$ *and therefore for* $i = 1, 2, \ldots, b-1, \tilde{x}_i = m + a + i - 1$.

Proof. Follows similarly to the proof of lemma 27 □

Now, from lemma 27 and lemma 28 we have

$$a \sum_{i \in A} \tilde{x}_i - b \sum_{i \in B} \tilde{x}_i = a\tilde{x}_a - b\tilde{x}_{a+1} + aX(m, m+a-2) - bX(m+a, m+s-2).$$

This, together with inequality (6.62), implies that

$$a\tilde{x}_a - b\tilde{x}_{a+1} \leq a(m+s-1) - b(m+a-1) - 1$$

Also from lemma 27 and lemma 28 we have that $\tilde{x}_a \geq m+a-1$ and $\tilde{x}_{a+1} \leq m+a-1$. It is clear that since $\tilde{x}$ minimizes cx we must have either $\tilde{x}_a = m+a-1$ or $\tilde{x}_{a+1} = m+a-1$, for if not, then $\tilde{x}_a > m+a-1$ and we can define $\hat{x} = \tilde{x}$ except that $\hat{x}_a = m+a-1$ to yield a feasible integer point satisfying $c\hat{x} < c\tilde{x}$. Therefore only two cases to consider.

If $\tilde{x}_{a+1} = m+a-1$ then $\tilde{x}_a \geq m+s-1$ because all values in the set $\{m, m+1, \ldots, m+s-2\}$ are taken by some $\tilde{x}_i, i \in A \cup B - \{a\}$. Actually, since the coefficient of x_a is positive in cx it must be that $\tilde{x}_a = m+s-1$. But then $a(m+s-1) - b(m+a-1) = a\tilde{x}_a - b\tilde{x}_{a+1} \leq a(m+s-1) - b(m+a-1) - 1$, yielding a contradiction.

If $\tilde{x}_a = m+a-1$ then $\tilde{x}_{a+1} \leq m-1$ because all values in the set $\{m, m+1, \ldots, m+s-2\}$ are taken by some $\tilde{x}_i, i \in A \cup B - \{a+1\}$. Actually, since the coefficient of x_{a+1} is negative in cx it must be that $\tilde{x}_{a+1} = m-1$. But then $a(m+a-1) - b(m-1) = a\tilde{x}_a - b\tilde{x}_{a+1} \leq a(m+s-1) - b(m+a-1) - 1$, which upon simplification yields $a^2 \leq a^2 - 1$, a contradiction.

This concludes the proof that $cx \geq b_1$ is a valid inequality.

6.7.2 Facet Proof

Theorem 21 *Inequality (6.56) is facet-defining.*

Proof. Let $F = \{x \in P_I : cx = b_1\}$ and suppose that $\mu x = \mu_0$ is satisfied by all $x \in F$. We show that there exists a λ such that

$$\mu_i = \begin{cases} a\lambda & : \text{ for } i \in A \\ -b\lambda & : \text{ for } i \in B \end{cases}$$

and

$$\mu_i^j = \begin{cases} as\lambda & : \text{ for } i \in A, j \in A_i \\ -bs\lambda & : \text{ for } i \in B, j \in B_i \end{cases}$$

and $\mu_0 = b_1\lambda$, thereby proving that (6.56) is facet defining. Note that for simplicity, we assume that $H = A \cup B$ and that

$$T_i = \begin{cases} A_i & : \text{ for } i \in A \\ B_i & : \text{ for } i \in B \end{cases}$$

and for the more general situation where either some variable in the handle is not is $A \cup B$ or some variables in the teeth are not in some A_i or B_i the proof follows similarly.

Without loss of generality, suppose that $A = \{x_1, x_2, \ldots, x_a\}$ and that $B = H - A = \{x_{a+1}, \ldots, x_s\}$, and that $T_i = \{x_i^1, x_i^2, \ldots, x_i^m\}$ for all $i \in H$. Let $\tilde{x}$ be defined by

$$\tilde{x}_i = \begin{cases} m + i - 1 & : \text{ for } i = 1, 2, \ldots, a-1 \\ m + s - 1 & : \text{ for } i = a \\ m + i - 2 & : \text{ for } i = a+1, a+2, \ldots, a+b = |H|, \end{cases}$$

and

$$\tilde{x}_i^j = \begin{cases} j - 1 & : \text{ for } i \in A, j \in A_i \\ j + m + s - 2 & : \text{ for } i \in B, j \in B_i. \end{cases}$$

$\tilde{x} \in F$. Define $\hat{x}$ identical to $\tilde{x}$ except that $\hat{x}_1 = m + 1$ and $\hat{x}_2 = m$. $\hat{x} \in F$. Therefore, $\mu\tilde{x} = \mu\hat{x}$ and after cancelling common terms, we get $\mu_1 = \mu_2$. Similarly we can pick any two variables $x_i, x_k \in A$ and find $\mu_i = \mu_k$. Let λ be defined such that $a\lambda$ is this common value.

Now, define $\hat{x}$ identical to $\tilde{x}$ except that $\hat{x}_{a+1} = m + a$ and $\hat{x}_{a+2} = m + a - 1$. Again, $\hat{x} \in F$ and so $\mu\tilde{x} = \mu\hat{x}$ and after cancelling common terms, we get $\mu_{a+1} = \mu_{a+2}$. Similarly we can pick any two variables $x_i, x_k \in B$ and find $\mu_i = \mu_k$. Let λ' be this common value.

Now, define $\hat{x}$ identical to $\tilde{x}$ except that $\hat{x}_a = m + a - 1$ and $\hat{x}_{a+1} = m - 1$. Again, $\hat{x} \in F$ and so $\mu\tilde{x} = \mu\hat{x}$ and after cancelling common terms, we get $\mu_a(m+s-1) + \mu_{a+1}(m+a-1) = \mu_a(m+a-1) + \mu_{a+1}(m-1)$. Substituting in $s = a+b$ yields $b\mu_a = a\mu_{a+1}$ and so $\lambda' = -b\lambda$. Summarizing, we now have $\mu_i = a\lambda, \forall i \in A$ and $\mu_i = -b\lambda, \forall i \in B$.

Now, define $\bar{x}$ identical to $\tilde{x}$ except that $\bar{x}_a = m - 1$ and $\bar{x}_a^m = m$. It can be verified that $\bar{x} \in F$ and so $\mu\tilde{x} = \mu\bar{x}$. Cancelling like terms yields $\mu_a(m + s - 1) + \mu_a^m(m - 1) = \mu_a(m - 1) + \mu_a^m(m)$ which implies that $\mu_a^m = s\mu_a = as\lambda$. In addition, this change can be made $\forall i \in A$ and $\forall j \in A_i$. This is because we can switch the values assigned to $\tilde{x}_i$ and $\tilde{x}_k$

for any two $i, k \in A$ and for a fixed i we can switch the values of $\tilde{x}_i^j$ and $\tilde{x}_i^l$ for any $k, l \in A_i$. Hence, $\forall i \in A, \forall j \in A_i, \mu_i^j = as\lambda$, as desired.

Now, define $\bar{x}$ identical to $\hat{x}$ (as defined two paragraphs above) except that $\bar{x}_{a+1} = m + s - 1$ and $\bar{x}_{a+1}^1 = m + s - 2$. It can be verified that $\bar{x} \in F$ and so $\mu\hat{x} = \mu\bar{x}$. Cancelling like terms yields $\mu_{a+1}(m-1) + \mu_{a+1}^1(m+s-1) = \mu_{a+1}(m+s-1) + \mu_{a+1}^1(m+s-2)$ which implies that $\mu_{a+1}^1 = s\mu_{a+1} = -bs\lambda$. In addition, this change can be made $\forall i \in B$ and $\forall j \in B_i$. This is because we can switch the values assigned to $\tilde{x}_i$ and $\tilde{x}_k$ for any two $i, k \in B$ and for a fixed i we can switch the values of $\tilde{x}_i^j$ and $\tilde{x}_i^l$ for any $k, l \in B_i$. Hence, $\forall i \in B, \forall j \in B_i, \mu_i^j = -bs\lambda$, as desired.

Finally, plugging in any point, for example $\tilde{x}$, which is in F into $ux = \mu_0$ yields $\mu_0 = b_1\lambda$, concluding the proof that $cx \geq b_1$ is facet defining. □

6.7.3 0-1 Cut and $z-$cut

As discussed in the context of cycles, in Section 6.6.4, the comb inequalities presented here (and in [47, 48, 53]) can be mapped into 0-1 space to strengthen the continuous relaxation (6.30). In addition, $z-$cuts can be obtained from the $x-$cut (6.56) as described in Theorem 18. We omit the details for brevity, and stress that the purpose of displaying the facet-defining inequalities (6.56) was to display how the result in Theorem 15 can manifest itself.

6.8 Webs

We next study cuts that arise from webs. A web $W(q, r)$ is a graph in which vertices can be arranged cyclically so that the edges connect pairs of vertices separated by a distance of at least r on the cycle. More formally, given that $q \geq 2r + 1$ and $r \geq 1$, a web $W(q, r)$ is a graph on vertices $0, \ldots, q-1$ whose edges are all (i, i') such that $0 \leq i \leq q - r - 1$ and $r \leq i' - i \leq q - r$. Thus $W(q, 1)$ is a clique. When q is odd, $W(q, \frac{q-1}{2})$ is an odd hole, and $W(q, 2)$ is an odd anti-hole (the complement of an odd hole). Figure 6.4 illustrates $W(7, 2)$.

We will focus on webs for which q and r are mutually prime. Odd holes and odd anti-holes are special cases. Such webs give rise to 0-1 finite-domain cuts that provide tighter

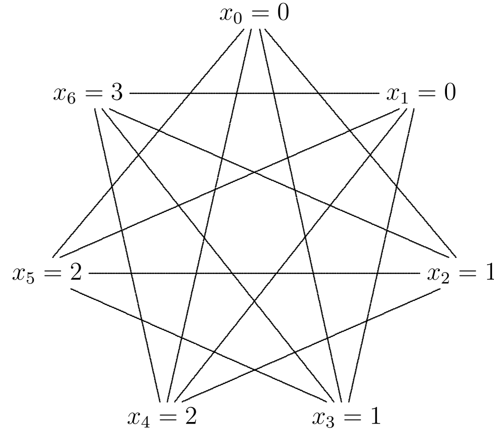


Figure 6.4: Web $W(7, 2)$, which is an odd antihole. Variables connected by an edge appear in a common alldiff constraint. A feasible solution is shown.

bounds than known 0-1 cuts.

6.8.1 Facet-Defining Inequalities

Theorem 22 *Let vertices* $0, \ldots, q-1$ *of* G *induce a web* $W(q, r)$ *, where* q *and* r *are mutually prime. The following inequality is facet-defining for (6.1):*

$$\sum_{i=0}^{q-1} x_i \geq \gamma(q, r) \tag{6.63}$$

where

$$\gamma(q, r) = r \sum_{j=0}^{t-1} v_j + (q - tr)v_t$$

and $t = \lfloor q/r \rfloor$*.*

Proof. To show that (6.63) is valid, we observe that each color can be used at most r times. This is because if any set of r vertices receive color j, no two of these vertices can be separated by distance of r or more in the cycle, because any such pair of vertices are connected. The vertices must therefore be adjacent. Because every other vertex is connected to one of them, no other vertex can receive color j, and no more than r vertices can receive color j. This means that at least $t + 1$ colors must be used. Thus

$$\sum_{i=0}^{q-1} x_i \geq r(v_0 + v_1 + \cdots + v_{t-1}) + dv_t = \gamma(q, r)$$

where the coefficient $d = q - tr$ is number of vertices remaining after assigning each the colors $v_0, \ldots, v_{t-1}$ to r vertices.

To show that (6.63) is facet defining, let

$$F = \{x \text{ feasible for (6.1)} \mid (x_0, \ldots, x_{q-1}) \text{ satisfies (6.63) at equality}\}$$

It suffices to show that if $\mu x \geq \mu_n$ holds for all $x \in F$, then there is a scalar $\lambda > 0$ such that

$$\mu_i = \begin{cases} \lambda & \text{for } i = 0, \ldots, q-1 \\ \gamma(q,r)\lambda & \text{for } i = n \\ 0 & \text{otherwise} \end{cases} \tag{6.64}$$

The partial solution

$$(\bar{x}_0, \ldots, \bar{x}_{q-1}) = \Big(\underbrace{v_0, \ldots, v_0}_{r}, \underbrace{v_1, \ldots, v_1}_{r}, \ldots, \underbrace{v_{t-1}, \ldots, v_{t-1}}_{r}, \underbrace{v_t, \ldots, v_t}_{d}\Big) \tag{6.65}$$

is clearly feasible and satisfies (6.63) at equality. We construct a partial solution $(\tilde{x}_0, \ldots, \tilde{x}_{q-1})$ from (6.65) by swapping the color assignment of the vertices receiving color v_t with that of the last d vertices receiving color v_0. That is, we let

$$\tilde{x}_i = \begin{cases} v_t & \text{if } i \in \{r-d, \ldots, r-1\} \\ v_0 & \text{if } i \in \{q-d, \ldots, q-1\} \\ \bar{x}_i & \text{otherwise} \end{cases}$$

Note that $(\tilde{x}_0, \ldots, \tilde{x}_{q-1})$ is feasible, because colors $v_1, \ldots, v_{t-1}$ are assigned to r adjacent vertices as before, color v_0 is assigned to the r adjacent vertices $q-d, \ldots, q-1, 0, \ldots, r-d-1$, and color v_t is assigned to the remaining adjacent vertices $r-d, \ldots, r-1$. Extend the two partial solutions to feasible solutions $\bar{x}$ and $\tilde{x}$ of (6.1). Because $\bar{x}$ and $\tilde{x}$ satisfy (6.63) at equality, we have $\mu\bar{x} = \mu\tilde{x}$, which yields

$$\mu_{r-d} + \cdots + \mu_{r-1} = \mu_{q-d} + \cdots + \mu_{q-1}$$

By symmetry, we conclude

$$\mu_i + \cdots + \mu_{(i+d-1) \bmod q} = \mu_{(i-r) \bmod q} + \cdots + \mu_{(i-r+d-1) \bmod q}$$

for $i = 0, \ldots, q-1$. Because q and r are mutually prime, this implies that the sums $\mu_i + \cdots + \mu_{(i+d-1) \bmod q}$ are equal for all i. Thus, in particular, they are equal for i and $i+1$,

which yields $\mu_i = \mu_{(i+d) \bmod q}$ for all i. Because $d \neq q$, this implies that $\mu_0 = \cdots = \mu_{q-1}$ and

$$\mu_i = \lambda, \;\; i = 0, \ldots, q-1 \tag{6.66}$$

for some $\lambda > 0$.

Define $\bar{x}'$ by letting $\bar{x}' = \bar{x}$ except that for an arbitrary $\ell \notin \{0, \ldots, q-1\}$, $\bar{x}'_\ell$ is assigned a value that does not appear in the tuple $\bar{x}$. Since $\bar{x}, \bar{x}' \in F$, we have $\mu\bar{x} = \mu\bar{x}'$, which implies

$$\mu_i = 0, \; i \in V \setminus \{0, \ldots, q-1\} \tag{6.67}$$

Because $\mu\bar{x} = \mu_n$, we have from (6.66) that $\mu_n = \gamma(q,r)\lambda$. This, (6.66) and (6.67) imply (6.65). $\square$

For domain D_δ with $\delta > 0$, the cut (6.63) is

$$\sum_{i=0}^{q-1} x_i \geq \left(q - \tfrac{1}{2}(t+1)r\right) t\delta$$

For an odd antihole $W(q,2)$ with domain D_δ, the cut simplifies to

$$\sum_{i=0}^{q-1} x_i \geq \tfrac{1}{4}(q-1)^2\delta \tag{6.68}$$

Theorems 18 and 22 imply

Corollary 2 *If the graph coloring problem (6.1) is defined on a graph in which vertex sets V_r induce a web $W(q,r)$, where q and r are mutually prime, and each x_i has domain D_δ with $\delta > 0$, then*

$$z \geq \frac{1}{q}\sum_{i=1}^{q} x_i + \left(1 - \frac{r}{2q}(t+1)\right) t\delta \tag{6.69}$$

is facet defining for (6.1), where $t = \lfloor q/r \rfloor$.

For example, the antihole of Fig. 6.4 gives rise to the facet-defining cuts

$$\sum_{i=0}^{6} x_i \geq 9, \quad z \geq \tfrac{1}{7}\sum_{i=0}^{6} x_i + \tfrac{9}{7}$$

6.8.2 Mapping to 0-1 Cuts

If each x_i has domain $D_1 = \{0, 1, \ldots, n-1\}$ the x-cut (6.17) maps into the cut

$$\sum_{i=0}^{q-1}\sum_{j=1}^{q-1} j y_{ij} \geq \left(q - \tfrac{1}{2}(t+1)r\right) t \tag{6.70}$$

which is valid by Lemma 9. The z-cut (6.69) maps into

$$\sum_{j=0}^{q-1} w_j - 1 \geq \frac{1}{q}\sum_{i=0}^{q-1}\sum_{j=1}^{q-1} j y_{ij} + \left(1 - \frac{r}{2q}(t+1)\right) t \tag{6.71}$$

We wish to compare these cuts with known cuts for webs. Facet-defining web cuts for a 0-1 model of the coloring problem are given in [59]. These cuts are defined in a space in which the variables correspond to edges and colorings correspond to "admissible star partitions" of the graph. However, the cuts are based on the fact that at most r vertices can be assigned any given color, and we can write analogous web cuts in the y_{ij}-space:

$$\sum_{i=0}^{q-1} y_{ij} \leq r \cdot w_j, \quad \text{all } j \tag{6.72}$$

As with odd cycle cuts, the finite-domain web cuts provide a tighter bound than known 0-1 cuts. For example, for an antihole $W(7, 2)$, seven 0-1 web cuts (6.72) give a bound of 3.5, while the two finite-domain cuts provide a bound of 3.5714. For the web $W(8, 3)$, eight 0-1 cuts give the bound 2.6667, while the two finite-domain cuts yield the bound 2.75. This if an existing separation algorithm identifies 0-1 cuts for a given web, they can be replaced (at no additional cost) with two finite-domain cuts that yield a tighter bound. Further comparisons appear in Section 6.11.

6.9 Paths

Paths present an interesting case because they give rise to finite-domain cuts that are redundant in the 0-1 model. That is, they have no effect on the bound when the problem consists entirely of a path system. However, a few finite-domain cuts may replace a large number of inequalities in a more complex coloring problem, allowing a substantial reduction in the size of the 0-1 model. In addition, path cuts are useful in a finite-domain model of the problem.

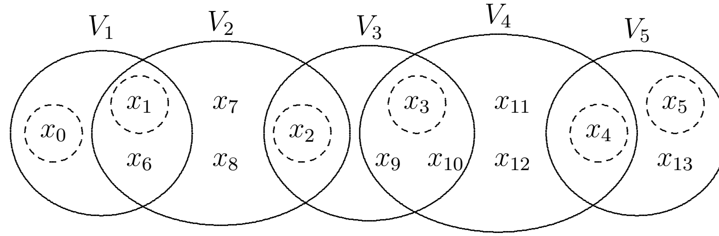


Figure 6.5: A path with $q = 5$. The variables in dashed circles appear in one possible valid cut.

A path is a subgraph of G induced by the vertices in subsets $V_0, \ldots, V_q$ of V, provided the subgraph induced by each V_k is a clique, and only adjacent V_k's overlap. That is,

$$V_k \cap V_\ell = \begin{cases} S_k & \text{if } k+1 = \ell \\ \emptyset & \text{otherwise} \end{cases}$$

where $S_k \neq \emptyset$.

6.9.1 Facet-defining Inequalities

Facet-defining inequalities can be obtained by selecting one variable from each overlap S_k. Valid cuts can be obtained if two or more variables are selected, as with cycles, but they are redundant of the single-variable cuts.

Lemma 29 *Let $V_0, \ldots, V_q$ be a path, and let $x_0 \in V_0 \setminus V_1$, $x_q \in V_q \setminus V_{q-1}$, and $x_i \in S_i$ for $i = 1, \ldots, q-1$. If q is odd, the following inequality is valid for (6.1):*

$$(v_2 - v_0)(x_0 + x_q) + (v_1 - v_0)\sum_{i=1}^{q-1} x_i \geq \phi(q) \tag{6.73}$$

where

$$\phi(q) = \left(\frac{q-1}{2}(v_1 - v_0) + v_2 - v_0\right)(v_0 + v_1)$$

Proof. Suppose to the contrary (6.73) is not satisfied, which implies

$$(v_1 - v_0)\sum_{i=1}^{q-1} x_i < \phi(q) - (v_2 - v_0)(x_0 + x_q) \tag{6.74}$$

Adjacent variables in the list $x_0, \ldots, x_q$ must take distinct values. So we have $x_i + x_{i+1} \geq v_0 + v_1$ for $i = 1, 3, \ldots, q-2$. Summing these, we obtain

$$\sum_{i=1}^{q-1} x_i \geq \frac{q-1}{2}(v_0 + v_1)$$

This and (6.74) imply

$$\frac{q-1}{2}(v_1 - v_0)(v_0 + v_1) < \left(\frac{q-1}{2}(v_1 - v_0) + v_2 - v_0\right)(v_0 + v_1) - (v_2 - v_0)(x_0 + x_q)$$

which implies $x_0 + x_q < v_0 + v_1$. This is possible only if $x_0 = x_q = v_0$, because $v_1 > v_0 \geq 0$. Thus $x_1, x_{q-1} \neq v_0$, which means that at most $(q-3)/2$ of the variables $x_1, \ldots, x_{q-1}$ can take the value v_0, and at least one variable must take a value larger than v_1. So

$$(v_1 - v_0)\sum_{i=1}^{q-1} x_i \geq (v_1 - v_0)\left(\frac{q-3}{2}v_0 + \frac{q-1}{2}v_1 + v_2\right) = \phi(q) + (v_1 - v_0)(v_2 - v_0)$$

But this is inconsistent with (6.74) because $x_0 + x_q \geq 0$ and $(v_1 - v_0)(v_2 - v_0) > 0$. □

If each x_i has domain D_δ for $\delta > 0$, the cut (6.73) is

$$2(x_0 + x_q) + \sum_{i=1}^{q-1} x_i \geq \tfrac{1}{2}(q+3)\delta \tag{6.75}$$

Theorem 23 *If the graph coloring problem (6.1) is defined on a graph in which vertex sets $V_0, \ldots, V_q$ induce a path, where q is odd, then inequality (6.73) is facet defining for (6.1).*

Proof. Let

$$F = \{x \text{ feasible for (6.1)} \mid (x_0, \ldots, x_q) \text{ satisfies (6.73) at equality}\}$$

It suffices to show that any equation $\mu x \geq \mu_{n+1}$ that holds for all $x \in F$ is a positive scalar multiple of (6.73). It therefore suffices to show that there is a scalar $\lambda > 0$ such that

$$\mu_i = \begin{cases} (v_1 - v_0)\lambda & \text{for } i = 1, \ldots, q-1 \\ (v_2 - v_0)\lambda & \text{for } i = 0, q \\ \phi(q)\lambda & \text{for } i = n+1 \\ 0 & \text{otherwise} \end{cases} \tag{6.76}$$

The following partial solutions are feasible for (6.1):

$$(\bar{x}_0, \ldots, \bar{x}_q) = (v_1, v_0, v_1, v_0, v_1, v_0, v_1, \ldots, v_1, v_0)$$

$$(\hat{x}_0, \ldots, \hat{x}_q) = (v_0, v_2, v_1, v_0, v_1, v_0, v_1, \ldots, v_1, v_0)$$

$$(\tilde{x}_0, \ldots, \tilde{x}_q) = (v_0, v_1, v_2, v_0, v_1, v_0, v_1, \ldots, v_1, v_0)$$

They can be extended to complete solutions $\bar{x}, \hat{x}, \tilde{x}$ with

$$\bar{x}_i = \hat{x}_i = \tilde{x}_i \text{ for } i \notin \{0, \ldots, q\}$$

in the manner described above. Because $\hat{x}$ and $\tilde{x}$ satisfy (6.73) at equality, they satisfy $\mu x = \mu_{n+1}$, and we have $\mu\hat{x} = \mu\tilde{x} = \mu_{n+1}$. This implies $\mu(\hat{x} - \tilde{x}) = 0$, and therefore $\mu_1 = \mu_2$. By symmetry, we have

$$\mu_1 = \cdots = \mu_{q-1} \tag{6.77}$$

Also $\bar{x}$ satisfies (6.73) at equality, and we have $\mu\bar{x} = \mu\hat{x} = \mu_{n+1}$. This implies $(v_1 - v_0)\mu_0 = (v_2 - v_0)\mu_1$. Thus by (6.77) and symmetry

$$(v_1 - v_0)\mu_0 = (v_1 - v_0)\mu_q = (v_2 - v_0)\mu_i,\ i = 1, \ldots, q-1 \tag{6.78}$$

Define $\bar{x}'$ by letting $\bar{x}' = \bar{x}$ except that for an arbitrary $\ell \notin \{0, \ldots, q\}$, $\bar{x}'_\ell$ is assigned a value that does not appear in the tuple $\bar{x}$. Since $\bar{x}$, $\bar{x}'$ are feasible and satisfy (6.73) at equality, we have $\mu\bar{x} = \mu\bar{x}'$. This and $\bar{x}_\ell \neq \bar{x}'_\ell$ imply

$$\mu_i = 0,\ i \in V \setminus \{0, \ldots, q\} \tag{6.79}$$

Finally, (6.78) implies that for some $\lambda > 0$,

$$\mu_0 = \mu_q = (v_2 - v_0)\lambda \text{ and } \mu_i = (v_1 - v_0)\lambda,\ \ i = 1, \ldots, q-1 \tag{6.80}$$

Because $\mu\bar{x} = \mu_{n+1}$, we have from (6.80) that $\mu_{n+1} = \phi(q)\lambda$. This, (6.79), and (6.80) imply (6.76). □

Theorems 18 and 23 imply

Corollary 3 *If the graph coloring problem (6.1) is defined on a graph in which vertex sets $V_0, \ldots, V_q$ induce a path, where q is odd and each x_i has domain D_δ for $\delta > 0$, then*

$$z \geq \frac{1}{q+3}\left(2(x_0 + x_q) + \sum_{i=1}^{q-1} x_i\right) + \frac{\delta}{2} \tag{6.81}$$

is facet defining for (6.1).

If the colors are $0, 1, \ldots, 4$, the cuts for the path in Fig. 6.5 are

$$2(x_0 + x_5) + \sum_{i=1}^{4} x_i \geq 4, \quad z \geq \tfrac{1}{4}(x_0 + x_5) + \tfrac{1}{8}\sum_{i=1}^{4} x_i + \tfrac{1}{2}$$

6.9.2 Mapping to 0-1 Cuts

Assuming domain D_1, the x-cut (6.73) and z-cut (6.81) respectively map to 0-1 space as follows:

$$2\sum_{j=1}^{q-1} j(y_{0j} + y_{qj}) + \sum_{i=1}^{q-1}\sum_{j=1}^{q-1} jy_{ij} \geq \tfrac{1}{2}(q+3) \tag{6.82}$$

$$\sum_{j=1}^{q} w_j - 1 \geq \frac{2}{q+3}\sum_{j=1}^{q-1} j(y_{0j} + y_{qj}) + \frac{1}{q+3}\sum_{i=1}^{q-1}\sum_{j=1}^{q-1} jy_{ij} + \tfrac{1}{2} \tag{6.83}$$

To simplify notation, we suppose in this section that each $S_k = \{k\}$. The arguments are very similar for the more general case. Given this simplification, the continuous relaxation of the 0-1 path model is

$$\begin{aligned}
&\sum_{j=0}^{q-1} y_{ij} = 1, \;\; i = 0, \ldots, q && (r_i)\\
&y_{ij} + y_{i+1,j} \leq w_j, \;\; i = 0, \ldots, q-1, \; j = 0, \ldots, q-1 && (s_{ij})\\
&0 \leq y_{ij}, w_j \leq 1, \;\; \text{all } i, j &&
\end{aligned} \tag{6.84}$$

It can be shown that if the w_js are treated as constants equal to 1, the constraint matrix of this model is totally unimodular. The 0-1 x-cut (6.82) is therefore redundant of (6.84).

We cannot use a similar argument to show that the 0-1 z-cut (6.83) is redundant, because the full model (6.84) with w_js is not totally unimodular. In fact, the 0-1 z-cut is not redundant, because it is not implied by (6.84) augmented with symmetry-breaking constraints $w_i \geq w_{i+1}$. The following (extreme point) solution satisfies (6.84) with $q = 3$ but violates the cut because it results in a left-hand side of $1\frac{2}{7}$ and a right-hand side of $2\frac{5}{14}$.

$$y = \begin{bmatrix} 0 & \frac{4}{7} & 0 & \frac{3}{7} \\ \frac{2}{7} & 0 & \frac{4}{7} & \frac{1}{7} \\ \frac{2}{7} & \frac{4}{7} & 0 & \frac{1}{7} \\ 0 & 0 & \frac{4}{7} & \frac{3}{7} \end{bmatrix}, \qquad w = (\tfrac{4}{7}, \tfrac{4}{7}, \tfrac{4}{7}, \tfrac{4}{7})$$

The 0-1 z-cut has no effect on the bound in a problem consisting entirely of a path system, because the relaxation (6.84) already implies the optimal bound of 2. The sum of the negated constraints (r_{q-1}) and (r_q) with constraints (s_{q-1}, j) for $j = 0, \ldots, q$ yields the bound $\sum_j w_j \geq 2$.

On the other hand, if the 0-1 x-cut and z-cut are present, the 0-1 model yields the same bound of 2 even if all of the constraints (s_{ij}) are dropped. This suggests that for a more complex problem, a few finite-domain cuts could replace many constraints in the 0-1 model with little effect on the resulting bound. We leave this issue to future research.

6.10 Intersecting Systems

Finally, we identify cuts for more general structures for which there are apparently no previously known 0-1 cuts. They illustrate how a finite-domain perspective can lead to facet-defining cuts for a wide variety of situations.

We define an *intersecting system* to be a family of clique-inducing vertex sets such that (a) at least one vertex belongs to all the sets, (b) every set contains at least one vertex that belongs to no other set, and (c) every set excludes at least one vertex that belongs to all the other sets. Formally, an intersecting system consists of a family $V_1, \ldots, V_q$ of clique-inducing vertex sets such that the following sets are nonempty:

$$U = \bigcap_{k=1}^{q} V_k$$

$$A_k = V_k \setminus \bigcup_{\ell \neq k} V_\ell, \quad k = 1, \ldots, q$$

$$S_k = (\bigcap_{\ell \neq k} V_\ell) \setminus V_k, \quad k = 1, \ldots, q$$

Figure 6.6 illustrates an intersecting system with $q = 3$.

6.10.1 Valid Inequalities

We first establish two families of valid inequalities for an intersecting system.

Lemma 30 *Given an intersecting system $V_1, \ldots, V_q$, suppose $x_k \in A_k$, $\bar{S}_k \subset S_k$, and $|\bar{S}_k| = s$ for $k = 1, \ldots, q$. Then if $A = \bigcup_{k=1}^{q} \{x_k\}$, $S = \bigcup_k \bar{S}_k$, $\bar{U} \subset U$, and $|\bar{U}| = u$, the*

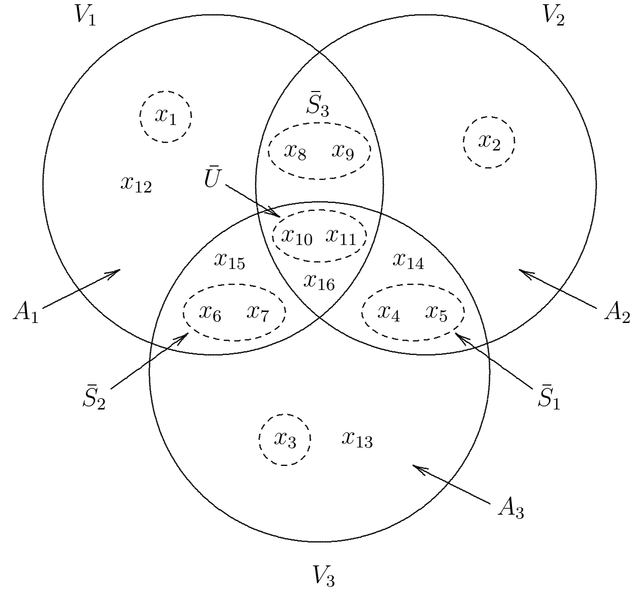


Figure 6.6: An intersecting system with $q = 3$. The variables x_1, x_2, x_3 and sets $\bar{S}_1, \bar{S}_2, \bar{S}_3, \bar{U}$ provide the basis for two possible facet-defining cuts with $s = u = 2$.

following inequalities are valid for (6.1):

$$c_1 \sum_{i \in A} x_i + c_2 \sum_{i \in S \cup \bar{U}} x_i \geq \psi(q, s, u) \tag{6.85}$$

$$d_1 \sum_{i \in A} x_i + d_2 \sum_{i \in S} x_i + d_3 \sum_{i \in \bar{U}} x_i \geq \omega(q, s, u) \tag{6.86}$$

where

$$
\begin{aligned}
c_1 &= v_{qs+u} - v_0 \\
c_2 &= \sum_{i=1}^{q-1} v_i - (q-1)v_0 \\
\psi(q,s,u) &= c_1 q v_0 + c_2 \sum_{i=1}^{qs+u} v_i \\
d_1 &= v_{qs+u} - v_u \\
d_2 &= \sum_{i=1}^{q+u-1} v_i - q v_u \\
d_3 &= (v_{qs+u} - v_u) q \\
\omega(q,s,u) &= d_1 q v_u + d_2 \sum_{i=u+1}^{qs+u} v_i + d_3 \sum_{i=1}^{u-1} v_i
\end{aligned}
$$

Proof. We prove the validity of (6.85) only, as the proof for (6.86) is similar. Write (6.85) as $ax \geq \psi(q,s,u)$, and let $\bar{x}$ be any feasible solution of (6.1) that minimizes ax. It suffices to show that $a\bar{x} = \psi(q,s,u)$.

We first show that $\bar{x}_i \leq v_{qs+u}$ for all $i \in S \cup \bar{U}$. Suppose to the contrary that $\bar{x}_j = w > v_{qs+u}$ for some $j \in S \cup \bar{U}$. Then at least two values $w_1, w_2 \in \{v_0, \ldots, v_{qs+u}\}$ fail to appear in $\{\bar{x}_i \mid i \in S \cup \bar{U} \setminus \{j\}\}$, because the latter set has $qs+u-1$ elements. Suppose $w_1 < w_2$. Then w_2 does not appear in $\{\bar{x}_i \mid i \in A\}$, because if it did, we could define a feasible solution $\hat{x}$ that is identical to $\bar{x}$ except that $\hat{x}_k = w_1$ for some $k \in A$, and we would have $a\hat{x} < a\bar{x}$, contrary to the assumption that $\bar{x}$ minimizes ax. So w_2 appears nowhere in $\{\bar{x}_i \mid i = 1, \ldots q\}$. This means we can define a feasible solution $\tilde{x}$ that is identical to $\bar{x}$ except that $\tilde{x}_j = w_2$ (rather than w), and we have $a\tilde{x} < a\bar{x}$, contrary to assumption.

Now since $\bar{x}_i \leq v_{qs+u}$ for all $i \in S \cup \bar{U}$, and all the variables in $S \cup \bar{U}$ must take different values, the set $\{\bar{x}_i \mid i \in S \cup \bar{U}\}$ must be $\{v_0, \ldots, v_{qs+u}\} \setminus \{v_r\}$ for some $r \in \{0, \ldots, qs+u\}$. This implies

$$\min_x \{ax\} = c_1 \sum_{i \in A} \bar{x}_i + c_2 \left(\sum_{i=0}^{qs+u} v_i - v_r \right) \tag{6.87}$$

We consider two cases.

Case 1: $r \geq q$. Here all the values in $\{v_0, \ldots, v_{q-1}\}$ appear in $\{\bar{x}_i \mid i \in S \cup \bar{U}\}$. So each

value in $\{v_0,\ldots,v_{q-1}\}$ can appear at most once in $\{\bar{x}_j \mid j\in A\}$. Thus from (6.87) we have

$$\min_x \{ax \mid r\geq q\} = c_1\sum_{i=0}^{q-1} v_i + c_2\left(\sum_{i=0}^{qs+u} v_i - v_r\right) = c_1\sum_{i=0}^{q-1} v_i + c_2\sum_{i=0}^{qs+u-1} v_i = \psi(q,s,u)$$

where the second equality is due to the fact that we must have $r = qs+u$. Otherwise, $\bar{x}_i = v_{qs+u} > v_r$ for some $i\in S\cup\bar{U}$, and we can create a feasible solution $\hat{x}$ that is identical to $\bar{x}$ except that $\hat{x_i} = v_r$, and we have $a\hat{x} < a\bar{x}$. The third equality follows algebraically from the definitions of c_1, c_2.

Case 2. $0\leq r\leq q-1$. We show that the minimum is obtained in this case only if $c_1\geq c_2$, and that the minimum is again $\psi(q,s,u)$. Because v_r does not appear in $\{\bar{x}_i \mid i\in S\cup\bar{U}\}$, no value greater than v_r appears in $\{\bar{x}_i \mid i\in A\}$. Furthermore, all the values in $\{v_0,\ldots,v_{r-1}\}$ appear in $\{\bar{x}_i \mid i\in S\cup\bar{U}\}$. Thus each value in $\{v_0,\ldots,v_{r-1}\}$ can appear at most once in $\{\bar{x}_j \mid j\in A\}$. The remaining $q-r$ variables in A can be assigned the value v_r, because it does not appear in $\{\bar{x}_i \mid i\in S\cup\bar{U}\}$. So, from (6.87) we have

$$\min_x\{ax \mid 0\leq r\leq q-1\} = c_1\left(\sum_{i=0}^{r-1} v_i + (q-r)v_r\right) + c_2\left(\sum_{i=0}^{qs+u} v_i - v_r\right)$$

If $c_1\geq c_2$, this expression is minimized by setting $r=0$, whereupon it reduces to $\psi(q,s,u)$. If $c_1 < c_2$, the minimum occurs when $r=q-1$, and the expression becomes

$$c_1\sum_{i=0}^{q-1} v_i + c_2\left(\sum_{i=1}^{qs+u} v_i - v_{q-1}\right) > c_1\sum_{i=0}^{q-1} v_i + c_2\sum_{i=0}^{qs+u-1} v_i = \psi(q,s,u)$$

Thus if $c_1 < c_2$, the minimum occurs in Case 1, and the lemma follows. □

When the domains are D_δ, the inequalities (6.85)–(6.86) become

$$\begin{aligned} &(qs+u)\sum_{i\in A} x_i + \tfrac{1}{2}q(q-1)\sum_{i\in S\cup\bar{U}} x_i \geq \psi(q,s,u) \\ &s\sum_{i\in A} x_i + \tfrac{1}{2}(q-1)\sum_{i\in S} x_i + qs\sum_{i\in\bar{U}} x_i \geq \omega(q,s,u)/q \end{aligned} \qquad (6.88)$$

where

$$\psi(s,q,u) = \tfrac{1}{4}q(q-1)(qs+u)(qs+u+1)\delta$$

$$\omega(s,q,u)/q = \tfrac{1}{4}qs(2u(u+1) + (q-1)(sq+2u+1))\delta$$

Given domain D_1, the cuts for the system in Fig. 6.6 are

$$8(x_1+x_2+x_3) + 3(x_4+\cdots+x_{11}) \geq 108$$

$$2(x_1+x_2+x_3) + (x_4+\cdots+x_9) + 6(x_{10}+x_{11}) \geq 51$$

6.10.2 Facet-Defining Inequalities

Theorem 24 *If $V_1, \ldots, V_q$ is an intersecting system, then inequalities (6.85) and (6.86) are facet defining.*

Proof. We prove that (6.85) is facet defining, as the proof for (6.86) is similar. We know that (6.85) is valid, by Lemma 30. Let

$$F = \{x \text{ feasible for } (6.1) \mid x \text{ satisfies } (6.85) \text{ at equality}\}$$

To show that (6.85) is facet defining, it suffices to show that if $\mu x \geq \mu_0$ holds for all $x \in F$, then there is a scalar $\lambda > 0$ such that

$$\mu_i = \begin{cases} \lambda c_1 & \text{for } i \in A \\ \lambda c_2 & \text{for } i \in S \cup \bar{U} \\ 0 & \text{for } i \in V \setminus (A \cup S \cup \bar{U}) \\ \lambda \psi(q, s, u) & \text{for } i = 0 \end{cases} \tag{6.89}$$

Let x^A be the tuple of variables x_j for $i \in A$, x^i the tuple of variables x_j for $j \in S_i$, and x^U the tuple of variables x_j for $j \in \bar{U}$. The partial solution

$$\begin{aligned} \bar{x}^A &= (v_0, \ldots, v_0) \\ \bar{x}^i &= (v_i, v_{i+q}, v_{i+2q}, \ldots, v_{i+(s-1)q}), \text{ for } i = 1, \ldots, q \\ \bar{x}^U &= (v_{sq+1}, \ldots, v_{sq+u}) \end{aligned}$$

is feasible and satisfies (6.85) at equality. As in previous proofs, it is straightforward to show that $\mu_i = 0$ for $i \in V \setminus (A \cup S \cup \bar{U})$. Also, by symmetry, μ_i and μ_j take the same value λ_A (or λ_S or λ_U) for any pair i, j in A (or S or $\bar{U}$). So if $x^S = (x^1, \ldots, x^q)$, the equation $\mu x = \mu_0$ reduces to

$$\lambda_A \sum_{i \in A} x_i + \lambda_S \sum_{i \in S} x_i + \lambda_U \sum_{i \in \bar{U}} x_i = \mu_0 \tag{6.90}$$

Now let $(\tilde{x}^A, \tilde{x}^S, \tilde{x}^U) = (\bar{x}^A, \bar{x}^S, \bar{x}^U)$ except that $\tilde{x}_j = \bar{x}_k$ for arbitrary $j \in S$, $k \in \bar{U}$. Extend these two partial solutions to feasible solutions $\bar{x}$, $\tilde{x}$ of (6.1). Since $\bar{x}, \tilde{x}$ satisfy (6.90), we have $\lambda_S = \lambda_U$. So (6.90) reduces to

$$\lambda_A \sum_{i \in A} x_i + \lambda_S \sum_{i \in S \cup \bar{U}} x_i = \mu_0 \tag{6.91}$$

By definition of $\bar{x}$, we have that $\bar{x}_j = v_q$ for some $j \in S_q$, and $\bar{x}_k = qs + u$ for some $k \in \bar{U}$. Let $(\hat{x}^A, \hat{x}^S, \hat{x}^U) = (\bar{x}^A, \bar{x}^S, \bar{x}^U)$ except that $\hat{x}_j = v_0$ and $\hat{x}_k = v_q$, and extend this partial solution to a feasible solution $\hat{x}$. Then since $\bar{x}, \hat{x}$ satisfy (6.91), we have

$$\lambda_A q v_0 + \lambda_S \sum_{i=1}^{qs+u} v_i = \lambda_A \sum_{i=1}^{q-1} v_i + \lambda_S \left(\sum_{i=1}^{q-1} v_i + v_0 + \sum_{i=q+1}^{qs+u-1} v_i + v_q \right)$$

This yields

$$\lambda_A \left(\sum_{i=1}^{q-1} x_i - (q-1)v_0 \right) = \lambda_S \left(v_{qs+u} - v_0 \right)$$

or $\lambda_A c_2 = \lambda_S c_1$. Thus $\mu_j/\mu_i = c_2/c_1$ for any $i \in A$ and any $j \in S \cup \bar{U}$. Also since $\mu \bar{x} = \mu_0$, we have

$$\mu_0/\mu_i = \sum_{i \in A} \bar{x}_i + \frac{c_2}{c_1} \sum_{i \in S \cup \bar{U}} \bar{x}_i = q v_0 + \frac{c_2}{c_1} \sum_{i=1}^{qs+u} v_i = \psi(q,s,u)/c_1$$

So there exists a λ satisfying (6.89). $\square$

6.10.3 Bounds on the Chromatic Number

Theorems 18 and 24 imply

Corollary 4 *If $V_1, \ldots, V_q$ is an intersecting system and each x_i has domain D_δ with $\delta > 0$, then the inequalities*

$$\begin{aligned} z &\geq \frac{2}{q(q+1)} \sum_{i \in A} x_i + \frac{q-1}{(q+1)(qs+u)} \sum_{i \in S \cup \bar{U}} x_i + \tfrac{1}{2} \frac{q-1}{q+1} (qs+u+1) \\ z &\geq \frac{2}{qh} \sum_{i \in A} x_i + \frac{q-1}{qsh} \sum_{i \in S} x_i + \frac{2}{h} \sum_{i \in \bar{U}} x_i + \frac{1}{2h} (2u(u+1) + (q-1)(qs+2u+1)) \end{aligned} \tag{6.92}$$

are facet defining for (6.1), where $h = q + 2u + 1$.

Given domain D_1, the bounds for the system in Fig. 6.6 are

$$z \geq \tfrac{1}{6}(x_1 + x_2 + x_3) + \tfrac{1}{16}(x_4 + \cdots + x_{11}) + \tfrac{9}{4}$$

$$z \geq \tfrac{1}{12}(x_1 + x_2 + x_3) + \tfrac{1}{24}(x_2 + \cdots + x_9) + \tfrac{1}{4}(x_{10} + x_{11}) + \tfrac{17}{8}$$

As with other cuts, the inequalities (6.85), (6.86), and (6.92) can be mapped to valid 0-1 inequalities by replacing each x_i with $\sum_j v_j y_{ij}$ and z with $\sum_j w_j - 1$.

6.10.4 Separation

A polynomial-time separation algorithm for (6.85), given domain D_1, is as follows. Let $V_1, \ldots, V_q$ define an intersecting system for which we wish to find a separating cut, and let $(\bar{x}, \bar{z})$ be a solution of the current continuous relaxation (perhaps mapped from the 0-1 model). For each $k = 1, \ldots, q$, let $\bar{x}_{a(i)} = \min_{i \in A}\{\bar{x}_i\}$ and $\bar{x}_{b(i)} = \max_{i \in A}\{\bar{x}_i\}$, and define the bijection $\pi_k : \{1, \ldots, |S_k|\} \to S_k$ such that $\bar{x}_{\pi_k(i)} \leq \bar{x}_{\pi_k(i')}$ whenever $i < i'$. Also define the bijection $\pi : \{1, \ldots, |U|\}$ such that $\bar{x}_{\pi(i)} \leq \bar{x}_{\pi_(i')}$ whenever $i < i'$. Then for $s = 1, \ldots, \min_k |S_k|$ and $u = 1, \ldots, |U|$, generate a separating x-cut

$$(qs+u)\sum_{i=1}^{q} x_{a(i)} + \tfrac{1}{2}q(q-1)\left(\sum_{k=1}^{q}\sum_{i=1}^{s} x_{\pi_k(i)} + \sum_{i=1}^{u} x_{\pi(i)}\right) \geq \tfrac{1}{4}q(q-1)(qs+u)(qs+u+1)$$

whenever $\bar{x}$ violates this inequality, and generate a separating z-cut

$$z \geq \frac{2}{q(q+1)}\sum_{i=1}^{q} x_{b(i)} + \frac{q-1}{(q+1)(qs+u)}\left(\sum_{k=1}^{q}\sum_{i=1}^{s} x_{|S_k|-i+1} + \sum_{i=1}^{u} x_{|U|-i+1}\right) + \tfrac{1}{2}\frac{q-1}{q+1}(qs+u+1)$$

whenever $(\bar{x}, \bar{z})$ violates this inequality.

This procedure obtains a separating cut whenever one exists. Separation for (6.86) is similar.

6.11 Computational Results

6.11.1 Cycles

We generated instances of the cycle problem parameterized by s and q. All of the overlap sets S_k have size s, and vertex set $V_k = S_k \cup S_{k+1}$ for $k = 1, \ldots, q-1$ (with $V_k = S_k \cup S_1$). For each instance, we solved the linear programming relaxation that minimizes $\sum_j w_j$ subject to (6.30) and various classes of cuts. Clique inequalities are always present.

We generated the instances indicated in Table 6.1, which shows the resulting bounds, the optimal value (chromatic number), and the number of odd hole cuts.

For $s = 1$, the table confirms that 0-1 x-cuts are redundant of odd hole cuts. However, the combination of one x-cut and one z-cut yields a tighter bound than n odd hole cuts. The

Table 6.1: Lower bounds on the chromatic number in a 0-1 clique formulation of problem instances consisting of one q-cycle with overlap of s.

q	s	Without cuts	Odd hole cuts only	x-cut only	z-cut only	x-cut & z-cut	Optimal value	No. of odd hole cuts
5	1	2.00	2.50	2.00	2.30	2.60	3	5
	2	4.00	4.00	4.00	4.50	5.00	5	320
	3	6.00	6.00	6.00	6.77	7.53	8	3645
	4	8.00	8.00	8.00	9.00	10.00	10	20,480
	5	10.00	10.00	10.00	11.26	12.52	13	78,125
7	1	2.00	2.33	2.00	2.21	2.43	3	7
	2	4.00	4.00	4.00	4.36	4.71	5	1792
	3	6.00	6.00	6.00	6.50	7.00	7	45,927
	4	8.00	8.00	8.00	8.68	9.36	10	458,752
9	1	2.00	2.25	2.00	2.17	2.33	3	9
	2	4.00	4.00	4.00	4.28	4.56	5	9216
	3	6.00	6.00	6.00	6.39	6.78	7	531,441

improvement is modest, but it is obtained at no additional cost. It is therefore advantageous to replace any set of standard cuts generated for an odd hole with these two cuts.

For $s > 1$, neither odd hole cuts nor 0-1 x-cuts alone have any effect on the bound when clique inequalities are present. However, a single 0-1 z-cut significantly improves the bound. Combining the z-cut with the x-cut raises the bound still further, substantially reducing the integrality gap, sometimes to zero. Two finite-domain cuts therefore provide a much tighter relaxation than a large set of standard clique inequalities and odd hole cuts.

We also investigated whether the finite-domain cuts are equally effective in the x-space, where they take their original form (6.15) and (6.28). We formulated a linear relaxation of the finite-domain model that minimizes $z+1$ subject to $z \geq x_i$ for all i, plus cuts. It is easy to show that the LP bound subject to the x-cut alone, or to the z-cut alone, is $\beta(q,s)/qs+1$.

The bound subject to both cuts is $2\beta(q,s)/qs+1$. These bounds appear in the left half of Table 6.2. The two cuts, when combined, yield the same bound as in the 0-1 model.

One might obtain a fairer comparison if clique inequalities are added to the finite-domain model, because they appear in the 0-1 model. In the finite-domain model, clique inequalities correspond to the individual alldiff constraints. We know from [40, 69] that for domain D_1, the following is facet-defining for alldiff(X_k):

$$\sum_{i\in V_k} x_i \geq \tfrac{1}{2}|V_k|(|V_k|-1)$$

In the test instances, $|V_k| = 2s$. We therefore added the following cuts:

$$\sum_{i\in V_k} x_i \geq s(2s-1), \quad k=1,\ldots,q$$

Using Theorem 18, we also added the cuts:

$$z \geq \frac{1}{qs}\sum_{i\in V_k} x_i + \frac{2s-1}{q}, \quad k=1,\ldots,q$$

The results appear in the right half of Table 6.2. The x-cut performs as before, but now the z-cut provides the same bound as in the 0-1 model. When combined, the x-cut and y-cut again deliver the same bound as in the 0-1 model.

Thus two odd cycle cuts yield the same bound in the very small finite-domain relaxation (even without clique inequalities) as in the much larger 0-1 relaxation. The finite-domain relaxation contains n variables x_i and $n+2$ constraints, while the 0-1 relaxation contains n^2+n variables y_{ij}, w_j and n^2+n+2 constraints (dropping odd hole cuts). It may therefore be advantageous to obtain bounds from a finite-domain model rather than a 0-1 model.

6.11.2 Webs

We generated the webs $W(q,r)$ shown in Table 6.3. We omitted clique cuts (when they exist) because they have no effect on the bound. The table shows that the 0-1 x-cut and z-cut, when used together, yield a tighter bound than the known 0-1 cuts discussed above. The improvement is modest, but the finite-domain cuts can replace known web cuts and tighten the bound at no additional cost.

Table 6.2: Lower bounds on the chromatic number in the finite-domain model of problem instances consisting of one q-cycle with overlap of s and color set $\{0, 1, \ldots, n-1\}$.

q	s	No cuts	x-cut only	z-cut only	x-cut & z-cut	Clique cuts	Plus x-cut	Plus z-cut	Plus x- & z-cut
5	1	1.00	1.80	1.80	2.60	1.50	1.80	2.30	2.60
	2	1.00	3.00	3.00	5.00	2.50	3.00	4.50	5.00
	3	1.00	4.27	4.27	7.53	3.50	4.27	6.77	7.53
	4	1.00	5.50	5.50	10.00	4.50	5.50	9.00	10.00
	5	1.00	6.76	6.76	12.52	5.50	6.76	11.26	12.52
7	1	1.00	1.71	1.71	2.43	1.50	1.71	2.21	2.43
	2	1.00	2.86	2.86	4.71	2.50	2.86	4.36	4.71
	3	1.00	4.00	4.00	7.00	3.50	4.00	6.50	7.00
	4	1.00	5.18	5.18	9.36	4.50	5.18	8.68	9.36
9	1	1.00	1.67	1.67	2.33	1.50	1.67	2.17	2.33
	2	1.00	2.78	2.78	4.56	2.50	2.78	4.28	4.56
	3	1.00	3.89	3.89	6.78	3.50	3.89	6.39	6.78

The LP bound given by the finite-domain model is $\gamma(q, r)/q + 1$ when the x-cut alone, or the z-cut alone, is present. The bound subject to both cuts is $2\gamma(q, r)/q + 1$. The latter bound is the same as shown in Table 6.3 for the combined cuts in the 0-1 model.

6.11.3 Benchmark Instances

We tested the strength of odd cycle cuts on benchmark instances of the vertex coloring problem taken from the DIMACS library. Table 6.4 displays the odd cycle bounds computed for instances with fewer than 100 variables. Larger instances almost always resulted in an out-of-memory error when the odd hole cuts were added.

We searched a given graph $G = (V, E)$ for cycles with $s = 1, 2, 3$ using the following greedy algorithm. Let the *co-neighborhood* of a set K of vertices be the intersection of

Table 6.3: Lower bounds on the chromatic number in a 0-1 formulation of problem instances consisting of a web $W(q, r)$. The bound given by the finite-domain formulation is the same as shown below when both cuts are used.

q	r	Without cuts	0-1 cuts only	x-cut only	z-cut only	x-cut & z-cut	Optimal value	No. of 0-1 cuts
5	2	2	2.50	2.00	2.30	2.60	3	5
7	2	2	3.50	2.29	2.79	3.57	4	7
	3	2	2.33	2.00	2.21	2.43	3	7
8	3	2	2.67	2.00	2.38	2.75	3	8
9	2	2	4.50	2.78	3.28	4.56	5	9
	4	2	2.25	2.00	2.17	2.33	3	9
10	3	2	3.33	2.20	2.70	3.40	4	10
11	2	2	5.50	3.27	3.77	5.55	6	11
	3	2	3.67	2.36	2.86	3.73	4	11
	4	2	2.75	2.00	2.41	2.82	3	11
	5	2	2.20	2.00	2.14	2.27	3	11

the neighborhoods of the individual vertices in K. For each $s \in \{1, 2, 3\}$ we proceed as follows. We first select the clique $\bar{S}_1$ of size s with the largest co-neighborhood (breaking ties arbitrarily). We then progressively build a path $\bar{S}_1, \bar{S}_2, \ldots$ by adding cliques $\bar{S}_\ell$. For each ℓ, if ℓ is odd, we examine cliques of size s that have vertices in $V \setminus (\bar{S}_1 \cup \cdots \cup \bar{S}_{\ell-1})$ and that continue the path, and select from these a clique $\bar{S}_\ell$ with the largest co-neighborhood. A clique K continues the path if all pairs $(i, j) \in \bar{S}_{\ell-1} \times K$ are edges in E. If ℓ is even, we check, for each clique K that continues the path, whether it allows completion of the cycle; that is, whether each pair $(i, j) \in \bar{S}_1 \times K$ is an edge in E. If so, we generate the cycle. (The vertices in $\bar{S}_1 \cup \cdots \cup \bar{S}_\ell$ may induce edges that are not in a cycle, but the odd cycle cuts are still valid.) We then let $\bar{S}_\ell$ be a clique that continues the path and has the largest

co-neighborhood in $V \setminus (\bar{S}_1 \cup \cdots \cup \bar{S}_{\ell-1})$. The process terminates when no clique continues the path.

Table 6.4 compares the bounds obtained from the 0-1 model after adding all odd hole cuts for the cycles found with the bounds obtained after adding all 0-1 x-cuts and z-cuts on these same cycles.

The results depend on the problem structure, but the finite-domain odd cycle cuts obtained tighter bounds in most instances, in some cases substantially tighter. As one might expect, the advantage is greater when there are cycles with $s > 1$. The time required to solve the LP relaxation was also consistently less for the finite-domain cuts (because there are only two of them per cycle), in some cases dramatically less.

6.12 Conclusion

We explored the idea of obtaining valid inequalities from a finite-domain formulation of a problem that is normally given a 0-1 formulation. We showed that in the case of graph coloring, this approach yields valid inequalities that provide tighter bounds on the chromatic number than known 0-1 cuts for the problem. In particular, we identified facet-defining inequalities for webs and odd holes that, when mapped into a 0-1 model, yield a tighter bound than standard 0-1 cuts. Furthermore, two finite-domain cuts for an odd cycle can yield substantially tighter bounds, in much less time, than hundreds or thousands of odd hole cuts. We also described a family of facet-defining cuts for intersecting systems, for which no 0-1 cuts seem to have been previously identified.

In addition, we discovered that web and odd cycle cuts provide the same tight bound in a relaxation of the finite-domain model as in a relaxation of the 0-1 model. If other families of finite-domain cuts follow this pattern, there could be advantage in obtaining bounds from a finite-domain relaxation that is much smaller than the 0-1 model. Given that some benchmark instances result in 0-1 models that are too large even to load into a linear solver [57], this could provide a viable alternative for solving large graph coloring and related problems.

The alternate polyhedral perspective afforded by the finite-domain formulation therefore

seems beneficial, at least in the case of graph coloring. The next step is to seek additional families of finite-domain cuts for graph coloring, perhaps corresponding to combs, anti-webs, and more general structures. The general strategy of obtaining valid inequalities and tight bounds from finite-domain formulations can be investigated for other problem classes.

Table 6.4: Lower bounds on the chromatic number in a 0-1 formulation of benchmark instances with n vertices and m edges, based on odd hole cuts and finite-domain odd cycle cuts. Number of cycles found (for $s = 1, 2, 3$) and LP solution time in seconds are also shown.

Instance	n	m	Bound Odd hole	Bound Odd cycle	Bound Opt	No. cycles found $s=1$	No. cycles found $s=2$	No. cycles found $s=3$	LP Time (sec) Odd hole	LP Time (sec) Odd cycle
1-Fulllns_3	30	100	2.00	2.00	3	18	0	0	0.4	0.4
1-Fulllns_4	93	593	2.00	2.00	4	61	0	0	208.0	0.4
1-Insertions_4	67	232	1.33	1.43	4	48	0	0	30.3	2.4
2-Fulllns_3	52	201	2.00	2.00	4	18	0	0	0.9	0.7
2-Insertions_3	37	72	1.25	1.33	3	8	0	0	2.9	0.2
3-Fulllns_3	80	346	2.00	2.00	5	25	0	0	25.8	0.2
3-Insertions_3	56	110	1.20	1.27	4	10	0	0	11.5	1.0
4-Insertions_3	79	156	1.17	1.23	3	12	0	0	12.1	6.0
david	87	406	2.00	8.00	10	103	48	10	11.0	0.8
huck	74	301	2.00	8.00	10	71	28	4	7.2	0.3
jean	80	254	2.00	8.00	10	26	0	8	10.2	1.8
mug88_1	88	146	2.00	2.00	3	2	0	0	7.8	2.7
mug88_25	88	146	2.00	2.00	3	4	0	0	5.3	1.7
myciel3	11	20	1.50	1.60	3	4	0	0	0.0	0.0
myciel4	23	71	1.50	1.60	4	18	0	0	0.6	0.0
myciel5	47	236	1.50	1.60	5	45	0	0	7.9	0.1
myciel6	95	755	1.50	1.60	6	153	0	0	1754.7	0.6
queen5_5	25	160	2.00	2.00	4	47	0	0	0.4	0.0
queen6_6	36	290	2.00	5.00	6	65	1	0	1.5	0.1
queen7_7	49	476	2.00	3.71	6	105	1	0	10.6	0.2
queen8_8	64	728	*	3.38	8	133	5	0	*	3.4
queen8_12	96	1368	2.00	8.00	11	229	31	20	439.6	1.7
queen9_9	81	1056	2.00	8.00	9	193	2	1	212.4	1.3

*LP solver ran out of memory.

Chapter 7

Conclusions

In conclusion, in this dissertation we explore novel solution methods and bounding techniques for discrete optimization problems.

The first method investigated is the use of approximate decision diagrams to represent the feasible set. We describe two forms of approximate decision diagrams: relaxed and restricted. Relaxed decision diagrams provide an over-approximation of the feasible set and are used to prove upper-bounds (assuming maximization) on the optimal value. Restricted decision diagrams provide an under-approximation of the feasible set and provide lower-bounds (assuming maximization) on the objective function. We then discuss how these structures can be used together in a branch-and-bound algorithm. Computational experiments show that the proposed algorithm is competitive with state-of-the-art integer programming software.

There are many opportunities for my current line of research on decision diagrams to develop in the future. Continuing in discrete optimization, one such direction is the use of a flow-polytope to transform the discrete relaxation provided by an approximate decision into a continuous relaxation. In this way, the relaxations we have been developing can be integrated with existing methodologies that utilize other continuous relaxations. This also allows, for example, decision diagrams to model non-linear objective function terms and constraints.

Beyond discrete optimization, decision diagrams have also shown to have an intimate

connection with dynamic programming and hence approximate decision diagrams may be useful for approximate dynamic programming. In addition, decision diagrams have recently been applied in the context of sequential pattern data mining and perhaps for large-scale problems, approximate decision diagrams may be an alternate mechanism to accomplish the same task. There is also the possibility of using decision diagrams for stochastic programming because of the sequential nature of the structure, viewing each layer as a stage in the stochastic process. I am excited about the numerous prospects of the application of decision diagrams and look forward to continuing this line of research.

The second method explored is the use of finite-domain models for finding strong cutting-planes for 0-1 problems. The idea is to analyze the problem in an alternate space, perhaps one that arises from a constraint programming formulation of the problem, and investigate the convex hull of feasible solution in this space. Then investigate a mapping that takes cuts in the finite-domain space and maps them to the original 0-1 space.

We apply this general technique to the graph coloring problem. We show that in the case of graph coloring, this approach yields valid inequalities that provide tighter bounds on the chromatic number than known 0-1 cuts for the problem. In particular, we identified facet-defining inequalities for webs and odd-holes that, when mapped into a 0-1 model, yield a tighter bound than standard 0-1 cuts. Furthermore, two finite-domain cuts for an odd-cycle can yield substantially tighter bounds, in much less time, than hundreds or thousands of odd-hole cuts. We also described a family of facet-defining cuts for intersecting systems, for which no 0-1 cuts seem to have been previously identified.

In addition, we discovered that web and odd-cycle cuts provide the same tight bound in a relaxation of the finite-domain model as in a relaxation of the 0-1 model. If other families of finite-domain cuts follow this pattern, there could be advantage in obtaining bounds from a finite-domain relaxation that is much smaller than the 0-1 model. Given that some benchmark instances result in 0-1 models that are too large even to load into a linear solver [57], this could provide a viable alternative for solving large graph coloring and related problems.

The alternate polyhedral perspective afforded by the finite-domain formulation therefore seems beneficial, at least in the case of graph coloring. The next step is to seek additional

families of finite-domain cuts for graph coloring, perhaps corresponding to combs, anti-webs, and more general structures. The general strategy of obtaining valid inequalities and tight bounds from finite-domain formulations can be investigated for other problem classes.

Bibliography

[1] E. Aarts and J. K. Lenstra. *Local Search in Combinatorial Optimization.* John Wiley & Sons, New York, 1997.

[2] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27:509–516, 1978.

[3] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A constraint store based on multivalued decision diagrams. In C. Bessière, editor, *Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *Lecture Notes in Computer Science*, pages 118–132. Springer, 2007.

[4] G. Appa, D. Magos, and I. Mourtos. Linear programming relaxations of multiple all-different predicates. In J. C. Régin and M. Rueher, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2004)*, volume 3011 of *Lecture Notes in Computer Science*, pages 364–369. Springer, 2004.

[5] G. Appa, D. Magos, and I. Mourtos. On the system of two all-different predicates. *Information Processing Letters*, 94:99–105, 2004.

[6] B. Becker, M. Behle, F. Eisenbrand, and R. Wimmer. BDDs in a branch and cut framework. In S. Nikoletseas, editor, *Experimental and Efficient Algorithms, Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA 05)*, volume 3503 of *Lecture Notes in Computer Science*, pages 452–463. Springer, 2005.

[7] M. Behle. *Binary Decision Diagrams and Integer Programming.* PhD thesis, Max Planck Institute for Computer Science, 2007.

[8] M. Behle and F. Eisenbrand. 0/1 vertex and facet enumeration with BDDs. In *Proceedings of the Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 158–165. SIAM, 2007.

[9] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. Variable ordering for the application of BDDs to the maximum independent set problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2012).* Springer, to appear.

[10] D. Bergman, A. A. Cire, W.-J. van Hoeve, and Tallys Yunes. Bdd-based heuristics for binary optimizatoin. submitted.

[11] D. Bergman and J. N. Hooker. Graph coloring facets from all-different systems,. In N. Jussien and T. Petit, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combintaorial Optimization Problems (CPAIOR 2012).* Springer, to appear.

[12] D. Bergman, W.-J. van Hoeve, and J. N. Hooker. Manipulating MDD relaxations for combinatorial optimization. In T. Achterberg and J.C. Beck, editors, *CPAIOR*, volume 6697 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2011.

[13] Timo Berthold. Primal heuristics for mixed integer programs. Master's thesis, Zuze Institute Berlin, 2006.

[14] Dimitris Bertsimas, Dan A. Iancu, and Dmitriy Katz. A new local search algorithm for binary optimization. *INFORMS Journal on Computing*, 2012.

[15] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45:993–1002, 1996.

[16] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

[17] V. Campos, E. Pi nana, and R. Martí. Adaptive memory programming for matrix bandwidth minimization. *Annals of Operations Research*, To appear.

[18] Alberto Caprara, Matteo Fischetti, and Paolo Toth. Algorithms for the set covering problem. *Annals of Operations Research*, 98:2000, 1998.

[19] Andre A. Cire and Willem-Jan van Hoeve. Mdd propagation for disjunctive scheduling. In *Proceedings of the Twenty-Second International Conference on Automated Planing and Scheduling (ICAPS).*

[20] P. Coll, J. Marenco, I. Méndez-Díaz, and P. Zabala. Facets of the graph coloring polytope. *Annals of Operations Research*, 116:79–90, 2002.

[21] Gianna M. Del Corso and Giovanni Manzini. Finding exact solutions to the bandwidth minimization problem. *Computing*, 62(3):189–203, 1999.

[22] R. Ebendt, W. Gunther, and R. Drechsler. An improved branch and bound algorithm for exact BDD minimization. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(12):1657–1663, 2003.

[23] Jonathan Eckstein and Mikhail Nediak. Pivot, cut, and dive: a heuristic for 0-1 mixed integer programming. *J. Heuristics*, 13(5):471–503, 2007.

[24] Uriel Feige. Approximating the bandwidth via volume respecting embeddings. *J. Comput. Syst. Sci*, 60(3):510–539, 2000.

[25] Matteo Fischetti, Fred Glover, and Andrea Lodi. The feasibility pump. *Math. Program*, 104(1):91–104, 2005.

[26] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pac. J. Math.*, 15:835–855, 1965.

[27] L. Genç-Kaya and J. N. Hooker. The circuit polytope. Manuscript, Carnegie Mellon University, 2010.

[28] Fred Glover and Manuel Laguna. General purpose heuristics for integer programming-part i. *Journal of Heuristics*, 2:343–358, 1997.

[29] Fred Glover and Manuel Laguna. General purpose heuristics for integer programming-part i. *Journal of Heuristics*, 2:343–358, 1997.

[30] Andrea Grosso, Marco Locatelli, and Wayne Pullan. Simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics*, 14:587–612, 2008.

[31] M. Grötschel and M. W. Padberg. On the symmetric traveling salesman problem: I: Inequalities. *Mathematical Programming*, 16:265–280, 1979.

[32] M. Grötschel and M. W. Padberg. On the symmetric traveling salesman problem: I: Lifting theorems and facets. *Mathematical Programming*, 16:281–302, 1979.

[33] Martin Grötschel, Lászlo Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2. Springer, 1993.

[34] Gurari and Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the mincut linear arrangement problem. *ALGORITHMS: Journal of Algorithms*, 5, 1984.

[35] Gurari and Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the mincut linear arrangement problem. *ALGORITHMS: Journal of Algorithms*, 5, 1984.

[36] T. Hadzic and J. N. Hooker. Postoptimality analysis for integer programming using binary decision diagrams, presented at GICOLAG workshop (Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geometry), Vienna. Technical report, Carnegie Mellon University, 2006.

[37] T. Hadzic and J. N. Hooker. Cost-bounded binary decision diagrams for 0-1 programming. In E. Loute and L. Wolsey, editors, *Proceedings of the International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2007)*, volume 4510 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2007.

[38] T. Hadzic, J. N. Hooker, B. O'Sullivan, and P. Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In P. J. Stuckey, editor, *Principles and Practice of Constraint Programming (CP 2008)*, volume 5202 of *Lecture Notes in Computer Science*, pages 448–462. Springer, 2008.

[39] S. Hoda, W.-J. van Hoeve, and J. N. Hooker. A systematic approach to MDD-based constraint programming. In *Proceedings of the 16th International Conference on Principles and Practices of Constraint Programming*, volume 6308 of *Lecture Notes in Computer Science*, pages 266–280. Springer, 2010.

[40] J. N. Hooker. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction.* Wiley, New York, 2000.

[41] J. N. Hooker. *Integrated Methods for Optimization.* Springer, 2007.

[42] J. N. Hooker. *Integrated Methods for Optimization.* Springer, 2007.

[43] J. N. Hooker. Decision diagrams and dynamic programming. In C. Gomes and M. Sellmann, editors, *CPAIOR 2013 Proceedings*. Springer, to appear.

[44] A. J. Hu. Techniques for efficient formal verification using binary decision diagrams. Thesis CS-TR-95-1561, Stanford University, Department of Computer Science, December 1995.

[45] C. Jordan. Sur les assemblages de lignes. *J. Reine Angew Math*, 70:185–190, 1869.

[46] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Multi-valued decision diagrams: Theory and applications. *International Journal on Multiple-Valued Logic*, 4:9–62, 1998.

[47] S. Kruk and S. Toma. On the system of the multiple all different predicates. *Congressus Numerantium*, 197:47–64, 2009.

[48] S. Kruk and S. Toma. On the facets of the multiple alldifferent constraint. *Congressus Numerantium*, 204:5–32, 2010.

[49] S. Kruk, S. Toma, and M. Wallace. Some facets of multiple alldifferent predicate. In P. Belotti, editor, *Workshop on Bound Reduction Techniques for Constraint Programming and Mixed-Integer Nonlinear Programming, at CPAIOR*, 2009.

[50] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal*, 38:985–999, 1959.

[51] E. Loekito, J. Bailey, and J. Pei. A binary decision diagram based approach for mining frequent subsequences. *Knowl. Inf. Syst*, 24(2):235–268, 2010.

[52] D. Magos and I. Mourtos. On the facial structure of the alldifferent system. *SIAM Journal on Discrete Mathematics*, pages 130–158, 2011.

[53] D. Magos, I. Mourtos, and G. Appa. A polyhedral approach to the *alldifferent* system. *Mathematical Programming*, to appear.

[54] Rafael Martí, Vicente Campos, and Estefanía Piñana. A branch and bound algorithm for the matrix bandwidth minimization. *European Journal of Operational Research*, 186(2):513–528, 2008.

[55] Rafael Martí, Manuel Laguna, Fred Glover, and Vicente Campos. Reducing the bandwidth of a sparse matrix with tabu search. *European Journal of Operational Research*, 135(2):450–459, 2001.

[56] I. Méndez-Díaz and P. Zabala. A polyhedral approach for graph coloring. *Electronic Notes in Discrete Mathematics*, 7:178–181, 2001.

[57] I. Méndez-Díaz and P. Zabala. A cutting plane algorithm for graph coloring. *Discrete Applied Mathematics*, 156:159–179, 2008.

[58] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *30th Conference on Design Automation*, pages 272–277. IEEE, 1993.

[59] G. Palubeckis. On the graph coloring polytope. *Information Technology and Control*, 37:7–11, 2008.

[60] Estefanía Piñana, Isaac Plana, Vicente Campos, and Rafael Martí. GRASP and path relinking for the matrix bandwidth minimization. *European Journal of Operational Research*, 153(1):200–210, 2004.

[61] Wayne Pullan, Franco Mascia, and Mauro Brunato. Cooperating local search for the maximum clique problem. *Journal of Heuristics*, 17:181–199, 2011.

[62] S. Rebennack, M. Oswald, D. Theis, H. Seitz, G. Reinelt, and P. Pardalos. A branch and cut solver for the maximum stable set problem. *Journal of Combinatorial Optimization*, 21:434–457, 2011.

[63] S. Rebennack, G. Reinelt, and P. Pardalos. A tutorial on branch and cut algorithms for the maximum stable set problem. *International Transactions in Operational Research*, 19:161–199, 2012.

[64] F. Rossi and S. Smriglio. A branch-and-cut algorithm for the maximum cardinality stable set problem. *Operations Research Letters*, 28:63–74, 2001.

[65] Thomas Rothvoß. Some 0/1 polytopes need exponential size extended formulations. *CoRR*, abs/1105.0036, 2011.

[66] J. Saxe. Dynamic programming algorithms for recognizing small-bandwidth graphs in polynomial time. *SIAM J. Algebraic Discrete Meth.*, 1:363–369, 1980.

[67] E. Tomita and T. Kameda. An efficient branch-and-bound algorithm for finding a maximum clique, with computational experiments. *Journal of Global Optimization*, 111:95–111, 2007.

[68] I. Wegener. *Branching programs and binary decision diagrams: theory and applications.* SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics, 2000.

[69] H. P. Williams and H. Yan. Representations of the all_different predicate of constraint satisfaction in integer programming. *INFORMS Journal on Computing*, 13:96–103, 2001.

[70] H. Yan and J. N. Hooker. Tight representations of logical constraints as cardinality rules. *Mathematical Programming*, 85:363–377, 1995.

[71] T. H. Yunes. On the sum constraint: Relaxation and applications. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP 2002)*, volume 2470 of *Lecture Notes in Computer Science*, pages 80–92. Springer, 2002.