Carnegie Mellon University
MELLON COLLEGE OF SCIENCE

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Doctor of Philosophy
in Algorithms, Combinatorics and Optimization

TITLE Integrating Relaxations for Combinatorial Optimization

PRESENTED BY Marla Slusky

ACCEPTED BY THE DEPARTMENT OF Mathematical Sciences

Willem-Jan van Hoeve MAJOR PROFESSOR	May 2015 DATE
Thomas Bohman DEPARTMENT HEAD	May 2015 DATE

APPROVED BY THE COLLEGE COUNCIL

Frederick J. Gilman DEAN	May 2015 DATE

CARNEGIE MELLON UNIVERSITY
DEPARTMENT OF MATHEMATICAL SCIENCES

DOCTORAL DISSERTATION

INTEGRATING RELAXATIONS FOR COMBINATORIAL OPTIMIZATION

MARLA REBECCA SLUSKY

May 4th, 2015

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Algorithms, Combinatorics, and Optimization.

DISSERTATION COMMITTEE:
WILLEM-JAN VAN HOEVE (CHAIR)
ALAN FRIEZE
GÉRARD CORNUÉJOLS
LOUIS-MARTIN ROUSSEAU

Acknowledgements

I have many people to thank for helping me finish this dissertation.

Firstly, thank you to my advisor, Willem-Jan van Hoeve, for all of his advice, guidance, unwavering support, and for keeping me on track to graduate.

Thank you to Andre Cire for lending me his codebase, and for answering all my questions about it.

Thank you to all my friends who helped me through the program: Emily Allen, Susie Backscheider, Deepak Bal, Patrick Bennett, Jacob Davis, Lisa Espig, Will Gunther, Jenny Iglesias, Brian Kell, Chris Lambie-Hanson, Misha Lavrov, Paul McKenney, Clive Newstead, Brendan Sullivan, and Spencer and Erin Unger.

Thanks especially to Brian for helping me talk through ideas and algorithms, for his advice on my writing, and for taking the time to give me valuable feedback on my presentation and thesis; to Will for answering all my Perl and LaTeXquestions; and to Chris for helping me cultivate a hobby outside graduate school.

Thank you to Bob and Laura Zimmermann and to Samantha Gross. Your perspective from outside the department helped me see the big picture and stay focused on finishing.

Thank you to my family, Susan Slusky, Ron Slusky, Joanna and David Slusky, and Ben Slusky and Reha Sterbin for all the support and encouragement you've given me through graduate school and before.

Lastly, thank you to Jason Rute for encouraging me and helping me stay motivated all the way through graduate school, but especially for the final push at the end.

To Havi and Jake.

Abstract

In this thesis we explore two methods of computing lower bounds. We first discuss the Lagrangian Relaxation as it applies to the Golomb ruler problem, and then we explore adding multi-valued decision diagrams to an additive bounding scheme.

The Golomb Ruler Problem asks to position n integer marks on a ruler such that all pairwise distances between the marks are distinct and the ruler has minimum total length. It is a notoriously challenging combinatorial problem, and provably optimal rulers are only known for n up to 27. Lower bounds can be obtained using linear programming (LP) formulations, but these are computationally expensive for large n. In Chapter 2 of this thesis, we propose a new method for finding lower bounds based on a Lagrangian relaxation. We apply a subgradient optimization scheme to find good bounds quickly, and we show experimentally that our method can find bounds that are very close to the optimal LP bound in a fraction of the time that is needed to compute the LP bound. We furthermore embed our Lagrangian bounds into a constraint programming search procedure, and show that these can help reduce the constraint programming search tree considerably.

Additive bounding is a method of taking several algorithms for computing lower bounds, each of which typically exploits a different substructure of the problem, and combining them to produce a single lower bound which is larger than the lower bound that any of the individual algorithms can produce alone. Approximate multi-valued decision diagrams (MDDs) have recently been used to compute upper and lower bounds on several optimization problems. In Chapter 3 of this thesis, we show how we can integrate MDDs into an addivite bounding scheme.

Contents

1 Introduction 1

2 Lagrangian Bounds for Golomb Rulers 3
- 2.1 Introduction . . . 3
- 2.2 Exact Models for the Golomb Ruler Problem . . . 5
- 2.3 Lagrangian Relaxation . . . 6
  - 2.3.1 Formulation . . . 6
  - 2.3.2 A Combinatorial Algorithm for Solving the Relaxation . . 8
  - 2.3.3 Subgradient Optimization Method . . . 9
- 2.4 Relationship with Other Formulations . . . 9
  - 2.4.1 Permutation Formulation . . . 10
  - 2.4.2 Equation Sums Bound . . . 12
- 2.5 Results for Approximating the LP Bound . . . 14
  - 2.5.1 Subset Formulation . . . 14
  - 2.5.2 Implementation and Results . . . 15
- 2.6 Lagrangian Bounds for Ruler Segments . . . 18
- 2.7 Embedding in Exact CP Search . . . 19
  - 2.7.1 Constraint Programming Representation and Search . . . 20
  - 2.7.2 Adding the Lagrangian Relaxation . . . 22
  - 2.7.3 Adding Bounds on Ruler Segments . . . 23
- 2.8 Conclusion . . . 26

3 Residual Costs from Relaxed MDDs 29
- 3.1 Introduction . . . 29
  - 3.1.1 Multi-valued Decision Diagrams . . . 30
  - 3.1.2 Using MDDs to Solve an Optimization Problem . . . 32
  - 3.1.3 Additive Bounding . . . 34
- 3.2 Extracting Residual Costs from an MDD . . . 35
  - 3.2.1 Sequential Ordering Problem . . . 37
  - 3.2.2 Other Methods of Obtaining Potentials . . . 39
- 3.3 Experiments . . . 44
  - 3.3.1 Other Additive Bounding Algorithms Used . . . 44
  - 3.3.2 Data sets . . . 44
  - 3.3.3 Results . . . 46

3.4 Conclusion 49

4 Conclusion 51

A Search Procedure for Ruler Segment Bound 53

Chapter 1

Introduction

Modern industry relies heavily on optimization. It can be used to find optimal shipping routes, facility placement, or production levels. It can be applied to any problem in which one wants to minimize or maximize an objective function, like cost or profit, subject to some constraints.

Often the variables in such problems are constrained to take on integer values. For example, one can only make use of an integer number of factories. Often such problems are NP-hard.

When a problem is NP-hard, it is often easier to find approximate solutions using upper and lower bounds. For examples of computing upper and lower bounds, consider the traveling salesperson problem. In this problem, we are given a network of N cities connected by roads. Each road has a cost associated with traversing it. The traveling salesperson problem asks to find a route that takes the salesperson from home, to each of the other cities, and back home. For $N = 30$, the number of possible routes is

$$4,420,880,996,869,850,977,271,808,000,000$$

or 4.4×10^{30}, which is too many for a computer to exhaustively check.

We can use a heuristic, (e.g., from each city, visit the cheapest city that has not yet been visited) to get a feasible solution. The cost of this, or any, feasible solution gives an upper bound on the optimal objective value.

We can use a relaxation of the problem to get a lower bound. A relaxation is a problem with fewer constraints, which is often easier to solve. Since all solutions to the original problem are feasible in the relaxation, and we are taking a minimum over all feasible solutions, the optimal solution to the relaxation provides a lower bound on the original problem.

In the example of the traveling salesperson, one relaxation is the problem in which we are allowed any number of salespeople, each starting in a different city, visiting at least one other city, and then returning home. This problem can be solved in polynomial time, specifically, $O(N^3)$ [1].

One common method for producing good solutions is called branch and bound which finds a sequence of solutions converging to the optimal solution.

The main idea in branch and bound is to branch on cases, for example the case in which the salesperson visits Pittsburgh immediately after New York, and the case in which they do not. From each of these cases, we branch further creating a tree of cases. At each node, we can use a relaxation to compute a lower bound on the objective function and if any case leads to a lower bound greater than some known upper bound, we can prune that branch of the tree, since no solutions on that branch can be optimal. For this reason it is important to have relaxations that can be used to efficiently compute lower bounds.

In this dissertation, we explore methods of computing lower bounds. In chapter 2 we explore the Lagrangian relaxation, in which we allow a constraint to be violated, but add a penalty to the objective function when it is. We show that applying the Lagrangian relaxation to the Golomb ruler problem can considerably speed up the computation of lower bounds — enough that it is practical to embed in a search procedure.

In chapter 3 we explore additive bounding, a technique for chaining together several algorithms which produce lower bounds. This is particularly useful when the different relaxations exploit different structures of the problem. We introduce the idea of using multi-valued decision diagrams for a discrete relaxation in an additive bounding scheme.

Decision diagrams were originally used for verification of switching circuits [2, 10, 33], and have more recently been applied to optimization (e.g., [3]). We show the effectiveness of our methods by combining the MDD relaxation with other relaxations from the literature. We apply this procedure to the sequential ordering problem, which is a variation of the traveling salesperson problem.

Chapter 2

Lagrangian Bounds for Golomb Rulers

2.1 Introduction

For some positive integer n, let $x_1, \ldots, x_n$ represent the integer positions of n marks on a ruler. We can assume that $x_i < x_j$ for all $1 \leq i < j \leq n$ and that $x_1 = 0$. A *Golomb ruler* has pairwise distinct distances between the marks, i.e., $x_j - x_i$ for all $1 \leq i < j \leq n$ are distinct. Given n, the Golomb ruler problem asks to find a Golomb ruler with minimum length x_n. For example, an optimal ruler for $n = 5$ is presented in Figure 2.1.

Practical applications of the Golomb ruler problem include radio communications, X-ray crystallography, coding theory, and radio astronomy [9, 17, 38, 42]. The problem continues to be very difficult to solve in practice, although it is still unknown whether it is NP-hard. Optimal Golomb rulers are only known up to $n = 27$. The optimality of rulers of 24, 25, 26 and 27 marks was proven by a massively parallel search coordinated by `distributed.net/ogr`. The search for the provably optimal 27 mark ruler lasted almost 5 years.[1]

The Golomb ruler problem is also a popular benchmark for discrete optimization methods, such as constraint programming methods [18, 51], algebraic methods (affine and projective plane constructions, [15, 49]), evolutionary algorithms [52], and hybrid methods combining constraint programming and local search [14, 40].

A crucial component of exact solution methods is producing lower bounds, which appears to be more challenging than providing upper bounds that correspond to feasible rulers. Lower bounds can help to dramatically prune an exhaustive search, but only if they can be found quickly enough. Lower bounds based on linear programming (LP) formulations were proposed by Lorentzen and Nilsen [35], Hansen et al. [22] and Shearer [48]. These three formulations were proved equivalent by Meyer and Jaumard [37]. A shortcoming of the LP bound

---

[1] See `http://stats.distributed.net/projects.php?project_id=27`

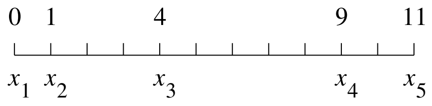


Figure 2.1: An optimal Golomb ruler with $n = 5$ marks.

is that direct LP formulations grow too large to fit in memory, and iterative methods are computationally expensive. Therefore, the practical applicability of the LP bound inside exact methods is limited. Another bound was discussed by Galinier et al. [18] and applied within a constraint programming approach for solving the problem. This bound is weaker than the LP bound, but it can be computed more quickly.

In this chapter, we propose a new method for producing lower bounds, based on a Lagrangian relaxation of the problem. We show that our relaxation generalizes the bounds proposed in Galinier et al. [18], and can produce a bound that is equivalent to the LP bound. We also show how we can extend our approach to provide bounds on *segments* of the ruler. Furthermore, we present an algorithm that solves the relaxation in $O(n^2 \log n)$ time for fixed Lagrangian multipliers. This allows us to efficiently approximate the LP bound using a subgradient optimization method. We experimentally demonstrate that in practice our method can produce bounds almost as strong as the LP bound, much faster than existing methods. Because this algorithm is much faster to compute than existing methods for the LP bound, we can apply our bounds within a constraint programming search procedure. We will demonstrate that it can decrease the constraint programming search tree size up to 91% in certain cases, which translates into a solving time reduction of up to 78%. We note that the performance heavily depends on the quality of the LP bound that we approximate. If the LP bound is too weak, our method is naturally also less effective.

The rest of the chapter is organized as follows. In Section 2.2 we present formal models of the Golomb ruler problem. In Section 2.3 we present the Lagrangian formulation, our efficient algorithm to solve the relaxation, and the subgradient optimization method. Section 2.4 discusses exact methods to solve the Lagrangian relaxation and relates our formulation to the formulations in [22], [37], and [18]. Section 2.5 contains the computational results comparing our new formulation to the formulations in [22] and [48], the current state of the art. We then introduce Lagrangian bounds for ruler segments in Section 2.6. Section 2.7 describes how we embed the Lagrangian bounds on the ruler and on ruler segments in a constraint programming search procedure, and discusses the benefits of introducing the bounds. Lastly, we provide a conclusion in Section 2.8.

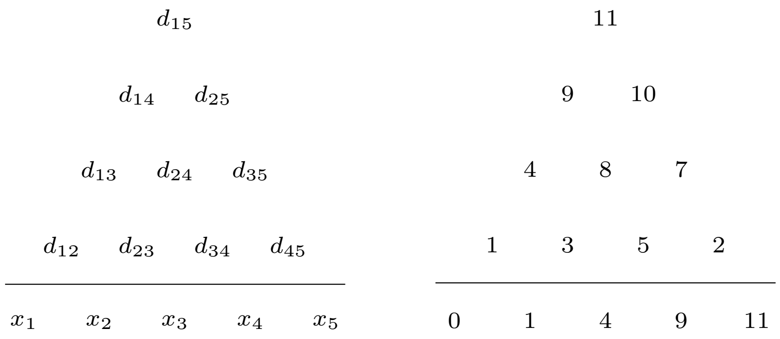

d_{15}
d_{14} d_{25}
d_{13} d_{24} d_{35}
d_{12} d_{23} d_{34} d_{45}
x_1 x_2 x_3 x_4 x_5
11
9 10
4 8 7
1 3 5 2
0 1 4 9 11

Figure 2.2: Distances associated with the Golomb ruler from Figure 2.1.

2.2 Exact Models for the Golomb Ruler Problem

We first present a formal model of the Golomb ruler problem. In the following, we will assume that the marks take their position from a range $\{0, 1, \ldots, L\}$ for some appropriate upper bound L.

Rather than taking the marks $x_1, \ldots, x_n$ to be our variables, we will take the $\binom{n}{2}$-many *segment lengths* $d_{ij} := x_j - x_i$ to be our variables (Figure 2.2 provides an illustration for the ruler in Figure 2.1). Then the Golomb ruler problem can be expressed as the following constraint programming (CP) model:[2]

$$\begin{aligned} \min \quad & \sum_{k=1}^{n-1} d_{k,k+1} \\ \text{s.t.} \quad & \texttt{AllDifferent}(\{d_{ij}\}) \\ & d_{ij} = \sum_{k=i}^{j-1} d_{k,k+1} \qquad \text{for all } 2 \leq i+1 < j \leq n. \end{aligned} \tag{2.1}$$

We can alternatively express this CP model as an integer programming (IP) model, by representing the `alldifferent` constraint explicitly as a bipartite matching problem. That is, we introduce a vertex set corresponding to the pairs of marks $\{(i, j) \mid 1 \leq i < j \leq n\}$, a vertex set corresponding to the possible lengths $\{1, 2, \ldots, L\}$, and we define the complete bipartite graph between these two vertex sets. Clearly, a maximum matching in this graph corresponds to a solution to `alldifferent` [41].

For our IP model, we introduce binary 'edge' variables such that $e_{ijv} = 1$ when the pair (i, j) induces a distance $v \in \{1, \ldots, L\}$ and $e_{ijv} = 0$ otherwise. The model thus becomes:

---

[2] The symbolic (or global) constraint `alldifferent`(X) on a set of variables X with finite domains specifies that the variables in X take distinct values [26].

$$
\begin{array}{lll}
\min & \displaystyle\sum_{k=1}^{n-1} d_{k,k+1} & \\
\text{s.t.} & \displaystyle\sum_{v=1}^{L} e_{ijv} = 1 & \text{for all } 1 \le i < j \le n, \\
& \displaystyle\sum_{i<j} e_{ijv} \le 1 & \text{for all } v = 1, \ldots, L, \\
& \displaystyle\sum_{v=1}^{L} v \cdot e_{ijv} = d_{ij} & \text{for all } 1 \le i < j \le n, \\
& \displaystyle\sum_{k=i}^{j-1} d_{k,k+1} = d_{ij} & \text{for all } 2 \le i+1 < j \le n, \\
& e_{ijv} \in \{0,1\} & \text{for all } 1 \le i < j \le n,\ v = 1, \ldots, L.
\end{array}
\tag{2.2}
$$

In this model, the first two constraints represent the bipartite matching. The third constraint establishes the relationship between the variables e_{ijv} and d_{ij}. The fourth is the requirement that each larger segment is made up of the smaller segments it contains. We note that model (2.2) corresponds to the formulation suggested by Lorentzen and Nilsen [35]. We will refer to it as the *matching* formulation and to the objective value of its linear programming relaxation (in which $0 \le e_{ijv} \le 1$) as z_{matching}. We will derive our Lagrangian relaxation from this model.

2.3 Lagrangian Relaxation

In this section, we first present the Lagrangian formulation, which provides a relaxation for any fixed set of Lagrangian multipliers. We then show that each such relaxation can be solved efficiently. In order to find the best relaxation (corresponding to the LP lower bound), we lastly present a subgradient optimization method that approximates the optimal Lagrangian multipliers.

2.3.1 Formulation

We create a Lagrangian relaxation from model (2.2) as follows. For every pair of non-consecutive marks, that is, for all i, j such that $2 \le i+1 < j \le n$, we choose a coefficient $\lambda_{ij} \in \mathbb{R}$ and consider the LP resulting from moving the last constraint of the matching formulation to the objective function:

$$\begin{aligned}
\min \quad & \sum_{k=1}^{n-1} d_{k,k+1} + \sum_{i+1<j} \lambda_{ij} \left(d_{ij} - \sum_{k=i}^{j-1} d_{k,k+1} \right) \\
\text{s.t.} \quad & \sum_{v=1}^{L} e_{ijv} = 1 && \text{for all } 1 \le i < j \le n, \\
& \sum_{i<j} e_{ijv} \le 1 && \text{for all } v = 1, \ldots, L, \\
& d_{ij} = \sum_{v=1}^{L} v \cdot e_{ijv} && \text{for all } 1 \le i < j \le n, \\
& e_{ijv} \ge 0 && \text{for all } 1 \le i < j \le n,\ v = 1, \ldots, L.
\end{aligned} \tag{2.3}$$

In this formulation we do not enforce $\sum_{k=i}^{j-1} d_{k,k+1} = d_{ij}$, but we do incur a penalty, weighted by λ_{ij}, if we do not satisfy that constraint. Note that the optimal solution for the matching formulation is still feasible in this relaxation, and gives the same objective value. Therefore, the optimal value here is *at most* z_{matching}, and therefore the optimal value is a lower bound for (2.2).

We can simplify our model further by rearranging the objective function to become

$$\sum_{i+1<j} \lambda_{ij} d_{ij} + \sum_{k=1}^{n-1} d_{k,k+1} \left(1 - \sum_{\substack{i \le k < j \\ i+1 \ne j}} \lambda_{ij} \right) . \tag{2.4}$$

Also, recall that we did not choose $\lambda_{k,k+1}$ for any k earlier, so let us take

$$\lambda_{k,k+1} := 1 - \sum_{\substack{i \le k < j \\ i+1 \ne j}} \lambda_{ij} \ . \tag{2.5}$$

Then for any fixed (λ_{ij}) satisfying equation (2.5), we have the simpler LP:

$$\begin{aligned}
\min \quad & \sum_{i<j} \lambda_{ij} d_{ij} \\
\text{s.t.} \quad & \sum_{v=1}^{L} e_{ijv} = 1 && \text{for all } 1 \le i < j \le n, \\
& \sum_{i<j} e_{ijv} \le 1 && \text{for all } v = 1, \ldots, L, \\
& d_{ij} = \sum_{v=1}^{L} v \cdot e_{ijv} && \text{for all } 1 \le i < j \le n, \\
& e_{ijv} \ge 0 && \text{for all } 1 \le i < j \le n,\ v = 1, \ldots, L.
\end{aligned} \tag{2.6}$$

This is the LP we will refer to as the Lagrangian relaxation, and we will refer to its objective value as z_{LR}. Note that the d_{ij} variables are simply an intermediate calculation. If we replace the d_{ij} in the objective function with $\sum_{v=1}^{L} v \cdot e_{ijv}$, then we can eliminate the third constraint, and so this LP represents a matching problem. Together with the linear objective function, this ensures that model (2.6) has an integer optimal solution, if a solution exists [43].

Proposition 1. *For any fixed* (λ_{ij}) *we have* $z_{\text{LR}} \leq z_{\text{matching}}$*, and there exists* (λ_{ij}) *for which* $z_{\text{LR}} = z_{\text{matching}}$*.*

Proof. The proposition follows from choosing (λ_{ij}) to be the dual variables of the last equation in (2.2). (See, e.g., [39]). □

Proposition 2. *If* (λ_{ij}) *gives* $z_{\text{LR}} = z_{\text{matching}}$*, then* $\lambda_{ij} \geq 0$ *for all* i, j*.*

Proof. We refer the reader to [37] for the proof of this. □

2.3.2 A Combinatorial Algorithm for Solving the Relaxation

Since we will repeatedly solve the Lagrangian relaxation to obtain better bounds in a subgradient optimization scheme, it is important that model (2.6) be solved efficiently. As it reflects a bipartite matching problem, we can for example apply the Hungarian algorithm to find a solution in polynomial time [12, 32]. However, because of the quadratic number of nodes in the bipartite graph the associated time complexity would be $O(n^6)$ for a ruler with n marks, which is too high for our purposes. Instead, we present next a combinatorial algorithm that solves model (2.6) more efficiently.

Proposition 3. *For any fixed* (λ_{ij})*, the Lagrangian relaxation can be solved in* $O(n^2 \log n)$ *time for a Golomb ruler with* n *marks.*

Proof. The Lagrangian relaxation represents a bipartite matching problem in which we are matching each λ_{ij} with a number in $\{1, \ldots, L\}$, and we are trying to minimize the sum of the products of the pairs. It is clear that if $\lambda_{ij} \geq 0$ for all $i < j$, then to minimize the objective value we must match the largest λ_{ij} with 1, the next largest λ_{ij} with 2, etc. Thus our method for solving the Lagrangian relaxation will be as follows.

Algorithm 1 Solve the Lagrangian Relaxation

```
1: Sort $(\lambda_{ij})$ into decreasing order.
2: Let $d_{ij}$ be the location of $\lambda_{ij}$ in the sorted list.
```

Since our algorithm for solving the Lagrangian relaxation reduces to sorting $\binom{n}{2}$ elements, we can solve it in $O(n^2 \log n)$ time. □

We remark that Hansen et al. [22] use a similar sorting procedure for solving the subproblems in their scheme for computing the ‘quasi-positive lower bound’. This equivalence will be further detailed in Section 2.4.1.

2.3.3 Subgradient Optimization Method

In order to find (near-)optimal values for (λ_{ij}), we apply an iterative subgradient optimization method similar in spirit as those developed for the 1-tree relaxation in [23, 24]. To approximate good values for (λ_{ij}), recall that λ_{ij} is a penalty for not satisfying the constraint $\sum_{k=i}^{j-1} d_{k,k+1} = d_{ij}$. Therefore, if we solve the Lagrangian relaxation and do not satisfy the constraint for pair (i, j), we should increase the penalty λ_{ij}. This process is repeated until some stopping criterion is met (either a time, iteration, or relative improvement limit).

Algorithm 2 Subgradient optimization scheme for finding (near-)optimal Lagrangian multipliers.

```
 1: Choose initial *stepsize*
 2: Choose initial values for $\lambda_{ij}$ with $i + 1 < j$ (for example, all 0)
 3: **while** Some stopping criterion is not met **do**
 4:     Set $\lambda_{k,k+1} := 1 - \sum_{\substack{i \le k < j \\ i+1 \ne j}} \lambda_{ij}$ for all $k \in \{1, \ldots, n-1\}$
 5:     Solve the Lagrangian relaxation
 6:     **for all** $i + 1 < j$ **do**
 7:         $\lambda_{ij} := \lambda_{ij} + \left( d_{ij} - \sum_{k=i}^{j-1} d_{k,k+1} \right) \frac{stepsize}{n^2}$
 8:     **end for**
 9:     Adjust *stepsize* if necessary
10: **end while**
```

The algorithm is presented in Algorithm 2. The performance of this algorithm highly depends on the choice and adjustment of the stepsize parameter. In our implementation, we start with a stepsize of 1 (in line 1). When an iteration results in negative values for some λ_{ij}, we divide the stepsize in half to refine the search. Otherwise, after each 5 iterations of decreasing values for z_{LR}, we multiply the stepsize by 0.999 (line 9).

We remark that the algorithm does not have a natural stopping condition based on optimality of the solution, in contrast with standard subgradient optimization algorithms [34]. In fact, even if we use the optimal (λ_{ij}) as initial data, one iteration will return different values. Nevertheless, this algorithm produces very good approximations of z_{matching} very quickly, as we will see in Section 2.5.

2.4 Relationship with Other Formulations

In this section we investigate the relationship of our Lagrangian relaxation with other, existing, formulations for obtaining lower bounds. Throughout this section we will use λ to mean $(\lambda_{ij}) \in \mathbb{R}^{\binom{n}{2}}$; $S_{\binom{n}{2}}$ to be the set of all permutations of the numbers $\{1, 2, \ldots, \binom{n}{2}\}$ indexed by pairs (i, j) with $i < j$; $\sigma = (\sigma_{ij}) \in S_{\binom{n}{2}}$; and $\lambda \cdot \sigma = \sum_{1 \le i < j \le n} \lambda_{ij} \sigma_{ij}$.

2.4.1 Permutation Formulation

One consequence of Proposition 3 is that out of the elements in $\{1, 2, \ldots, L\}$ we will use only the smallest $\binom{n}{2}$ elements so as to minimize our objective function. In other words, for fixed λ, the sorting procedure can also be viewed as finding a permutation σ of the elements $\{1, 2, \ldots, \binom{n}{2}\}$ such that $\lambda \cdot \sigma$ is minimized:

$$\min_{\sigma} \lambda \cdot \sigma$$

Our overall goal, however, is to find the best possible bound, that is

$$\max_{\lambda} \min_{\sigma} \lambda \cdot \sigma. \tag{2.7}$$

We can formulate problem (2.7) as a linear program by introducing λ_{ij} as a variable for each pair $i < j$. If we let z represent the objective function to be maximized, we can introduce a constraint for each possible permutation σ to bound z:

$$z \leq \sum_{i<j} \lambda_{ij} \cdot \sigma_{ij} \quad \text{for all } \sigma \in S_{\binom{n}{2}}.$$

From the previous section there are no restrictions put on λ, other than condition (2.5) which can be rewritten as

$$\sum_{i \leq k < j} \lambda_{ij} = 1.$$

The overall linear program thus becomes:

$$\begin{array}{lll} \max & z & \\ \text{s.t.} & \displaystyle\sum_{i \leq k < j} \lambda_{ij} = 1 & \text{for all } k = 1, \ldots, n-1, \\ & z \leq \displaystyle\sum_{i<j} \lambda_{ij} \cdot \sigma_{ij} & \text{for all } \sigma \in S_{\binom{n}{2}}. \end{array} \tag{2.8}$$

We will refer to this model as the *permutation* formulation. This model was also given by Hansen et al. [22] (as ‘quasi-positive lower bound’) and Meyer and Jaumard [37] (as (HJM99-B)), albeit in a different form. Using our notation λ for the variables and σ for the permutations, formulation (HJM99-B) from [37] is:

$$\begin{array}{lll}
\max & z & \\
\text{s.t.} & \displaystyle\sum_{j=2}^{n} \lambda_{1j} = 1 & \\
& \displaystyle\sum_{i=1}^{k-1} \lambda_{ik} - \sum_{j=k+1}^{n} \lambda_{kj} = 0 & \text{for all } k = 2, \ldots, n-1 \\
& \displaystyle\sum_{i=1}^{n-1} \lambda_{in} = 1 & \\
& \displaystyle z \leq \sum_{i<j} \lambda_{ij}\sigma_{ij} & \text{for all } \sigma \in S_{\binom{n}{2}} \\
& \lambda \geq 0 & \\
& \lambda_{1n} = 0. &
\end{array} \tag{2.9}$$

Proposition 4. *Formulation (2.8) is equivalent to formulation (2.9).*

Proof. We show that condition (2.5) can be reformulated into the first three constraints of model (2.9). Consider condition (2.5) for a given $k \geq 2$ and $k-1$, i.e., $\sum_{i \leq k-1 < j} \lambda_{ij} = 1$ and $\sum_{i \leq k < j} \lambda_{ij} = 1$. We have

$$\sum_{i=1}^{k-1} \sum_{j=k}^{n} \lambda_{ij} - \sum_{i=1}^{k} \sum_{j=k+1}^{n} \lambda_{ij} = 0.$$

We can reformulate this equation as

$$\left(\sum_{i=1}^{k-1} \sum_{j=k+1}^{n} \lambda_{ij} + \sum_{i=1}^{k-1} \lambda_{ik} \right) - \left(\sum_{i=1}^{k-1} \sum_{j=k+1}^{n} \lambda_{ij} + \sum_{j=k+1}^{n} \lambda_{kj} \right) = 0,$$

or

$$\sum_{i=1}^{k-1} \lambda_{ik} - \sum_{j=k+1}^{n} \lambda_{kj} = 0,$$

which is the second set of constraints in our model (2.9). The constraints $\sum_{j=2}^{n} \lambda_{1j} = 1$ and $\sum_{i=1}^{n-1} \lambda_{1j} = 1$ follow from considering condition (2.5) with respect to $k = 1$ and $k = n-1$, respectively. ☐

The correspondence between model (2.8) and our Lagrangian relaxation is that by solving model (2.8) we obtain optimal values for λ with respect to the Lagrangian relaxation, and both models will provide the same objective value. Unfortunately, solving the permutation model directly is non-trivial since we have about $\binom{n}{2}!$ constraints. Therefore, Hansen et al. [22] introduced an scheme that iteratively solves the permutation model (2.8) for a subset of constraints $C \subset S_{\binom{n}{2}}$:

$$\begin{array}{lll} \max & z & \\ \text{s.t.} & \displaystyle\sum_{i \le k < j} \lambda_{ij} = 1 & \text{for all } k = 1, \ldots, m-1, \\ & z \le \displaystyle\sum_{i<j} \lambda_{ij} \cdot \sigma_{ij} & \text{for all } \sigma \in C. \end{array} \tag{2.10}$$

Algorithm 3 Iterative algorithm for solving the permutation formulation.

```
1:  Choose any initial $C$
2:  Solve (2.10) and let $z$ be the objective value
3:  Sort $\lambda$ into non-increasing order
4:  For $i < j$ let $\sigma_{ij} =$ (the position of $\lambda_{ij}$ in sorted order)
5:  if $z = \lambda \cdot \sigma$ then
6:      terminate
7:  else
8:      $C := C \cup \{\sigma\}$
9:      Goto 2
10: end if
```

The associated algorithm is presented in Algorithm 3. For a given set C, we solve model 2.10 and obtain objective value z together with optimal values for λ (lines 1-2). We then apply Proposition 3 and consider the values of λ in non-increasing order to define the permutation σ by setting σ_{ij} to the ordered position of λ_{ij} (lines 3-4). We note that Hansen et al. [22] use the same sorting procedure to solve the subproblem. Observe that z provides a lower bound on the ruler, whereas the product $\lambda \cdot \sigma$ provides an upper bound. If these bounds meet optimality is proved (lines 5-6). Otherwise, we add the permutation σ to C and continue (lines 7-10). This algorithm not only provides a procedure for solving the permutation formulation (2.8), but it can also serve as a systematic alternative approach to our subgradient method for finding optimal values for λ.

2.4.2 Equation Sums Bound

We next study the relationship of the Lagrangian relaxation with the lower bounds proposed by Galinier et al. [18]. We first recall these lower bounds by illustration with an example.

Let $n = 5$, for which the length of the ruler is given by d_{15}. If we want to bound d_{15}, we can first divide this segment into sub-segments in different ways:

$$\begin{aligned} d_{15} &= d_{12} + d_{23} + d_{34} + d_{45} \\ d_{15} &= d_{13} + d_{35} \\ d_{15} &= d_{12} + d_{24} + d_{45} \end{aligned}$$

Multiplying each equation by $\frac{1}{3}$ and adding them together gives

$$d_{15} = \frac{2}{3}(d_{12} + d_{45}) + \frac{1}{3}(d_{23} + d_{34} + d_{13} + d_{24} + d_{35}) \ .$$

Since all these numbers will be distinct naturals, we get

$$\begin{aligned} d_{15} &\geq \frac{2}{3}(1+2) + \frac{1}{3}(3+4+5+6+7) \\ d_{15} &\geq 10.333 \end{aligned}$$

(Recall that the optimal length of a Golomb ruler with 5 marks is 11, so after rounding this bound is tight.)

There are, of course, many ways we can write out d_{1n} as a sum of smaller segments, and [18] discuss several options. We will refer to bounds of this form as *equation sums* bounds. Another option we have is to weight the equations differently. For example, we could have given the first two equations weights of 0.4 and the last equation a weight of 0.2 instead of giving them all a weight of $\frac{1}{3}$. This would result in the equation

$$d_{15} = 0.6(d_{12} + d_{45}) + 0.4(d_{23} + d_{34} + d_{13} + d_{35}) + 0.2(d_{24})$$

and the corresponding bound

$$\begin{aligned} d_{15} &\geq 0.6(1+2) + 0.4(3+4+5+6) + 0.2(7) \\ d_{15} &\geq 10.4 \end{aligned}$$

We will refer to bounds of this form as *generalized equation sums* bounds.

Proposition 5. *The generalized equation sums bounds are equivalent to* z_{LR} *for an appropriate choice of* λ*.*

Proof. The weights of the equations in the generalized equation sums bound must always be distributed so that they sum to 1. This way d_{1n} always gets a coefficient of 1, and we always end up with a bound of the form $d_{1n} \geq \sum \mu_{ij} d_{ij}$ for some coefficients μ_{ij}. Note that in each equation for each $k = 1, \ldots, n-1$, there is some term that encapsulates the segment $(k, k+1)$. That is, there is some d_{ij} such that $i \leq k < j$. Since the weights on each equation sum to one, the coefficients that encapsulate the pair $(k, k+1)$ should sum to 1. That is: $\sum_{i \leq k < j} \mu_{ij} = 1$. Then to find the minimum value of $\sum_{i<j} d_{ij}\mu_{ij}$ we simply sort (μ) into decreasing order and assign each d_{ij} the corresponding value. This is precisely what we did in Proposition 3. □

This shows that although the bound from [18] is weaker than the LP bound, it can be generalized to be as strong as the LP bound, and it gives a nice intuition for our constraint on λ.

2.5 Computational Results for Approximating the LP Bound

The purpose of the computational experiments in this section is twofold. First, we wish to obtain insight into the performance of our subgradient optimization scheme relative to the systematic iterative scheme based on the permutation formulation for solving the Lagrangian relaxation. Second, we wish to evaluate our Lagrangian relaxation with the state of the art for solving the LP relaxation.

2.5.1 Subset Formulation

The current fastest method for solving the LP relaxation for the Golomb ruler problem was proposed by [37]. It is based on the following formulation of the lower bound, proposed in [48]. Let $\mathcal{S} = \{(i,j) : i < j\}$ and let $\mathcal{P}(\mathcal{S})$ be the power set of $\mathcal{S}$.

$$\begin{array}{lll} \min & d_{1n} & \\ \text{s.t.} & \displaystyle\sum_{k=i}^{j-1} d_{k,k+1} = d_{ij} & \text{for all } 1 \le i < j \le n, \\ & \displaystyle\sum_{(i,j)\in R} d_{ij} \ge \frac{1}{2}|R| \cdot (|R|+1) & \text{for all } R \in \mathcal{P}(\mathcal{S}). \end{array} \tag{2.11}$$

We will call this the *subset* formulation. Again, this LP is too big to solve as stated since it has $O(2^{\binom{n}{2}})$ constraints. However, [37] proposes an iterative solving method in which we only include the second constraint above for some subset $\mathcal{T} \subset \mathcal{P}(\mathcal{S})$. The associated algorithm is presented in Algorithm 5. Since this approach is the currently best known algorithm for finding the LP bound, we will compare our methods to this.

Algorithm 5 Iterative algorithm for solving the subset formulation.

```
1:  Let $\mathcal{T} = \{\{(i,i+1)\} : 1 \le i < n\} \cup \{\{(i,i+1) : 1 \le i < n\}\}$
2:  Solve (2.11)
3:  Sort $(d_{ij})$
4:  for $1 \le k \le \binom{n}{2}$ do
5:      Let $T = \{(i,j) : d_{ij}$ is within the first $k$ positions$\}$
6:      if $\sum_{(i,j)\in T} d_{ij} < \binom{k}{2}$ then
7:          $\mathcal{T} := \mathcal{T} \cup \{T\}$
8:      end if
9:  end for
10: if $\mathcal{T}$ has changed then
11:     goto Line 2
12: end if
```

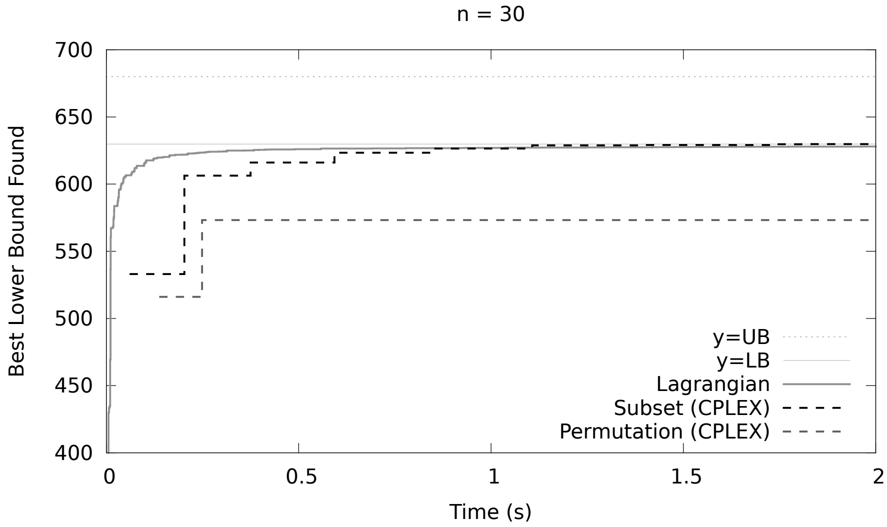


Figure 2.3: Performance comparison between the permutation, subset, and Lagrangian formulations, for $n = 30$. The best known upper bound and LP lower bound are also depicted.

2.5.2 Implementation and Results

We implemented the Lagrangian relaxation and the subgradient method in C++, following the description in Section 2.3.3. It was run using C++ on an Intel core i3 processor (2.13 GHz). The times reported are the number of seconds elapsed between when the program started running and when that lower bound was found.

We implemented the solving schemes for the subset formulation and the permutation formulation in AIMMS, using CPLEX 12.4 as linear programming solver. The AIMMS implementations were run on the same Intel core i3 processor. The times reported are the sums of the solving times for each call to CPLEX, i.e., we eliminate the overhead that AIMMS may add. We apply all methods to find bounds for $n = 30$, 40, 50, and 60 and 130. The exact LP bound could be computed within a time limit of 600 seconds for n up to 60.

We report a comparison of the performance between the three methods in Figures 2.3, 2.4, and 2.5. In each figure, we report the bound obtained by each method over time. We also report the best known upper bound and LP lower bound (if known).[3] The permutation formulation did not provide meaningful bounds for n larger than 30, and is not depicted for $n = 40$ up to 130. From the figures we can see that even though the Lagrangian bound may not precisely attain the LP bound, it finds bounds that are near the optimal LP bound, much

---

[3] See `http://www.research.ibm.com/people/s/shearer/grtab.html` for the list of shortest known rulers.

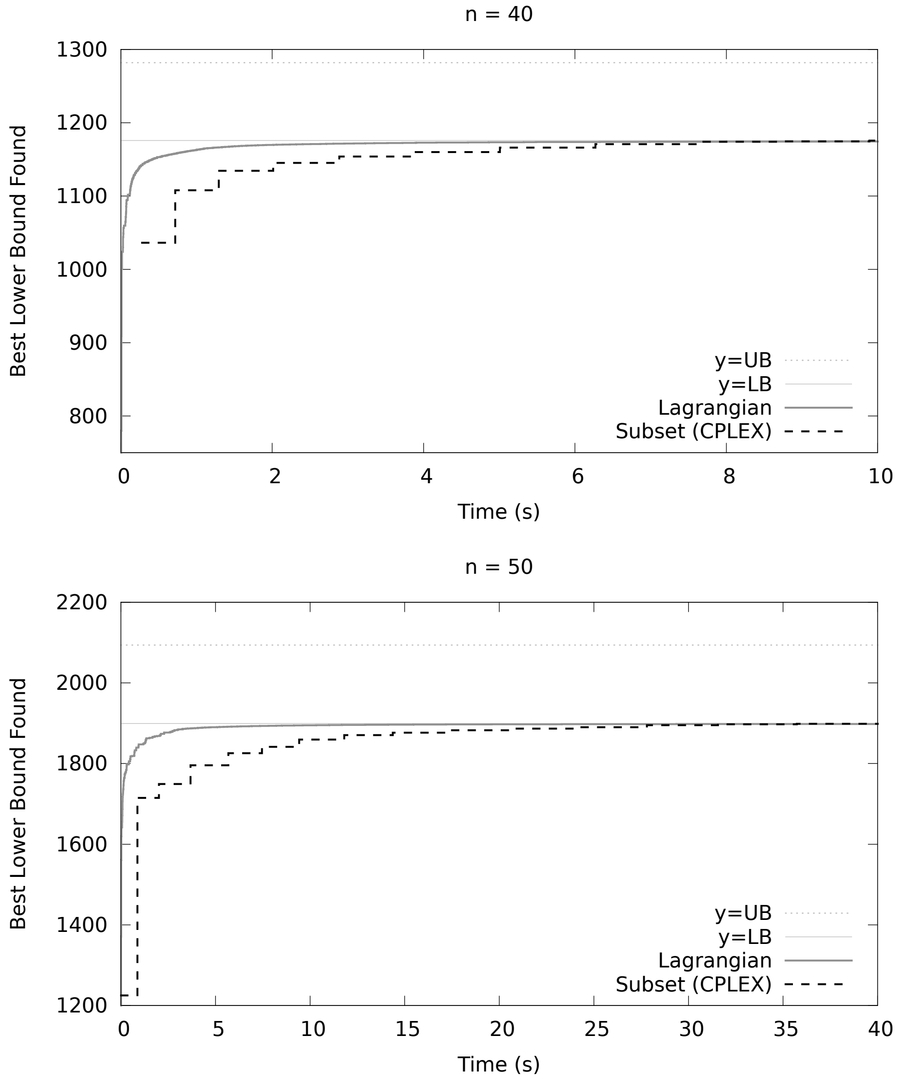


Figure 2.4: Performance comparison between the subset and Lagrangian formulations, for $n = 40, 50$. The best known upper bound and LP lower bound are also depicted.

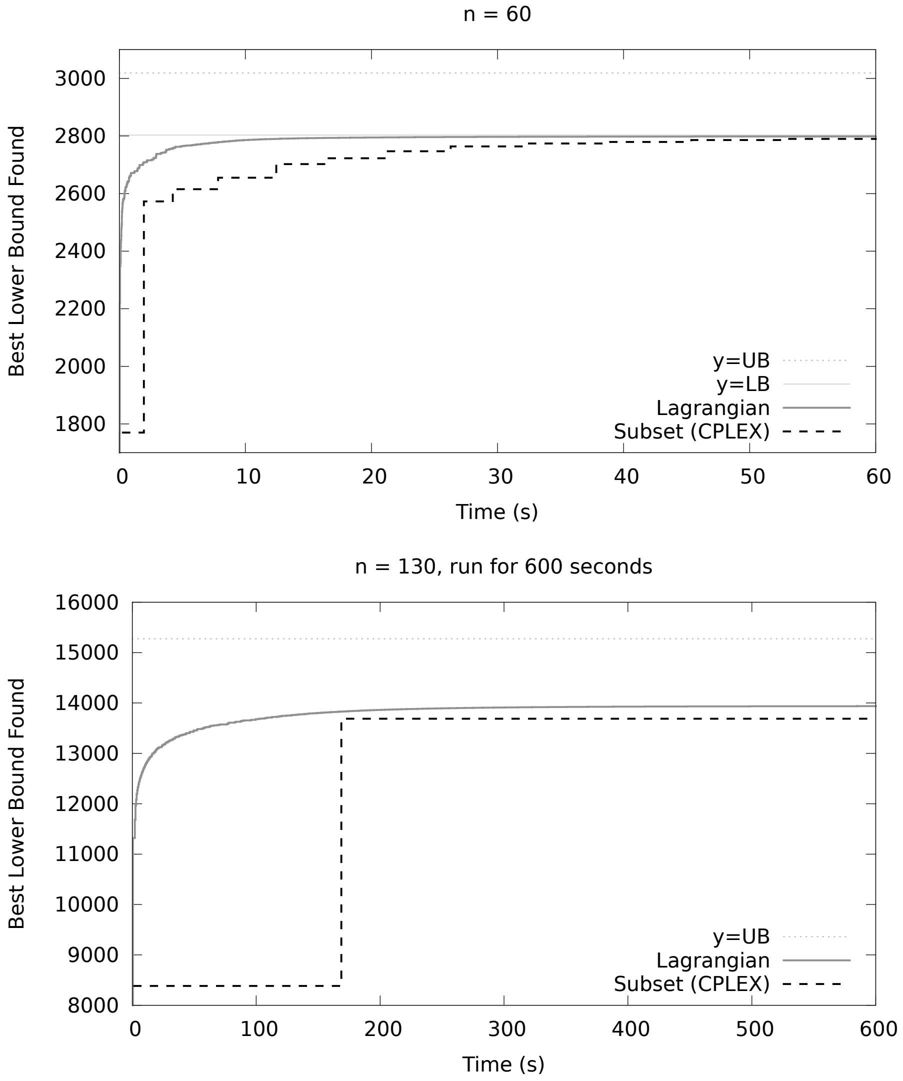


Figure 2.5: Performance comparison between the subset and Lagrangian formulations, for $n = 60, 130$. The best known upper bound (and LP lower bound for $n = 60$) is also depicted.

n	initial stepsize	running time	lower bound
1,000	1	1 hour	698,743
2,000	10^{-6}	1 hour	2,589,350
5,000	10^{-6}	1 hour	10,749,700
10,000	10^{-7}	3 hours	59,417,700

Table 2.1: Lagrangian bounds for large Golomb rulers.

faster than the scheme for the subset formulation does.

We also applied our scheme to obtain lower bounds for Golomb rulers of size 1,000, 2,000, 5,000 and 10,000. At this scale, we needed to adapt the initial stepsize of our algorithm with respect to the earlier experiments. Our results are summarized in Table 2.1. For each size, we report the initial stepsize, the running time, and the Lagrangian bound we obtained. For these instances, we could not obtain meaningful LP-based lower bounds using the existing schemes, including the subset formulation.

2.6 Lagrangian Bounds for Ruler Segments

We can apply our methodology to bound not just the length of the ruler, d_{1n}, but also the lengths of the other segments, d_{ij}. Observe that in a Golomb ruler the way to minimize d_{ij} is to make the segments between marks i and j follow an optimal ruler of $j - i + 1$ marks, and make the other distances as large as they need to be.

Recall that our problem statement takes as input n marks and an upper bound L. For the purpose of producing a lower bound on a ruler segment, we consider in this section the satisfiability problem, "does there exist a Golomb ruler of n marks of length L?"

Our first step towards producing a lower bound for d_{ij} is to minimize the following objective

$$\sum_{k=i}^{j-1} d_{k,k+1} + \sum_{k=1}^{n-1} d_{k,k+1}$$

subject to the constraints in our original matching formulation (2.2). Let z^* be the result of solving this LP. Then, for any Golomb ruler, we have

$$d_{ij} + d_{1n} = \sum_{k=i}^{j-1} d_{k,k+1} + \sum_{k=1}^{n-1} d_{k,k+1} \geq \lceil z^* \rceil.$$

Since we are assuming $d_{1n} = L$, we obtain the following bound on the segment:

$$d_{ij} \geq \lceil z^* \rceil - L.$$

Using a similar argument, we can get a bound on d_{ij} by minimizing

$$z(a,b) := a \sum_{k=i}^{j-1} d_{k,k+1} + b \sum_{k=1}^{n-1} d_{k,k+1}$$

for any $a, b \in \mathbb{N}$. Letting $z^*(a,b)$ be the optimal objective value, this yields the lower bound

$$d_{ij} \geq \left\lceil \frac{\lceil z^*(a,b) \rceil - bL}{a} \right\rceil \ . \tag{2.12}$$

To compute $z^*(a,b)$, we modify Algorithm 2 by replacing line 4 with

$$\begin{aligned} \lambda_{k,k+1} &:= (a+b) - \sum_{\substack{i \leq k < j \\ i+1 \neq j}} \lambda_{ij} && \text{for } k \in \{i, \ldots, j-1\}, \\ \lambda_{k,k+1} &:= b - \sum_{\substack{i \leq k < j \\ i+1 \neq j}} \lambda_{ij} && \text{for } k \in \{1, \ldots, i-1\} \cup \{j, \ldots, k-1\}. \end{aligned}$$

This approach can be extended to bound arbitrary linear combinations of segments as well, for example $d_{ij} + d_{i'j'}$. In our implementation we only consider segments of the form $d_{i,n-i+1}$ in order to exploit symmetry.

The quality of the bound from equation (2.12) depends on the choice of a and b. Since we do not have access to a closed-form solution for $z^*(a,b)$ we apply a search procedure to find appropriate values. The details of this procedure are presented in AppendixA.

2.7 Embedding in Exact Constraint Programming Search

An important motivation for our work is that existing methods for computing the LP bound are too expensive to be applied in an enumerative search scheme such as LP-based branch-and-bound to find provably optimal solutions. To assess the application of our method in this context, we embedded our Lagrangian relaxations into an exact systematic search procedure. Instead of LP-based branch-and-bound, we chose to develop a constraint programming (CP) search procedure. The main reason for this is the combinatorial nature of the Golomb ruler problem. Especially for highly combinatorial problems, the efficient constraint propagation techniques of CP may outperform conventional integer programming technology based on LP relaxations; we refer to Hooker [27] for a comparison of CP and conventional LP-based optimization technology as well as integrated approaches. In particular, Lagrangian relaxations have been successfully applied before in CP to improve the optimization reasoning [5, 11, 36, 45, 46].

2.7.1 Constraint Programming Representation and Search

We implemented a CP search program that, given n and L, finds all n-mark Golomb rulers of length L. It is implemented as a constraint satisfaction problem with 'set variables' [19]. A set variable will be assigned a set of values in a solution. In our case, we introduce two set variables: X, the set of marks in the ruler, and D, the set of distances measured by the ruler.

The domain of possible sets for each set variable is maintained via a 'lower bound' of mandatory elements (denoted by X^- and D^-) and an 'upper bound' of possible elements (denoted by X^+ and D^+). In addition, we require the sets to be of a given cardinality.[4] Our constraints are as follows:

$$\begin{array}{l} X = \{x_1, x_2, \ldots, x_n\} \in [\{0, L\}, \{0, \ldots, L\}] \\ |X| = n \\ D \in [\{L\}, \{1, \ldots, L\}] \\ |D| = \binom{n}{2} \\ d \in D \iff \exists x_i, x_j \in X \text{ s.t. } x_j - x_i = d \\ x_2 - x_1 < x_n - x_{n-1} \end{array} \tag{2.13}$$

We use the notation $[A, B]$ where A and B are sets to mean $\{U : A \subset U \subset B\}$. The first five constraints represent the conditions for the Golomb ruler, while the last constraint is added to break symmetry (we require the first distance d_{12} to be smaller than the last distance $d_{n-1,n}$).

Our search procedure is described in Algorithm 6, and initially called with 'BRANCH($\{0, L\}$, $\{0, \ldots, L\}$, $\{L\}$, $\{1, \ldots, L\}$)'. The procedure is called recursively (lines 25, 30, and 33), and either returns a solution (line 3) or reports that no solution exists (line 6). Note that infeasibility can be detected by inspecting whether $|D^+| < \binom{n}{2}$ (line 5).

Lines 9-20 implement the domain filtering rules that reflect the fifth constraint in model (2.13). Lines 9-17 remove possible elements from X^+ based on the current state of mandatory elements in X^- and D^-. Lines 18-20 in addition remove from X^+ the midpoint of two values $x, y \in X^-$ such that $x \equiv y \bmod 2$. Namely, the midpoint $(x + y)/2$ would create an equal distance to x and y.

Lines 22-33 implement the search strategy. We note that line 23 ensures that between any ruler and its mirror image only one is found by this program, reflecting the last constraint in model (2.13). The search strategy considers each distance $d \in \{1, \ldots, L\}$ in decreasing order and decides if and where d will be measured in the ruler. This strategy is based on the following result.

Proposition 6. *If we have already decided if and where to measure the lengths* $\{d + 1, \ldots, L\}$*, and we have not decided if and where to place* d*, then the only place* d *can be measured is from 0 to* d *or from* $L - d$ *to* L*.*

Proof. For the sake of contradiction, suppose there are $x, x + d \in X^+$ with $0 < x < L - d$. Then since we have decided if and where to place the distance

---

[4] This domain representation for set variables is referred to as the 'subset+cardinality' domain ordering in the literature [19].

Algorithm 6 Our branching algorithm for finding Golomb Rulers

```
procedure BRANCH($X^-, X^+, D^-, D^+$)
    if $|X^-| = n$ then
        output $X^-$ and terminate
    end if
    if $|D^+| < \binom{n}{2}$ then
        return
    end if

    $D^- := \{x_j - x_i : x_i, x_j \in X^-\}$
    for $x \in X^-$ and $d \in D^-$ do
        if $x - d \in X^+ \setminus X^-$ then
            $X^+ := X^+ \setminus \{x - d\}$
        end if
        if $x + d \in X^+ \setminus X^-$ then
            $X^+ := X^+ \setminus \{x + d\}$
        end if
    end for
    for $x, y \in X^-$ with $x \equiv y \bmod 2$ do
        $X^+ := X^+ \setminus \{\frac{x+y}{2}\}$
    end for

    $d^+ := \max(D^+ \setminus D^-)$
    if $|X^-| > 2$ then
        if $d^+ \in X^+ \setminus X^-$ then
            BRANCH($X^- \cup \{d^+\}, X^+, D^-, D^+$)
            $X^+ := X^+ \setminus \{d^+\}$
        end if
    end if
    if $L - d^+ \in X^+ \setminus X^-$ then
        BRANCH($X^- \cup \{L - d^+\}, X^+, D^-, D^+$)
        $X^+ := X^+ \setminus \{L - d^+\}$
    end if
    BRANCH($X^-, X^+, D^-, D^+ \setminus \{d^+\}$)
end procedure
```

$x+d$, and we know 0 will be a mark in our ruler, we already know whether we are including the mark $x+d$. Similarly, since $d < L-x$, we already know if and where we are including the distance $L-x$ and since we are including the mark L, we also know whether we are including the mark x. If we had decided to include both x and $x+d$, then we would not need to decide on the distance d. Thus if d is the largest distance we have not decided whether or not to include, we only need to consider three possibilities: the mark d is in the ruler, the mark $L-d$ is in the ruler, or the distance d is not measured by the ruler. $\square$

Thus every search node has at most 3 children: Either we add a mark at d^+ (line 25), add a mark at $L-d^+$ (line 30), or remove d^+ from the possible measurable distances (line 33). The third branch drives us closer to pruning at lines 5-6.

Throughout our search, we maintain the invariant that the unknown marks are consecutive, and therefore when we add a mark to X^- (lines 20 and 23), we know its index. This is useful as we will see in section 2.7.3, when we incorporate bounds on segments.

2.7.2 Adding the Lagrangian Relaxation

A direct application of our Lagrangian relaxation into the CP search procedure is to re-evaluate the relaxation at each node of the search tree (to be added in line 8 of Algorithm 6). If the value of the Lagrangian bound exceeds L we return 0 and prune the associated search node.

We note that we compute the Lagrangian multipliers once at the root node, and these remain fixed during the remainder of the CP search. Recomputing the multipliers during search proved to be too computationally expensive. Furthermore, we experimented with applying the Lagrangian multipliers for variable fixing during search (which has been done successfully before in CP), but found that the impact on the overall solution time was marginal.

The computational results of this first experiment are summarized in Table 2.2—more details can be found in [50]. The table lists results for Golomb ruler instances with $n = 10, \ldots, 13$ marks. For each mark we report the time to find a corresponding optimal ruler of length L (feasible), or to prove no ruler exists for length $L-1$ (infeasible) – this is listed in the third column as 'F' for feasible and 'I' for infeasible. In addition to the solution time for each method we report the number of search nodes. The fourth column $\frac{\text{LB}}{\text{UB}}$ represents the strength of the LP lower bound for each instance.

We can observe that in all cases the Lagrangian relaxation reduces the number of search nodes, sometimes by as much as 90% (for $n = 10$, $L = 54$). The Lagrangian relaxation does not appear to speed up the algorithm when we are searching for a ruler, but it can speed up the algorithm when we are trying to prove a ruler does not exist. Note also that the impact highly depends on the strength of the LP bound. When the LP bound is weak, the computational overhead can be too large to reduce the overall computation time (e.g., for $n = 11$, $L = 71$).

				CP		CP+LR	
n	L	F/I	$\frac{\text{LB}}{\text{UB}}$	nodes	time (s)	nodes	time (s)
10	54	I	98%	60,554	0.51	4,984	0.11
10	55	F	98%	4,492	0.04	3,512	0.07
11	71	I	93%	2,993,876	27.09	2,055,429	37.29
11	72	F	93%	5,581	0.05	5,343	0.11
12	84	I	96%	10,298,716	103.62	2,773,734	59.04
12	85	F	96%	7,103,301	70.84	4,698,798	96.17
13	105	I	92%	445,341,835	4,782	273,340,407	6,618
13	106	F	92%	205,714,305	2,187	177,429,879	4,278

Table 2.2: The performance of the CP search without (CP) and with the Lagrangian relaxation applied at each search node (CP+LR). For each instance with n marks and upper bound L, F stands for ‘feasible’, while I stands for ‘infeasible’. We also report the strength of the LP lower bound over the best known upper bound as $\frac{\text{LB}}{\text{UB}}$ for each instance. For each method, we report the total number of search nodes and the solving time in seconds.

2.7.3 Adding Bounds on Ruler Segments

We next consider the impact of adding bounds on different segment lengths, as described in Section 2.6. To apply a bound on the segment from mark i to $n-i+1$, we pre-compute the segment bound B_i before we start the search. We can apply as many of these segment bounds as we like for segment $(i, n-i+1)$, $i = 2, \ldots, n/2$. Recall from Section 7.1 that at each search state we have fixed (defined) marks $1, \ldots, k$ and $j, \ldots, n$. We can then consider the largest possible distance $d^+ = \max(D^+\ D^-)$ to be added to the ruler. If this distance is smaller than any pre-computed lower bound B_i, we inspect the endpoints of i and $n-i+1$. If either mark has not yet been defined, we are unable to complete the ruler since $B_i > d^+$. If both endpoints x_i and x_{n-i+1} are defined but their distance is smaller than B_i we again reach a conflict, and backtrack. Note that this last case can only occur directly after a new mark has been defined and D^- has not yet been updated.

We can add a bound of the form $d_{i,n-i+1} \geq B_i$ in Algorithm 6 after line 8 as follows.

```
8.1  $d^+ = \max(D^+ \setminus D^-)$
8.2  if ($d^+ < B_i$)
8.3      if  ($x_i$ undefined
8.4          or $x_{n-i+1}$ undefined
8.5          or $x_{n-i+1} - x_i < B_i$)
8.6              return 0
```

We now consider the impact of adding bounds of sums of segments. In particular, on $d_{2,n-1} + \ldots + d_{i,n-i+1}$. We can apply as many such bounds as we like. For each one, we pre-compute the bound on this sum, C_i. At some point in the algorithm, we will have fixed marks for $1, \ldots, i$ and $n-i+1, \ldots, n$. At this point, we test if the bound $d_{2,n-1} + \ldots + d_{i,n-i+1} \geq C_i$ is satisfied, and if it is not we reach a conflict and backtrack.

We apply a sum bound of the form $d_{2,n-1} + \ldots + d_{i,n-i+1} \geq C_i$ after the Lagrangian relaxation in line 8 of Algorithm 6, as follows.

```
8.7  if ($x_i$ and $x_{n-i+1}$ are defined)
8.8      if  (($x_{n-1} - x_2$) + ... + ($x_{n-i+1} - x_i$) < $C_i$)
8.9          return 0
```

We evaluated the impact on the search procedure using bounds on various segments. We refer to our tests by the number of segments that we bound. In these tests we computed bounds for and filtered on the following segments.

- Baseline: None
- Bound 1: $d_{2,n-1}$
- Bound 2: $d_{2,n-1}, d_{3,n-2}$
- Bound 3: $d_{2,n-1}, d_{3,n-2}, d_{4,n-3}$
- Bound all: $d_{2,n-1}, \ldots, d_{n/2,n/2+1}$

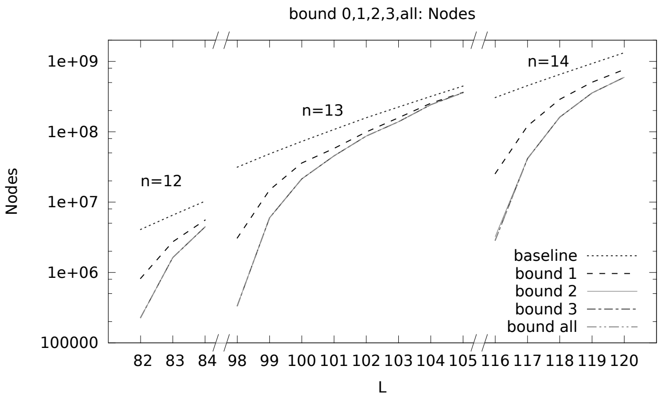
The results are presented in Figure 2.6. We compare the impact with respect to the baseline CP search procedure that does not apply any bounding.

We found that in the range $L \leq 120$, the bounds on $d_{2,n-1}$ and $d_{3,n-2}$ are helpful in that they reduce both the size of the search tree and also the search time. However adding in the third bound, $d_{4,n-3}$ and adding the rest of them did not improve the size of the search tree and thus only hindered our time because we still had to compute the relevant bounds.

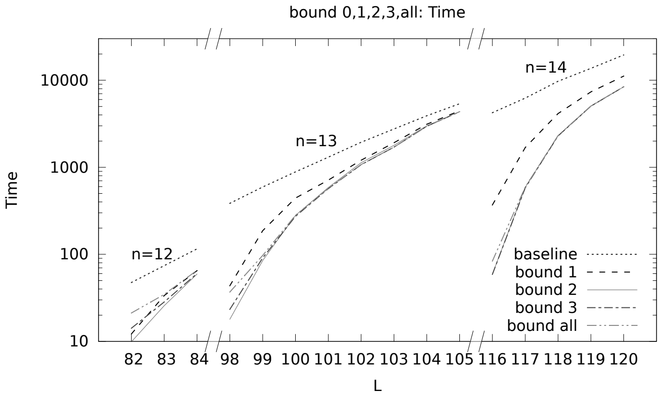
We tested our search procedure on $n = 12, 82 \leq L \leq 84$, $n = 13, 98 \leq L \leq 103$, and $n = 14, 116 \leq L \leq 120$. These are all infeasible instances. In Figure 2.6 we report the number of nodes in the search tree and number of seconds to prove infeasibility respectively. In Figure 2.6a, "bound 2", "bound 3", and "bound all" are directly on top of one another showing that the search tree size does not improve when we add more bounds than bound 2.

We also used bounds on sums of the form $s_k = \sum_{\ell=2}^{k} d_{\ell,n-\ell+1}$. Since segment bounds on $d_{2,n-1}$ and $d_{3,n-2}$ proved to be useful, we continued using them in our tests. The graphs in Figure 2.7 show our results in computing bounds for and filtering on the following.

- baseline: $d_{2,n-1}, d_{3,n-2}$
- 1 sum bound: $d_{2,n-1}, d_{3,n-2}, C_3$



(a) Impact with respect to search nodes.



(b) Impact with respect to solution time

Figure 2.6: Evaluating the impact of ruler segment bounds in terms of search nodes (a) and time (b), when bounding $d_{2,n-1}$, bounding $d_{2,n-1}$ and $d_{3,n-2}$, bounding $d_{2,n-1}$, $d_{3,n-2}$ and $d_{4,n-3}$, and bounding all segments of the form $d_{i,n-i+1}$. Tested are infeasible Golomb ruler instances with $n = 12, 13, 14$ marks and varying lengths L.

- 2 sum bounds: $d_{2,n-1}, d_{3,n-2}, C_3, C_4$
- 3 sum bounds: $d_{2,n-1}, d_{3,n-2}, C_3, C_4, C_5$
- 4 sum bounds: $d_{2,n-1}, d_{3,n-2}, C_3, C_4, C_5, C_6$
- 5 (all) sum bounds: $d_{2,n-1}, d_{3,n-2}, C_3, C_4, C_5, C_6, C_7$

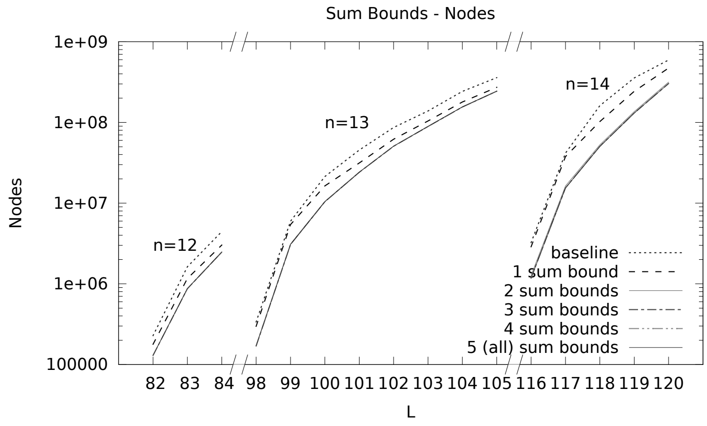
For these “sum bounds”, we found that s_3 and s_4 significantly prune the size of the search tree, and beyond that, we do not save enough time in the search to justify the computation of the bounds.

Figure 2.7 is the analog of Figure 2.6 for these tests.

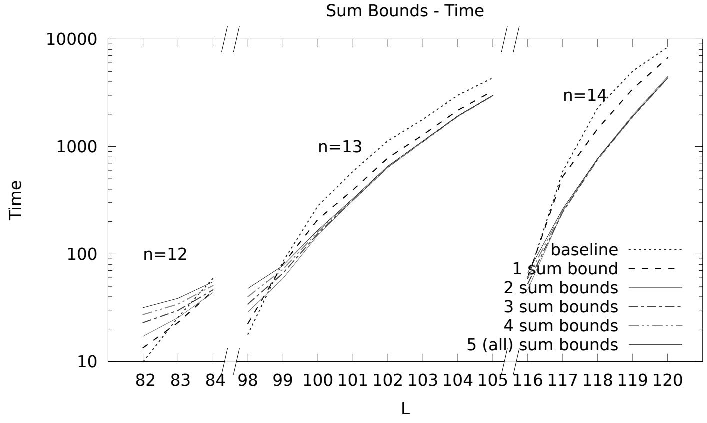
2.8 Conclusion

We have presented a new way to approximate the LP bound for Golomb Rulers. We have demonstrated its relationship to existing methods, and shown that we can compute the LP bound much faster using combinatorial methods.

We then used this fast computation in a search procedure, demonstrating that we can use this bound to reduce the size of the search tree and, in cases where the LP bound is strong enough, reduce the search time as well.



(a) Impact with respect to search nodes.



(b) Impact with respect to solution time (in seconds)

Figure 2.7: Evaluating the impact of ruler segment bounds in terms of search nodes (a) and time (b), when bounding $d_{2,n-1} + d_{3,n-2}$, bounding $d_{2,n-1} + d_{3,n-2} + d_{4,n-3}$, $d_{2,n-1} + d_{3,n-2} + d_{4,n-3} + d_{5,n-4}$ etc. Tested are infeasible Golomb ruler instances with $n = 12, 13, 14$ marks and varying lengths L.

Chapter 3

Residual Costs from Relaxed Decision Diagrams

3.1 Introduction

In 1938, Shannon [47] introduced boolean algebra to the field of switching circuits as a way to represent and evaluate them more formally. Lee introduced binary decision programs in 1958 as a more procedural alternative [33]. He showed that binary-decision programs have smaller representations and allow for faster computation, since each variable only needs to be considered once.

The phrase "binary decision diagrams," or BDDs, was coined by Akers [2] who first considered them as graphical structures. He showed how to construct them from a truth table, how to represent them as a data structure in a computer, and how they can be used to test Boolean circuits.

Binary decision diagrams gained popularity after Bryant's seminal paper [10] which proved that for a given variable ordering, there is a unique reduced BDD. Using this he showed how to combine BDDs together. Specifically, given BDDs for functions f_1 and f_2, he showed how to construct BDDs for $f_1 \wedge f_2$ and $f_1 \vee f_2$. This provides a method for building a BDD for a Boolean function based on its algebraic representation. He also showed that although BDDs can theoretically be exponential in size, most functions that are used in practice have much smaller BDD representations.

In this chapter, we discuss multi-valued decision diagrams (MDDs). In contrast with BDDs, MDD represent functions of variables whose variables and output can take on values from any set (rather than just $\{0, 1\}$). The idea of MDDs was introduced in [53]. This paper extends Bryant's results about BDDs including the uniqueness of an reduced MDD with a given variable ordering.

Since then, MDDs have been used for combinatorial optimization. They have been used for generating cutting planes in a branch and cut algorithm, computing max flow, counting solutions to a constraint program, and learning in genetic programming [4, 54]. They have also been used for post-optimality analysis [20].

Approximate MDDs are MDDs which represent a superset or subset of the solutions to a given problem but which use less memory. The idea was first proposed in [3], in which they use approximation MDDs to efficiently represent constraint stores in constraint programming.

Since then much work has been done on approximate MDDs. A refinement algorithm for MDDs is presented in [21], where it is applied to interactive configuration. In [25], it is shown how to use MDDs to propagate constraints in constraint programming. In [13], MDDs are applied to the traveling salesperson problem with time windows, and used to solve some instances from the standard library of traveling salesperson instances which had been open. MDDs have also been applied to propagation of the sequence constraint [7], the maximum independent set packing and set covering problems in [6, 8], the bin packing problem [29], and clause generation for the SAT problem [30]

Additive bounding was first introduced in [16]. It is a method of taking several algorithms for finding a lower bound and combining them, resulting in a bound that is stronger than any of the component algorithms could have come up with individually. It uses residual costs, which measure the increase in the objective function when variable is forced into a solution. In [31], additive bounding was used to combine approximate MDDs with linear programming. However, the methodology did not employ residual costs from the MDD.

In this chapter, we show how to integrate Cire's decision diagram algorithm [13] into an additive bounding scheme, how we can extract residual costs, from an MDD, and we apply it to the sequential ordering problem.

3.1.1 Multi-valued Decision Diagrams

Let P be an optimization problem in the variables $x_1, \ldots, x_n$, each with a finite domain. A *multivalued decision diagram* (MDD) for P is a directed acyclic multigraph whose nodes can be partitioned into $n + 1$ levels, $L_1, \ldots, L_{n+1}$, so that L_0 and L_n have one node (the "source" and "sink" respectively), and so that every edge goes from a node in some level L_i to a node in the next level, L_{i+1}. Each arc, a, going from level L_i to level L_{i+1} corresponds to a value v_a in the domain of x_i. This way, a path from the source to the sink represents an assignment of values to the variables.

An *exact MDD* for an optimization problem, P, is one in which the paths correspond exactly to the feasible solutions to P.

Example 1. The minimum vertex cover problem asks, given a graph $G = (V, E)$, what is the minimum subset $U \subset V$ such that every edge is incident to a vertex of U? The variables are the 0/1 variables x_i for $i \in V$ with $x_i = 1$ iff $i \in U$.

Let G be the graph in Figure 3.1. Then an exact MDD for this problem is as in Figure 3.2, where a solid line in layer i corresponds to $x_i = 1$, and a dashed line corresponds to $x_i = 0$.

We say two nodes in an MDD are *equivalent* if the sequences of labels from the two nodes to the sink are the same. We can contract equivalent nodes

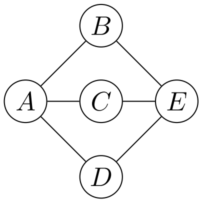


Figure 3.1: A graph for the minimum vertex cover example.

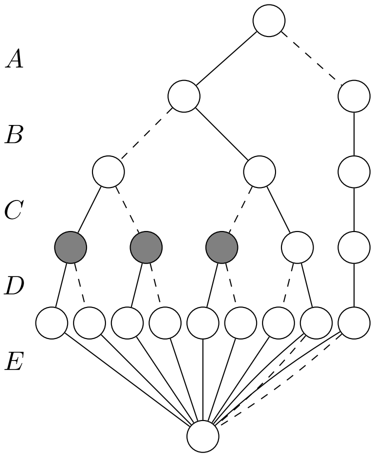


Figure 3.2: An exact MDD for the minimum vertex cover problem. The solid edges represent a value of 1; dashed lines represent a value of 0.

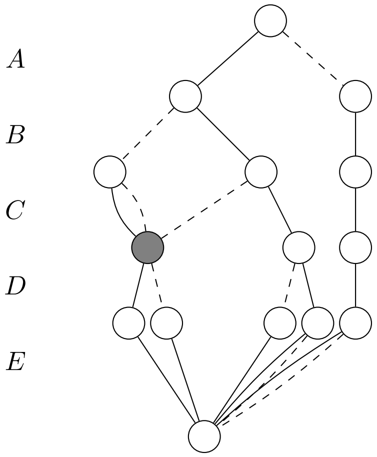


Figure 3.3: An exact MDD for the minimum vertex cover problem with 3 nodes contracted.

without changing the set of paths from the source to the sink.

Example 2. Continuing the example above, the shaded nodes in Figure 3.2 are equivalent since the sequences of labels from all shaded nodes are ($x_D = 1, x_E = 1$) and ($x_D = 0, x_E = 1$). The MDD in Figure 3.3 is equivalent to the MDD in Figure 3.2.

A *reduced MDD* is one in which there are no equivalent vertices. It is known [10] that every MDD has a unique equivalent reduced MDD.

Example 3. The reduced MDD for the MDD in Figure 3.2 is shown in Figure 3.4

A *relaxed decision diagram* is an MDD whose paths form a superset of the feasible solutions to the given problem. In our case, we will obtain relaxed decision diagrams by contracting vertices that are not equivalent.

Example 4. If we contract the two shaded nodes in Figure 3.4, we get the MDD in Figure 3.5. This MDD contains the path ($x_A = 0, x_B = 1, x_C = 1, x_D = 0, x_E = 1$) which corresponds to the set of vertices $\{B, C, E\}$. This set does not form a vertex cover as $\{A, D\}$ is an edge in our original graph. However, this MDD does contain all feasible vertex covers, so it is a relaxation.

3.1.2 Using MDDs to Solve an Optimization Problem

Let P be an optimization problem in the variables $x_1, \ldots, x_L$ with an objective function that can be expressed as $\min \sum f_i(x_i)$ (e.g., a linear or integer program), and let M be an exact MDD for P. We can use M to find an optimal solution to P as follows. For each arc a, in level i, let v_a be the associated value, and assign edge weight of $f_i(v_a)$ to arc a. Now to solve P, we simply need to

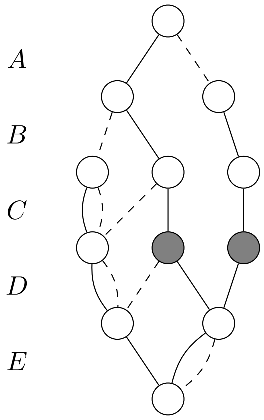


Figure 3.4: A reduced MDD for the minimum vertex cover problem.

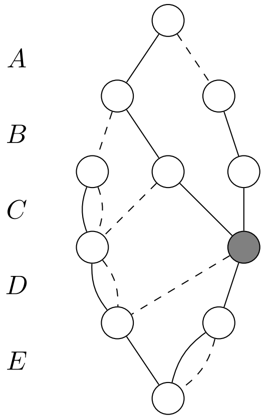


Figure 3.5: A relaxed MDD for the minimum vertex cover problem.

find a shortest path through M. If we follow the same procedure for a relaxed MDD M, we would get a lower bound.

3.1.3 Additive Bounding

The idea of additive bounding was introduced in [16].

Given a minimization problem,

$$\begin{array}{rl} \min & z(x) = c^T \cdot \mathbf{x} \\ \text{subject to} & \text{[some constraints]}, \end{array} \tag{3.1}$$

we would like to find a strong lower bound. There may be several algorithms that produce lower bounds which each take advantage of a different substructure of the problem. We could run each algorithm individually and take the maximum, but to exploit all the different substructures, we will use these algorithms together, resulting in a lower bound that is stronger than any algorithm can get individually.

An algorithm can be used for additive bounding if it solves a relaxation of (3.1) and outputs a solution vector, x^*, a lower bound, $z(x^*)$ and a residual cost vector $\overline{c}^T$ such that for any x that is feasible in (3.1) we have

$$z(x) \geq z(x^*) + \overline{c}^T \cdot x. \tag{3.2}$$

Intuitively, a residual cost is a lower bound on the amount by which the objective function increases if we force a particular variable into the solution. If we force several variables into the solution, the sum of the corresponding residual costs gives a lower bound on the increase of the objective function.

Taking a minimum of (3.2) gives us

$$\min z(x) \geq z(x^*) + \min(\overline{c}^T \cdot x).$$

This means that if we solve a relaxation of (3.1) using the objective value $z_1(x) = c^T \cdot x$ and get out x_1^*, $z_1(x_1^*)$ and $\overline{c_1}^T$, and then solve another relaxation of (3.1) using the objective value $z_2(x) = \overline{c_1}^T \cdot x$, getting out x_2^*, and $z_2(x_2^*)$, then $z_1(x_1^*) + z_2(x_2^*)$ is a valid lower bound for the original problem (3.1).

By chaining several such algorithms together, we get several lower bounds $z_1(x_1^*), z_2(x_2^*), \ldots, z_n(x_n^*)$, and summing them gives a lower bound on the original problem:

$$z(x) \geq \sum_i z_i(x_i^*)$$

As an example of such an algorithm, if we relax (3.1) to a linear problem, we can solve it and use the reduced costs from the linear program as the residual costs, $\overline{c}_T$.

In practice, additive bounding works best when the different algorithms take advantage of different structures within the problem. For example, in the traveling salesperson problem, one algorithm might take advantage of the connectivity constraint, and one algorithm might take advantage of the degree constraints.

Since approximate MDDs are a new relaxation, we would like to see if they exploit a different structure that can be used in additive bounding. In order to do this, we need a way to extract residual costs from the MDD that satisfy (3.2).

In Section 3.2 we show how to do this for the minimum vertex cover problem and the sequential ordering problem. In Section 3.3 we show experimental results from applying this method to the sequential ordering problem.

3.2 Extracting Residual Costs from an MDD

The problem of finding the shortest path through an MDD is itself an optimization problem, and the method of getting residual costs for the individual arcs is well known ([1]).

Proposition 7. *Let $G = (V, E)$ be a directed graph with a source and sink (s and t resp.), with edge weights w_{ij} for $(i, j) \in E$, and let ℓ be the length of the shortest path. If we assign a potential u_n to each node n so that $u_s = 0$ and $u_t = \ell$, and each arc gets a residual cost of $\overline{w}_{ij} := w_{ij} - (u_j - u_i)$, then for any path from $p = (s = n_0, n_1, n_2, \ldots, n_{L-1}, n_L = t)$, we have*

$$\sum_{i=0}^{L-1} w_{n_i,n_{i+1}} = \ell + \sum_{i=0}^{L-1} \overline{w}_{n_i,n_{i+1}}$$

Proof.

$$\begin{aligned}
\sum_{i=0}^{L-1} w_{n_i,n_{i+1}} &= \sum_{i=0}^{L-1} \overline{w}_{n_i,n_{i+1}} + (u_{n_{i+1}} - u_{n_i}) \\
&= \sum_{i=0}^{L-1} \overline{w}_{n_i,n_{i+1}} + \sum_{i=1}^{L} u_{n_i} - \sum_{i=0}^{L-1} u_{n_i} \\
&= \sum_{i=0}^{L-1} \overline{w}_{n_i,n_{i+1}} + u_{n_L} - u_{n_0} \\
&= \ell + \sum_{i=0}^{L-1} \overline{w}_{n_i,n_{i+1}}
\end{aligned}$$

□

This procedure gives us a way to extract residual costs from the MDD. In principle, if an arc, a, at level i associated with value v_a, has residual cost $\overline{w}_a$, then the residual cost of x_i should be $\frac{\overline{w}_a}{v_a}$ (if $v_a = 0$, then the residual cost has no effect on the objective function). However, since there are multiple arcs for each variable x_i, we need a way to decide which to use.

Proposition 8. *Let P be an optimization problem in the variables $x_1, \ldots, x_L$ which have finite numeric domains, with objective function $\min \sum_{i=1}^{L} c_i x_i$. Let M be an MDD for P with shortest path length ℓ, and let A_i be the set of arcs in M at level i. For each arc a in M, let v_a be the value associated with a, w_a be the cost associated with a, and $\overline{w}_a$ be the residual cost of a for the shortest path through M. Then taking $\overline{c}_i = \min_{a \in A_i} \frac{\overline{w}_a}{v_a}$ (where $\frac{\overline{w}_a}{0}$ is defined to be some sufficiently large value), we have that for any feasible solution $\{x_i\}$ to P,*

$$\sum_{i=1}^{L} c_i x_i \geq \ell + \sum_{i=1}^{L} \overline{c}_i x_i$$

Proof. Let $\{x_i\}$ be any feasible solution to P. Then there is some path in M, $(s = n_0, n_1, \ldots, n_{L-1}, t)$ such that the value associated with a_i is $v_{a_i} = x_i$ for all i in the path. Then

$$\begin{aligned} \ell + \sum_{i=1}^{L} \overline{c}_i x_i &\leq \ell + \sum_{i=1}^{L} \frac{\overline{w}_{a_i}}{v_{a_i}} x_i \\ &= \ell + \sum_{i=1}^{L} \overline{w}_{a_i} \\ &= \sum_{i=1}^{L} w_a \\ &= \sum_{i=1}^{L} c_i v_a \\ &= \sum_{i=1}^{L} c_i x_i \end{aligned}$$

□

When choosing values for the node potentials, it is desirable to enforce that on all arcs $a = (n, n')$, $w_a - (u_{n'} - u_n) \geq 0$ so that all residual costs $\overline{w_a}$ are non-negative. This ensures that all $\overline{c}_i \geq 0$, which ensures that later additive bounding algorithms will output non-negative objective values. Taking u_n to be the length of the shortest path from the source to n is one way to achieve this.

Example 5. Consider the minimum vertex problem on the graph in Figure 3.1. In Figure 3.6 we present an exact MDD for this problem. We have labeled each node n with the shortest path from the source to n, u_n, and each arc $a = (n, n')$ with its residual cost $\overline{w}_a = w_a - (u_{n'} - u_n)$.

The shortest path through the MDD has length 2, corresponding to the minimum vertex cover $\{A, E\}$. Taking a minimum as in Proposition 8, we get

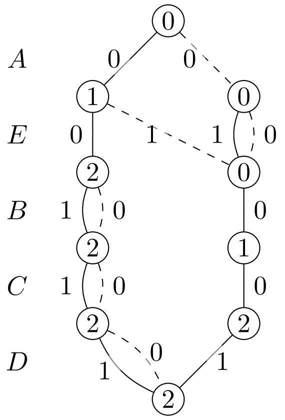


Figure 3.6: Exact MDD for the vertex cover problem with the shortest path labeled at each node, and the residual cost labeled at each arc.

$c_A = c_E = c_B = c_C = 0$, but $c_D = 1$. This means that any cover has size at least $2 + x_D$ (where x_D is the 0/1 variable representing whether D is in the vertex cover).

3.2.1 Sequential Ordering Problem

The sequential ordering problem is as follows: Given a set of jobs, $\mathcal{J}$ to be executed in series, a cost matrix $(c_{j,j'})_{j,j'\in\mathcal{J}}$, and a set of precedence constraints of the form $j \ll j'$ requiring that job j must be executed before job j', find a permutation $\sigma : \{1,\dots,L\} \to \mathcal{J}$ where $L = |\mathcal{J}|$, such that for each precedence constraint $j \ll j'$ we have $\sigma^{-1}(j) < \sigma^{-1}(j')$ and so that $\sum c_{\sigma(i),\sigma(i+1)}$ is minimized.

Since the sequential ordering problem is a generalization of the traveling salesperson problem, it is NP-hard.

We express the problem as a constraint program in the variables y_i for $i = 1,\dots,n$ where the domain of y_i is $\mathcal{J}$ and $y_i = j$ means that j is the ith job executed.

$$\begin{array}{lc}
\min & \sum c_{y_i,y_{i+1}} \\
s.t. & y_i \in \mathcal{J} \\
 & \texttt{AllDifferent}(\{y_i\}) \\
 & (y_i = j \texttt{ and } y_{i'} = j') \implies i < i' \qquad \text{for} j \ll j'
\end{array}$$

For the sequential ordering problem, we cannot compute the shortest path and residual costs as before, since it is pairs of variables (y_i, y_{i+1}) that determine the objective function. To deal with this issue we will alter the theoretical

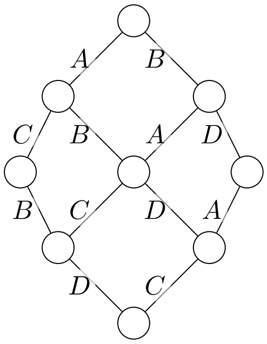


Figure 3.7: An exact MDD for the sequential ordering problem with $\mathcal{J} = \{A, B, C, D\}$ and constraints $A \ll C$ and $B \ll D$. The variables are y_i=[The ith job executed].

construction of the MDD. (In practice we will see that we can do the same computations on the MDD defined earlier.)

We construct an expanded MDD M' from M as follows. For each node n and each arc a coming into n, we create a node (n, a) in M'. For each arc a in M going from n' to n, we create an arc in M' from (n', a') to (n, a) for each a' entering n'. We also create a new sink t' and edges from all nodes (t, a) to t' where t is the sink in M.

Example 6. Consider the sequential ordering problem with $\mathcal{J} = \{A, B, C, D\}$, and the precedence constraints $A \ll C$ and $B \ll D$. Then the exact MDD is as in Figure 3.7 and the expanded MDD is as in Figure 3.8.

In the expanded MDD the variables are y_i with a domain of pairs of jobs, and the meaning $y_i = (j, j')$ means that we execute job j at time $i - 1$ and job j' at time i.

We compute the shortest path and residual costs on the arcs as before. Since our costs are associated with pairs of jobs j, j', we should like to have our variables be $x_{jj'}$ which is 1 if we go from job j to job j'. Our arcs represent "we go from job j to job j' at time i." So in order to get a single residual cost, let $A_{jj'}$ be the set of arcs in M' that represent going from job j to job j' and take

$$\overline{c}_{jj'} := \min_{A_{jj'}} \overline{w}_a. \tag{3.3}$$

Although the variables we started out with have non-numeric values, (the domain was $\mathcal{J}$) by using the expanded MDD we changed the variables to be Booleans, and so we are able to extract residual costs.

Proposition 9. *If σ is a feasible ordering of the jobs, and ℓ is the length of the shortest path through the expanded MDD, then*

$$\sum_{i=1}^{L-1} c_{\sigma(i)\sigma(i+1)} \geq \ell + \sum_{i=1}^{L-1} \overline{c}_{\sigma(i)\sigma(i+1)} \tag{3.4}$$

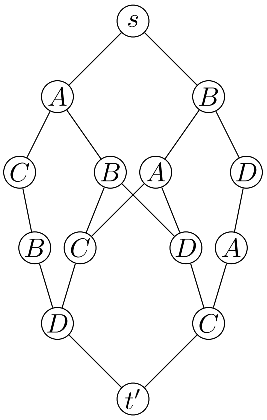


Figure 3.8: The expanded MDD for the MDD in Figure 3.7.

Proof. There is a path, $p = (s, n_1, \ldots, n_{L-1}, t)$ through M' with length ℓ so that the value associated with $a_i := (n_{i-1}, n_i)$ is $(\sigma(i-1), \sigma(i))$. So the length of the path is

$$\sum_{i=1}^{L-1} c_{\sigma(i)\sigma(i+1)} = \ell + \sum_{i=1}^{L-1} \overline{w}_a \geq \ell + \sum_{i=1}^{L-1} \overline{c}_{\sigma(i)\sigma(i+1)}$$

□

Note that the nodes in the expanded MDD M' other than the source and sink are in one-to-one correspondence with the edges in M, and that we have no data associated with the arcs of M'. This means that in implementation, we can use M instead of M' and do all the computations on the arcs. This is advantageous because M uses less memory.

3.2.2 Other Methods of Obtaining Potentials

Note that following equation (3.3), if any arc a associated with value (j, j') has $\overline{w}_a = 0$, then the residual cost $\overline{c}_{jj'}$ will be 0. If this happens for too many edges, it becomes very likely that a tour of cost 0 will be formed in the residual cost graph, and so further additive bounding algorithms will produce a lower bound of 0. We found this to be the case when taking u_n to be the length of the shortest path from s to n, so we would like to find a better way of generating u_n.

Since we require $w_{nn'} \geq u'_n - u_n$, and since $u_s = 0$ and $u_t = \ell$ where ℓ is the length of the shortest path from s to t, every arc on a shortest path will have $w_{nn'} = u_{n'} - u_n$, so $\overline{w}_{nn'} = 0$.

It seems likely that we can fix this issue using a different method of choosing u_n. If (n, n') has value (j, j'), then $\overline{w}_{mm'}$ for any other arc with value (j, j') will be ignored. It seems reasonable then to move the residual cost off that arc onto the one above by decreasing u_m as much as possible without making $\overline{w}_{mm''}$ go negative for any node m''. Similar cost modifications were proposed in [28].

If m is a node with only one incoming and one outgoing edge, then following the procedure above will result in the incoming edge having a higher residual cost than the original cost. While this is legal, it seems likely that there are other arcs with the same value that do not have this property, so we will move the residual cost upward in that situation as well.

Algorithm 7 starts from the bottom of the MDD and works upward to achieve this.

Algorithm 7 An algorithm that produces node potentials u_n

```
for all nodes $n \in M$ do
    Initialize $u_n$ to the length of the shortest path from $s$ to $n$
end for
Let $B = \{v_a : \text{a is on a shortest path through } M\}$
for $i = L, \ldots, 1$ do
    for $a \in A_i$ do
        if $v_a \in B$ or $a$ has only one incoming and one outgoing edge then
            Let $n$ be the start of $a$
            $u_n := \min_{(n,n') \in A_{i+1}} (w_{nn'} - u'_n)$
        end if
    end for
end for
```

This effectively means that if a has value (j, j') and is part of some shortest path, then we make $\overline{w}_a$ as small as possible since we already know that $\min_{a' : v_{a'} = (j,j')} \overline{w}_{a'} s = 0$, and move the residual cost somewhere else.

Example 7. Consider the sequential ordering problem on $\mathcal{J} = \{A, B, C, D\}$ with the precedence constraints $A \ll C$ and $B \ll D$ and the symmetric setup times as in Figure 3.9. We show artificial start and end nodes as well whose incident edges have cost 0. Note that the shortest path is (A, C, B, D) with total setup time 70.

A relaxed MDD for this problem is in Figure 3.10, and the corresponding expanded MDD is in Figure 3.11.

Using this MDD and these values for the potentials, we get a lower bound of 10, and residual costs as pictured in Figure 3.12. Note that this graph has a 0-cost path, (B, A, C, D), so any other algorithm that takes these residual costs as input will output 0 as the lower bound.

However, if we apply the algorithm to move the residual costs, we get the MDD in Figure 3.13, in which the edge from A to C now has residual cost 50. In this graph, the shortest path is still (B, A, C, D), but this path now has a cost of 50, giving us a lower bound of $10 + 50 = 60$.

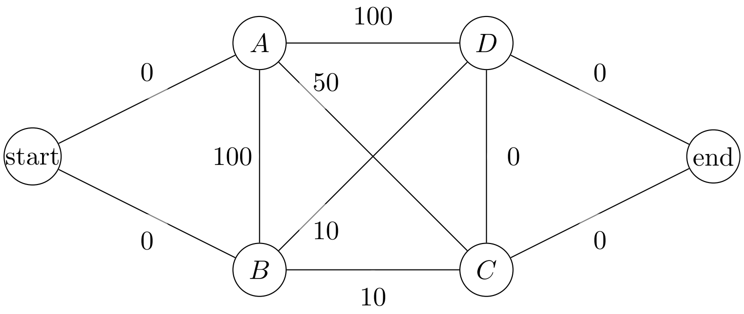


Figure 3.9: Setup times for an instance of the sequential ordering problem.

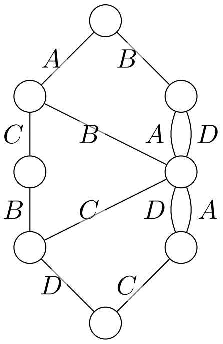


Figure 3.10: A reduced MDD for the sequential ordering problem in Figure 3.9.

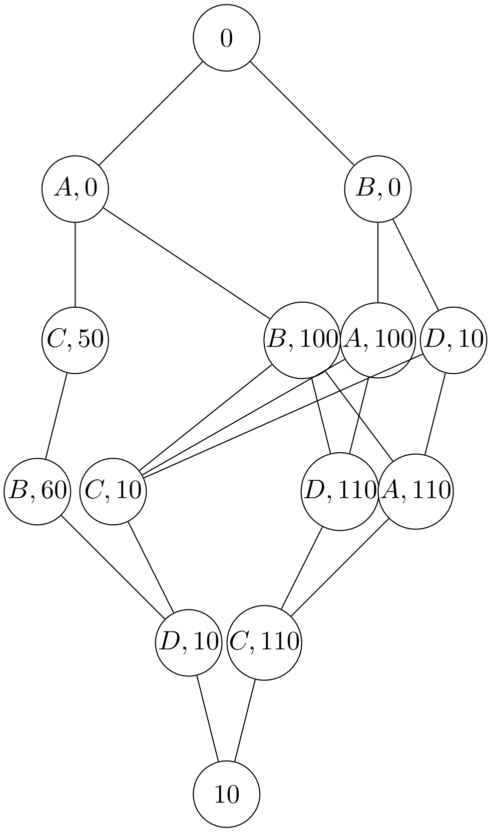


Figure 3.11: The expanded MDD of Figure 3.10. Each node is labeled with the length of the shortest path from the source to that node.

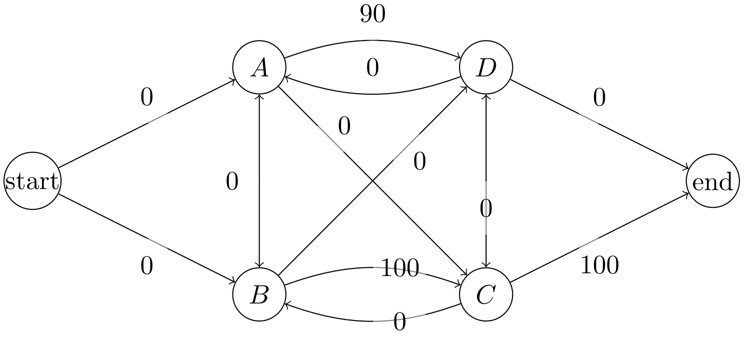


Figure 3.12: Residual costs when using u_n as the length of the shortest path to n.

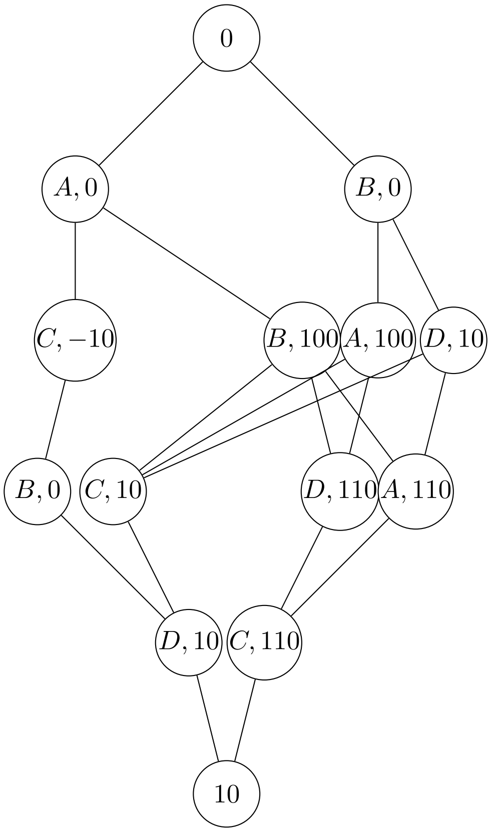


Figure 3.13: The expanded MDD of Figure 3.10. Each node is labeled with the u_n as output by Algorithm 7.

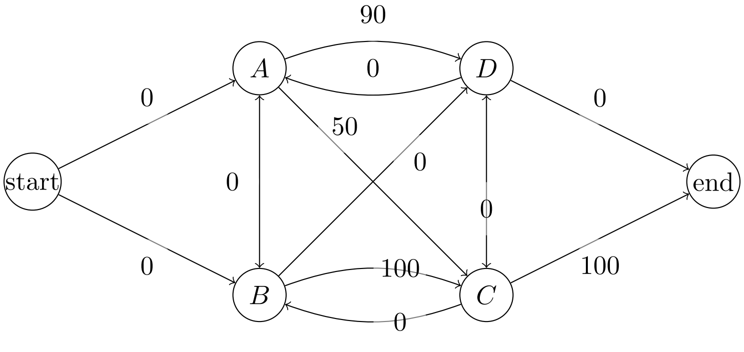


Figure 3.14: Residual costs when using u_n as the output of Algorithm 7.

3.3 Experiments

3.3.1 Other Additive Bounding Algorithms Used

For our experiments, we used three algorithms for solving relaxations: the Hungarian algorithm for solving the assignment problem (AP), the Held-Karp algorithm (HK), and the shortest path MDD problem described in Section 3.2 (MDD).

The code we use for the Hungarian Algorithm comes from `http://www.assignmentproblems.com/APC_APS.htm`. The code for the Held-Karp algorithm was written following the algorithm in [44]. The code for the MDD algorithm comes from Andre Cire, and follows the algorithm in [13]. We restricted the width of the MDD to 512.

The assignment problem is the unimodular LP in variables $x_{jj'}$

$$\begin{array}{ll} \min & c^T \cdot x \\ s.t. & \sum\limits_{j \in \mathcal{J}} x_{jj'} = 1 \quad \forall j \in \mathcal{J} \\ & \sum\limits_{j' \in \mathcal{J}} x_{jj'} = 1 \quad \forall j \in \mathcal{J} \\ & 0 \leq x_{jj'} \leq 1 \quad \forall j, j' \in \mathcal{J} \end{array} \tag{3.5}$$

which outputs a subgraph in which each vertex has one incoming edge and one outgoing edge, but may not be connected.

For the Held-Karp algorithm we work on a graph $G = (V, E)$ with a special vertex, $v_0 \in V$. We define a v_0-tree as a spanning tree on $V \setminus \{v_0\}$ together with two edges from v_0. In other words, it is a connected graph whose one and only cycle includes v_0. Note that a Hamiltonian cycle is a special case of a v_0-tree. The Held-Karp algorithm iteratively finds a minimum spanning v_0-tree, and then for every vertex v that does not have in-degree and out-degree 1, increases or decreases the weight of the incoming and outgoing edges. The specifics are presented in Algorithm 8.

3.3.2 Data sets

We randomly generated sequential ordering problem instances on which to test our additive bounding scheme. For each of 20 data sets, we generated 100 points in the unit square, and computed the distance as $\lfloor 100 \times [\text{Euclidean distance}] \rfloor$. We then randomly selected $\rho\binom{100}{2}$ precedence constraints for constraint density $\rho \in \{0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18\}$, resulting in 9 instances per dataset, and 180 instances total.

For each instance we computed an upper bound using by running the MDD algorithm from [13] for 3 hours.

Algorithm 8 The Held-Karp algorithm for finding a lower bound on the sequential ordering problem.

```
while stepsize $> \epsilon$ do
    Let $T$ be the edges in an undirected minimum spanning tree on $V \setminus \{v_0\}$
    Let $e^+$ be the minimum incoming edge to $v_0$
    Let $e^-$ be the minimum outgoing edge to $v_0$
    $T := T \cup \{e^+, e^-\}$
    for all $v \in V$ do
        $w_{vu}$ += stepsize($\deg_T(v) - 2$)$\forall u \in V$
        $w_{uv}$ += stepsize($\deg_T(v) - 2$)$\forall u \in V$
    end for
    Adjust stepsize
end while

Let $T$ be the set of edges in an undirected minimum spanning tree on $V \setminus \{v_0\}$
for all $u, v \in V \setminus \{v_0\}$ do
    if $(u, v) \in T$ then
        $\overline{w}_{uv} := 0$
    else
        Let $C$ be the unique cycle consisting of $(u, v)$ and edges from $T$
        Let $(x, y)$ be the highest weight edge in that cycle
        $\overline{w}_{uv} := w_{uv} - w_{xy}$
    end if
end for
Let $e^+$ be the minimum incoming edge to $v_0$
Let $e^-$ be the minimum outgoing edge to $v_0$
for $v \in V \setminus \{v_0\}$ do
    $\overline{w}_{v,v_0} := \overline{w}_{v,v_0} - \overline{w}_{e^+}$
    $\overline{w}_{v_0,v} := \overline{w}_{v_0,v} - \overline{w}_{e^-}$
end for
```

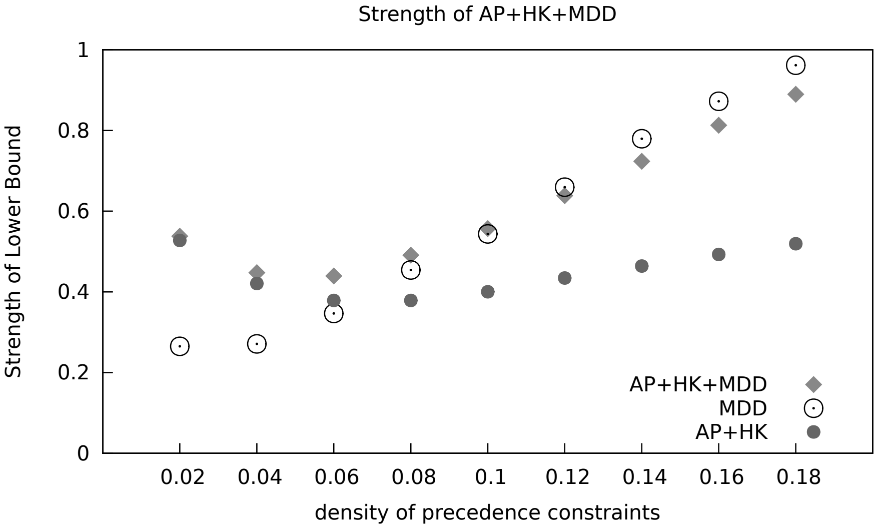


Figure 3.15: Results of running AP, then HK, then MDD.

3.3.3 Results

We ran all 180 instances through all 6 permutations of {AP, HK, MDD}. For each one, we compute the strength of the lower bound as $\frac{\text{lowerbound}}{\text{upperbound}}$. We present the average strength of the lower bound for each value of ρ in Figures 3.15-3.20. To explain our naming scheme through an example, the graph labeled "AP+HK+MDD" shows the results of running the AP, then HK, then MDD.

In each of these figures, we look at the strength of the combination of three versus the MDD alone and versus the AP and HK together.

We can see from these that the MDD performs better when there are more precedence constraints. This is because when there are more constraints, the exact MDD is smaller and so the relaxed MDD that we store in memory is more similar to the exact MDD.

We can also see that AP and HK together (in either order) perform better when there are fewer precedence constraints. This is because these algorithms ignore the precedence constraints, so in instances with fewer of them, these algorithms are using more of the instance data.

We can see in Figures 3.15, 3.16, 3.18, and 3.19 that when the constraint density is around 0.06-0.08, the combination of all three performs the best.

In figures 3.17 and 3.20, we see that running the assignment problem or Held-Karp with respect to the MDD residual costs had no affect on the bound. This is because the residual costs from the MDD were largely zero. In Figure 3.21 we show the average percentage of the edges that had residual cost 0 when we use the shortest path as the potentials and when we use Algorithm 7 for the potentials. We can see that in either case, the fraction is significant (20%-60%),

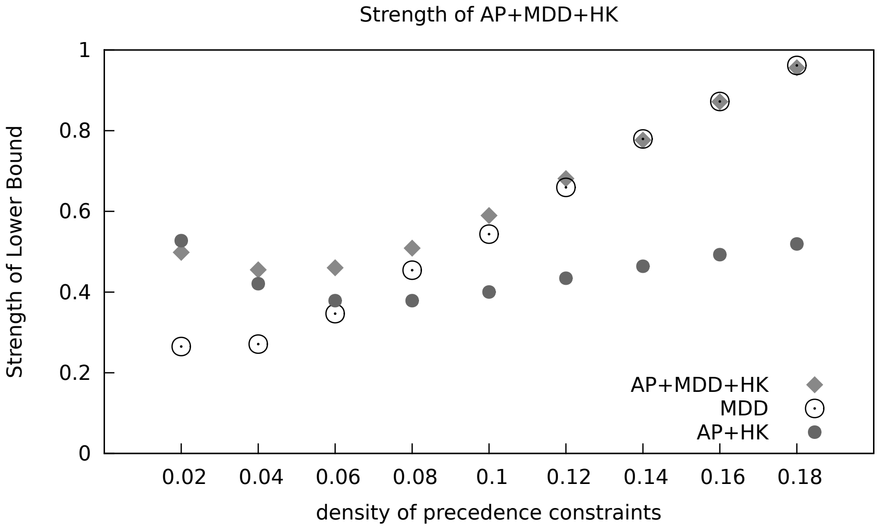


Figure 3.16: Results of running AP, then MDD, then HK.

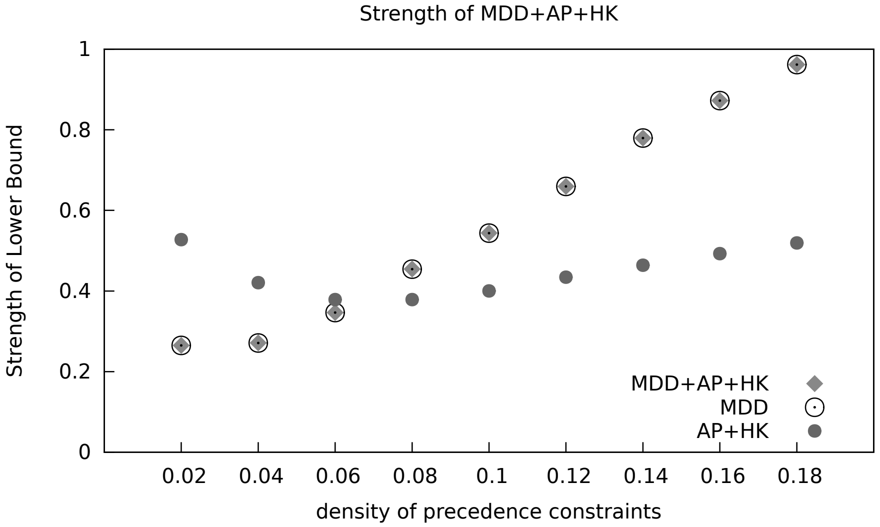


Figure 3.17: Results of running MDD, then AP, then HK.

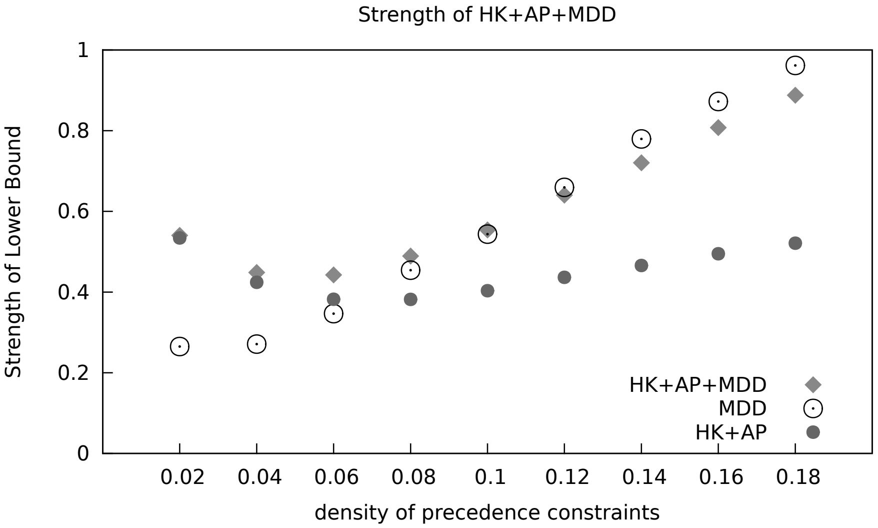


Figure 3.18: Results of running HK, then AP, then MDD.

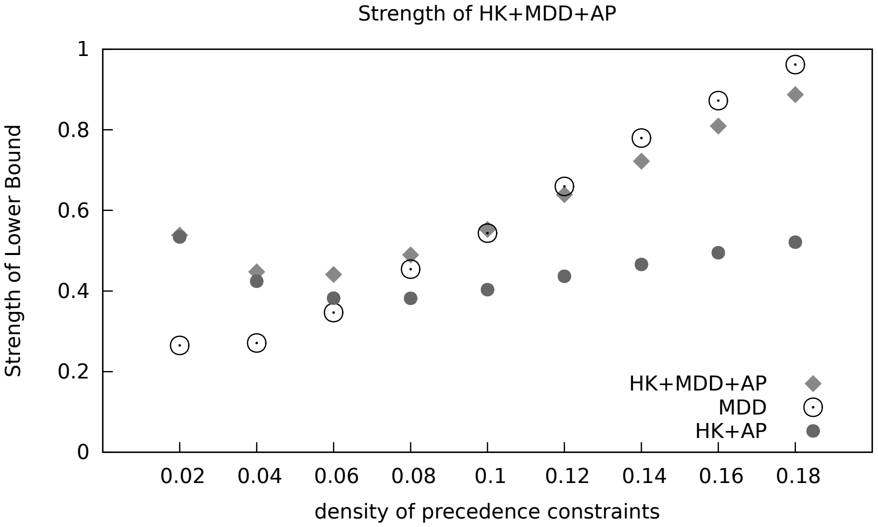


Figure 3.19: Results of running HK, then MDD, then AP.

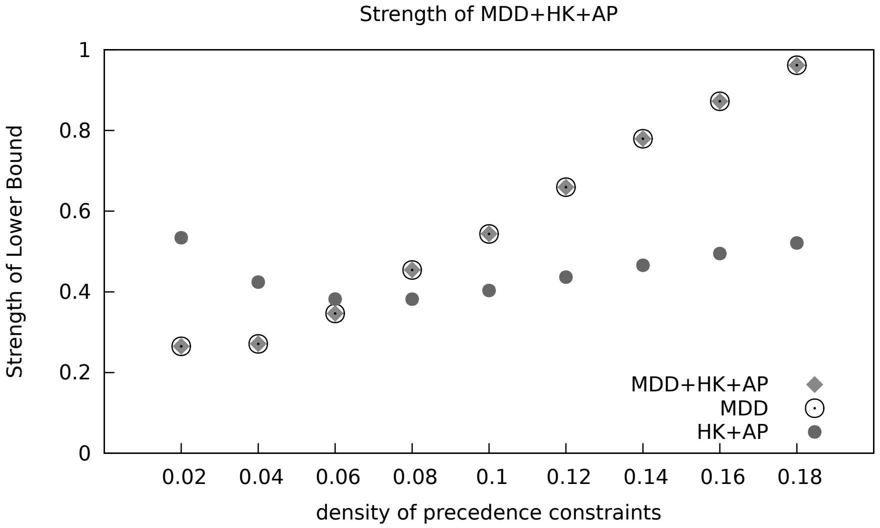


Figure 3.20: Results of running MDD, then HK, then AP.

which is why we cannot improve our bounds though additive bounding.

In Figure 3.22 we show the reduction in the fraction of the edges with 0 residual cost that algorithm 8 gets us. We can see that although it does reduce the number of zeros for $\rho \leq 0.14$, it only reduces it by a maximum of 4%, and increases it for $\rho \geq 0.16$.

3.4 Conclusion

We have shown a method of incorporating MDDs into an additive bounding scheme. We can see that these methods can outperform pre-existing methods when the MDD is the final step of the additive bounding process. It is possible that through a better assignment of potentials to the edges, the MDD could output a residual cost matrix that has fewer zeros and could then be used anywhere in the additive bounding process.

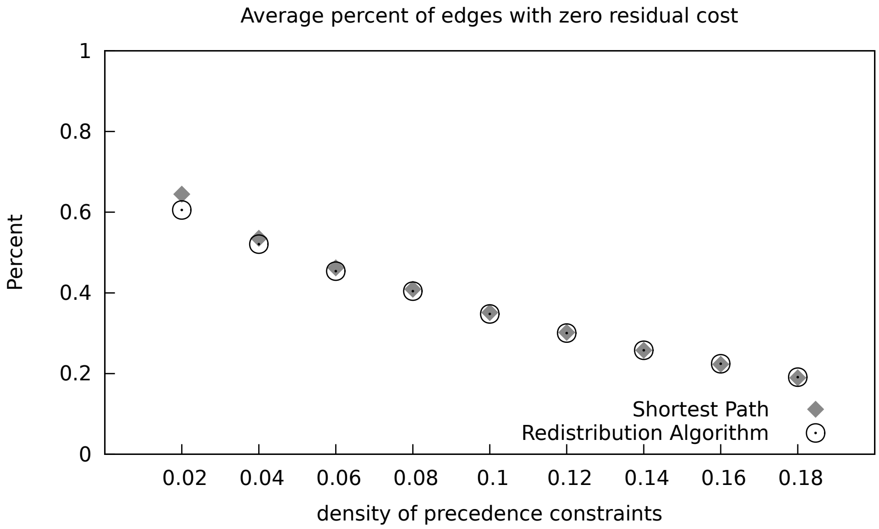


Figure 3.21: The fraction of 0 residual costs output by the MDD algorithm.

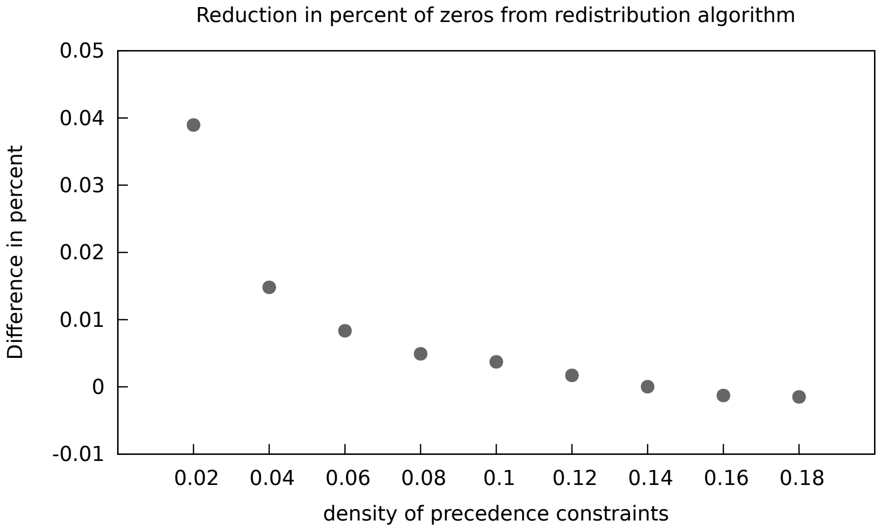


Figure 3.22: The reduction in the fraction of 0 residual costs when we redistribute the residual costs.

Chapter 4

Conclusion

In Chapter 2, we presented a new way to approximate the LP bound for Golomb rulers. We demonstrated its relationship to existing methods, and showed that we can compute the LP bound much faster using combinatorial methods. We then used this fast computation in a search procedure, demonstrating that we can use this bound to reduce the size of the search tree and, in cases where the LP bound is strong enough, reduce the search time as well.

In Section 2.6, we showed how bounding ruler segments can further reduce the search time and search tree size. We only considered bounding the segments $d_{i,n-i+1}$ and sums of such segments. It is left open whether bounding other segments or linear combinations of segments can improve the search even further, and what heuristics should be used for this.

We saw that although we can approximate the LP bounds very quickly, the LP bound itself is somewhat weak. To push this idea further, we would need to develop stronger lower bounds.

The Golomb ruler problem is closely related to the graceful graph problem. A graph is graceful if all vertices v, can be assigned labels x_v so that the $|x_v - x_{v'}|$ are distinct for all edges (v, v'). There is no known way to efficiently test whether a given graph is graceful. In particular, it is an open conjecture whether all trees are graceful. It may be that applying a Lagrangian relaxation is helpful for problems such as this.

In Chapter 3, we showed a method of incorporating MDDs into an additive bounding scheme. We showed these methods can outperform pre-existing methods when the MDD is the final step of the additive bounding process.

Future research in this area could focus on finding a better assignment of potentials to the edges, so that the MDD would output a residual cost matrix that has fewer zeros. This way the MDD could then be used anywhere in the additive bounding process, instead of strictly at the end.

It is left open whether these methods can be applied to other problems, such as other variations of the traveling salesperson problem, set packing and set covering problems, or bin-packing problems.

Appendix A

Search Procedure for Ruler Segment Bound

Let $\ell_{a/(a+b)} := \left\lceil \frac{\lceil z(a,b)^* \rceil - bL}{a} \right\rceil$ as in equation (2.12), so that $d_{ij} \geq \ell_{a/(a+b)}$ for all $a, b \in \mathbb{N}$.

We apply a procedure inspired by binary search to find values for a and b, as depicted in Algorithm 9. The procedure is called with 'FindBestBound(1, 5, 6)' to search the interval $\frac{a}{a+b} \in [0, 1]$.

Algorithm 9 Search procedure to find appropriate values for a and b that determine the bound on the ruler segment.

```
procedure FindBestBound(numerator-start, numerator-end, denominator)
    for $i$ = numerator-start .. numerator-end do
        calculate $\ell_{\frac{i}{denominator}}$
    end for
    Let $M := \left\{ k : \ell_{\frac{k}{denominator}} = \max_i \ell_{\frac{i}{denominator}} \right\}$.
    if $|M| = 1$ then
        Let $k$ be such that $M = \{k\}$
        return FindBestBound($3k - 2, 3k + 2, 3 \cdot denominator$)
    else if $|M| = 2$ then
        Let $k$ be such that $M = \{k, k + 1\}$
        return FindBestBound($6k + 1, 6k + 5, 6 \cdot denominator$)
    else $|M| \geq 3$, $k \in M$
        return $\ell_{\frac{k}{denominator}}$
    end if
end procedure
```

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Inc., 1993.

[2] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, 1978.

[3] H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A constraint store based on multivalued decision diagrams. In *Principles and Practice of Constraint Programming–CP 2007*, pages 118–132. Springer, 2007.

[4] M. Behle. *Binary Decision Diagrams and Integer Programming.* PhD thesis, Universität des Saarlandes, 2007.

[5] P. Benchimol, W.-J. van Hoeve, J.-C. Régin, L.-M. Rousseau, and M. Rueher. Improved filtering for weighted circuit constraints. *Constraints*, 17(3): 205–233, 2012.

[6] D. Bergman, A. A. Cire, W.-J. van Hoeve, and T. Yunes. BDD-based heuristics for binary optimization. *Journal of Heuristics*, 20(2):211–234, 2014.

[7] D. Bergman, A. A. Cire, and W.-J. van Hoeve. MDD propagation for sequence constraints. *Journal of Artificial Intelligence Research*, 50:697–722, 2014.

[8] D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. Optimization bounds from binary decision diagrams. *INFORMS Journal on Computing*, 26(2):253–268, 2014.

[9] G. S. Bloom and S. W. Golomb. Applications of numbered undirected graphs. *Proceedings of the IEEE*, 65(4):562–570, 1977.

[10] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug 1986.

[11] H. Cambazard, E. O'Mahony, and B. O'Sullivan. Hybrid methods for the multileaf collimator sequencing problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 56–70. Springer, 2010.

[12] G. Carpaneto, S. Martello, and P. Toth. Algorithms and codes for the assignment problem. *Annals of Operations Research*, 13:191–223, 1988.

[13] A. A. Cire and W.-J. van Hoeve. Multivalued decision diagrams for sequencing problems. *Operations Research*, 61(6):1411–1428, 2013.

[14] I. Dotú and P. Van Hentenryck. A simple hybrid evolutionary algorithm for finding Golomb rulers. In *The IEEE Congress on Evolutionary Computation*, pages 2018–2023. IEEE, 2005.

[15] K. Drakakis, R. Gow, and L. O'Carroll. On some properties of costas arrays generated via finite fields. In *2006 40th Annual Conference on Information Sciences and Systems*, pages 801–805. IEEE, 2006.

[16] M. Fischetti and P. Toth. An additive bounding procedure for combinatorial optimization problems. *Operations Research*, 37(2):319–328, 1989.

[17] R. Gagliardi, J. Robbins, and H. Taylor. Acquisition sequences in PPM communications. *IEEE Transactions on Information Theory*, IT-33(5): 738–744, 1987.

[18] P. Galinier, B. Jaumard, R. Morales, and G. Pesant. A constraint-based approach to the Golomb ruler problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2001. A more recent version (June 11, 2007) can be downloaded from `http://www.crt.umontreal.ca/~quosseca/pdf/41-golomb.pdf`.

[19] C. Gervet. Constraints over structured domains. *Handbook of Constraint Programming*, pages 605–638, 2006.

[20] T. Hadzic and J. N. Hooker. Postoptimality analysis for integer programming using binary decision diagrams. Technical report, Carnegie Mellon University, 2008.

[21] T. Hadzic, J. N. Hooker, B. O'Sullivan, and P. Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In *Principles and Practice of Constraint Programming*, volume 5202 of *Lecture Notes in Computer Science*, pages 448–462. Springer Berlin Heidelberg, 2008.

[22] P. Hansen, B. Jaumard, and C. Meyer. On lower bounds for numbered complete graphs. *Discrete Applied Mathematics*, 94(13):205 – 225, 1999.

[23] M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[24] M. Held, P. Wolfe, and H. P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–88, 1974.

[25] S. Hoda, W.-J. van Hoeve, and J. N. Hooker. A systematic approach to MDD-based constraint programming. In *Principles and Practice of Constraint Programming CP 2010*, volume 6308 of *Lecture Notes in Computer Science*, pages 266–280. Springer Berlin Heidelberg, 2010.

[26] W.-J. van Hoeve and I. Katriel. Global constraints. *Handbook of constraint programming*, pages 169–208, 2006.

[27] J. N. Hooker. *Integrated Methods for Optimization.* Springer, 2007.

[28] J. N. Hooker. Decision diagrams and dynamic programming. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 94–110. Springer Berlin Heidelberg, 2013.

[29] B. Kell and W.-J. van Hoeve. An MDD approach to multidimensional bin-packing. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 128–143. Springer Berlin Heidelberg, 2013.

[30] B. Kell, A. Sabharwal, and W.-J. van Hoeve. BDD-guided clause generation. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 215–230. Springer International Publishing, 2015.

[31] J. Kinable. *Decomposition Approaches for Optimization Problems.* PhD thesis, KU Leuven, 2014.

[32] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[33] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38(4):985–999, 1959.

[34] C. Lemaréchal. Lagrangian relaxation. In *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions [Based on a Spring School]*, pages 112–156, London, UK, UK, 2001. Springer-Verlag.

[35] R. Lorentzen and R. Nilsen. Application of linear programming to the optimal difference triangle set problem. *IEEE Trans. Inf. Theor.*, 37(5): 1486–1488, 2006.

[36] J. Menana and S. Demassey. Sequencing and counting with the multicost-regular constraint. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 178–192. Springer, 2009.

[37] C. Meyer and B. Jaumard. Equivalence of some LP-based lower bounds for the Golomb ruler problem. *Discrete Appl. Math.*, 154(1):120–144, 2006.

[38] A. T. Moffet. Minimum-redundancy linear arrays. *IEEE Transactions on Anntennas and Propagation*, AP-16(2):172–175, 1968.

[39] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, 1988.

[40] S. Prestwich. Trading completeness for scalability: Hybrid search for cliques and rulers. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2001.

[41] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of AAAI*, pages 362–367. AAAI Press, 1994.

[42] J. P. Robinson and A. J. Bernstein. A class of binary recurrent codes with limited error propagation. *IEEE Transactions on Information Theory*, IT-13(1):106–113, 1967.

[43] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1986.

[44] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[45] M. Sellmann. Theoretical foundations of CP-based Lagrangian relaxation. In *Proceedings of CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 634–647. Springer, 2004.

[46] M. Sellmann and T. Fahle. Constraint programming based Lagrangian relaxation for the automatic recording problem. *Annals of Operations Research*, 118(1–4):17–33, 2003.

[47] C. E. Shannon. A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers*, 57(12):713–723, 1938.

[48] J. B. Shearer. Improved LP lower bounds for difference triangle sets. *Journal of Combinatorics*, 6, 1999.

[49] J. Singer. A theorem in finite projective geometry and some applications to number theory. *Transactions of the American Mathematical Society*, 43 (3):377–385, 1938.

[50] M. R. Slusky and W.-J. van Hoeve. A Lagrangian relaxation for Golomb rulers. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 251–267. Springer, 2013.

[51] B. M. Smith, K. Stergiou, and T. Walsh. Modelling the Golomb ruler problem. In *IJCAI Workshop on Non-binary Constraints*, 1999.

[52] S. W. Soliday, A. Homaifar, and G. L. Lebby. Genetic algorithm approach to the search for Golomb rulers. In *6th International Conference on Genetic Algorithms (ICGA95*, pages 528–535. Morgan Kaufmann, 1995.

[53] A. Srinivasan, T. Ham, S. Malik, and R. K. Brayton. Algorithms for discrete function manipulation. In *Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on*, pages 92–95. IEEE, 1990.

[54] I. Wegener. *Branching programs and binary decision diagrams: theory and applications*, volume 4. SIAM, 2000.