UNIVERSIDADE FEDERAL DO CEARÁ

DEPARTAMENTO DE ESTATÍSTICA E MATEMÁTICA APLICADA

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM E MÉTODOS QUANTITATIVOS

LUCAS BRAGA DE ALBUQUERQUE

FINDING MAXIMUM PATTERNS USING DECISION DIAGRAMS

FORTALEZA

2020

LUCAS BRAGA DE ALBUQUERQUE

FINDING MAXIMUM PATTERNS USING DECISION DIAGRAMS

Dissertação apresentada ao Programa de Pós-Graduação em Modelagem e Métodos Quantitativos, da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Modelagem e Métodos Quantitativos. Área de Concentração: Modelagem e Métodos Quantitativos.

Orientador: Prof. Dr. Tibérius de Oliveira e Bonates

FORTALEZA

2020

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A311f Albuquerque, Lucas Braga de.
Finding maximum patterns using decision diagrams / Lucas Braga de Albuquerque. – 2020.
71 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Modelagem e Métodos Quantitativos, Fortaleza, 2020.
Orientação: Prof. Dr. Tibérius de Oliveira e Bonates.

1. Maximum pattern. 2. Logical analysis of data. 3. Decision diagram. 4. Branch-and-bound. I. Título.
CDD 510

LUCAS BRAGA DE ALBUQUERQUE

FINDING MAXIMUM PATTERNS USING DECISION DIAGRAMS

Dissertação apresentada ao Programa de Pós-Graduação em Modelagem e Métodos Quantitativos, da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Modelagem e Métodos Quantitativos. Área de Concentração: Modelagem e Métodos Quantitativos.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Tibérius de Oliveira e Bonates (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Ronan Pardo Soares
Universidade Federal do Ceará (UFC)

Prof. Dr. Carlos Diego Rodrigues
Universidade Federal do Ceará (UFC)

Prof. Dr. Javier Marenco
Universidad Nacional de General Sarmiento (UNGS)
Universidad de Buenos Aires (UBA)

ABSTRACT

Logical Analysis of Data (LAD) is a rule-based algorithm for supervised classification that is based on optimization, combinatorics, and Boolean functions. A central concept in LAD is that of a pattern, which summarizes knowledge extracted from a given dataset. Let D be a set of binary vectors partitioned into a set of positive and a set of negative observations. A positive pattern is a subcube of the n-dimensional hypercube having a nonempty intersection with the positive part of D, and an empty intersection with the negative part of D. An observation is covered by a pattern if it belongs to the corresponding subcube, and the coverage of a pattern is the number of observations in D covered by it. The maximum positive a-pattern problem consists in finding a positive pattern whose coverage is maximum among those that cover a given positive observation a in D, which amounts to solving a nonlinear set covering problem. A generalization of it, the maximum positive pattern problem, asks for a positive pattern of maximum coverage among all patterns, not only among those covering a particular observation. We review all integer linear programming (ILP) approaches from the literature for these two problems and empirically evaluate them using a commercial ILP software. Furthermore, we introduce a dynamic programming model, a merging rule, and all necessary heuristics in order to model and solve the two problems using a recently-developed optimization methodology based on decision diagrams (DDs). The methodology consists of a branch-and-bound (BAB) algorithm, in which DDs play the traditional role of the linear programming relaxation, as well as that of primal heuristics. We also discuss relevant implementation details in order to enhance the performance of the DD-based BAB. Lastly, we compare the performance of our DD-based solver with that of the ILP approaches from the literature. Our results indicate that a straightforward DD-based branch-and-bound implementation typically produces higher quality solutions than a commercial MILP software within a common time limit.

Keywords: Maximum pattern. Logical analysis of data. Decision diagram. Branch-and-bound.

RESUMO

Análise Lógica de Dados (ALD) é um algoritmo de classificação supervisionada baseado em regras, o qual é fundamentado em otimização, combinatória, e funções Booleanas. Um conceito central em ALD é o de padrão, o qual resume informação extraída de um dado conjunto de dados. Seja D um conjunto de vetores binários particionado em um conjunto de observações positivas e um conjunto de observações negativas. Um padrão positivo é um subcubo do hipercubo n-dimensional, o qual possui uma interseção não-vazia com a parte positiva de D, e uma interseção vazia com a parte negativa de D. Uma observação é coberta por um padrão se pertence ao subcubo correspondente, e a cobertura de um padrão é o número de observações em D cobertas por ele. O problema do a-padrão positivo máximo consiste em encontrar um padrão cuja cobertura é máxima entre todos aqueles que cobrem uma dada observação positiva a em D, o que corresponde a resolver um problema de cobertura de conjuntos não-linear. Uma generalização do problema, o problema do padrão positivo máximo, busca um padrão positivo cuja cobertura é máxima entre todos os padrões, não apenas aqueles que cobrem uma observação em particular. Revisamos todas as abordagens por programação linear inteira (PLI) para esses dois problemas encontradas na literatura e as avaliamos empiricamente usando um software comercial de PLI. Além disso, introduzimos um modelo de programação dinâmica, uma regra de mescla, e todas as heurísticas necessárias para modelar e resolver os dois problemas utilizando-se de uma metodologia de otimização baseada em diagramas de decisão (DDs), a qual foi desenvolvida recentemente. A metodologia consiste em um algoritmo de *branch-and-bound* (BAB), no qual DDs fazem o papel tradicional da relaxação linear, assim como o de heurísticas primais. Também discutimos detalhes de implementação relevantes com o intuito de melhorar a performance do BAB baseado em DDs. Por fim, comparamos a performance do nosso resolvedor baseado em DDs com as abordagens de PLI da literatura. Nossos resultados sugerem que uma implementação direta de um BAB baseado em DDs produz, em geral, soluções de mais qualidade do que um software comercial de PLI, dentro de um mesmo limite de tempo.

Palavras-chave: Padrão máximo. Análise lógica de dados. Diagrama de decisão. Branch-and-bound.

LIST OF FIGURES

Figure 1 – A BDD 28
Figure 2 – An exact DD 33
Figure 3 – A restricted DD 36
Figure 4 – A relaxed DD 39
Figure 5 – DDs for subproblems in a BAB 43
Figure 6 – A partially constructed exact DD for the MPP 48
Figure 7 – Construction of a relaxed DD for the MPP with maximum width 3 51
Figure 8 – A partially constructed exact DD for the MPP with different variable ordering 52
Figure 9 – DDs for subproblems in a BAB with modified costs 53
Figure 10 – Lower bound evolution for each instance 71

LIST OF TABLES

Table 1 – A binary dataset 14
Table 2 – Examples of patterns in the dataset given in Table 1 16
Table 3 – Pattern generation models 24
Table 4 – Dissimilarity matrix 52
Table 5 – Instances 57
Table 6 – Results for the ILP models using CPLEX 58
Table 7 – Results for the ILP models using CPLEX 58
Table 8 – Results for the ILP models using CPLEX 58
Table 9 – Results for the ILP models using CPLEX 59
Table 10 – Results for the ILP models using CPLEX 59
Table 11 – Results for the ILP models using CPLEX 59
Table 12 – Results for the ILP models using CPLEX 60
Table 13 – Results for the ILP models using CPLEX 60
Table 14 – Results for the ILP models using CPLEX 60
Table 15 – Comparison CPLEX vs. DD 62
Table 16 – DD results with and without the use of a variable ordering heuristic 63
Table 17 – Bounds comparison 70

LIST OF ALGORITHMS

Algorithm 1 – `compile_exact_DD` 32
Algorithm 2 – `compile_layer` 33
Algorithm 3 – `compile_restricted_DD` 35
Algorithm 4 – `compile_relaxed_DD` 37
Algorithm 5 – `node_select` 40
Algorithm 6 – `DD_based_BAB` 42

CONTENTS

1	**INTRODUCTION**	11
1.1	**Objectives**	13
1.2	**Organization**	13
2	**MAXIMUM PATTERNS**	14
2.1	**Definitions**	14
2.2	**The maximum α-pattern problem (α-MPP)**	16
2.2.1	***Complexity***	17
2.3	**The general case: the maximum pattern problem (MPP)**	18
2.4	**Optimal prime patterns: the maximum pattern of minimum degree problem (MPMDP)**	18
2.5	**Bonates *et al.* ILP model for the α-MPP**	19
2.6	**Ryoo and Jang MILP model for the MPP**	20
2.7	**Guo and Ryoo MILP model for the MPP**	21
2.7.1	***Adaptation to the MPMDP***	22
2.8	**Yan and Ryoo MILP models for the MPP**	23
2.8.1	***Valid inequalities***	24
3	**DECISION DIAGRAMS FOR OPTIMIZATION: AN OVERVIEW**	26
3.1	**Discrete optimization problems**	26
3.2	**Decision diagrams: definitions**	27
3.2.1	***Variable ordering***	29
3.3	**Dynamic programming formulations**	29
3.3.1	***Validity***	31
3.4	**Exact decision diagrams**	32
3.5	**Restricted decision diagrams**	34
3.6	**Relaxed decision diagrams and merging rules**	35
3.6.1	***Validity of relaxation operators***	38
3.7	**Node selection for restricted and relaxed DDs**	39
3.8	**A decision diagram-based branch-and-bound**	40
3.8.1	***Search node selection and types of exact cut sets***	43
4	**FINDING MAXIMUM PATTERNS USING DECISION DIAGRAMS**	45
4.1	**A coverage-based formulation for the MPP**	45

4.2 **Dynamic programming formulation for the MPP** 46
4.2.1 ***Validity*** 47
4.3 **Properties** 48
4.4 **Adaptation to the α-MPP** 49
4.5 **Merging rule** 50
4.6 **Node selection** 50
4.7 **Variable ordering** 51
4.8 **Improving bounds obtained from restricted DDs** 53
4.9 **Speeding up the compilation of relaxed DDs** 54
5 **COMPUTATIONAL RESULTS** 56
5.1 **Instances** 56
5.2 **Evaluation of the ILP models** 57
5.3 **Evaluation of the DD-based algorithm** 61
5.3.1 ***Evaluation of the variable ordering heuristic*** 63
6 **CONCLUSIONS AND FUTURE WORK** 64
6.1 **Contributions** 64
6.2 **Acknowledgements** 65
REFERENCES 66
APPENDIX A – COMPARISON OF LINEAR RELAXATION AND DD-BASED BOUNDS 70
APPENDIX B – LOWER BOUND EVOLUTION IN AN EXECUTION OF THE DD-BASED BRANCH-AND-BOUND 71

1 INTRODUCTION

Logical analysis of data (CRAMA *et al.*, 1988), or simply LAD, belongs to a class of supervised classification methods based on rules, such as rough set theory (PAWLAK, 1982) and decision trees (QUINLAN, 1986). LAD has found numerous applications in the industry, medicine, and finance (LEJEUNE *et al.*, 2019). An implementation of the LAD methodology was described by Boros *et al.* (2000). Rule-based classifiers can have some advantage over powerful methods in the field of machine learning such as support vector machines and neural networks. One advantage is that rules can be interpretable by humans and, thus, have the potential to provide cause-effect information hidden in the dataset. Another advantage is that algorithms for building classifiers based on rules may be more easily tuned in order to avoid overfitting than other methods.

A key step when building rule-based classifiers is generating good classification rules. These rules are known as *patterns* in LAD. We shall use LAD terminology from now on. Several approaches have been proposed in the literature for pattern generation, ranging from enumeration (by Alexe *et al.* (2006) and Alexe and Hammer (2006)), heuristics (by Bonates *et al.* (2008) and Caserta and Reiners (2016)), and mathematical programming (by Bonates *et al.* (2008), Ryoo and Jang (2009), Guo and Ryoo (2012), and Yan and Ryoo (2017a)). Yan and Ryoo (2017b) and Yan and Ryoo (2019) generate valid inequalities for existing mathematical programming models for pattern generation. Mathematical programming has also been used for generating patterns in studies that extended the original LAD framework: Lejeune (2012) applies pattern generation for solving stochastically constrained optimization problems, while Bonates (2010), Hansen and Meyer (2011), and Chou *et al.* (2017) generate patterns for building large margin LAD classifiers. More recently, Boccia *et al.* (2020) proposed exact and heuristic algorithms for solving two variants of the Simple Pattern Minimality Problem, which asks for the minimum number of patterns explaining all the observations of a data set.

Let $\Omega \subseteq \{0,1\}^n$ be a dataset consisting of binary data, and partitioned into two sets of observations, one called *positive* and the other *negative*. A subcube of $\{0,1\}^n$ is a subset of $\{0,1\}^n$ in which a subset of the coordinates is fixed. A positive pattern is a subcube of $\{0,1\}^n$ having a nonempty intersection with the positive part of Ω, and an empty intersection with the negative part of Ω. A negative pattern is defined analogously. An observation is covered by a pattern if it belongs to the corresponding subcube, and the coverage of a pattern is the number of observations in Ω covered by it. The maximum α-pattern problem (α-MPP) consists in finding

a pattern whose coverage is maximum among those that cover a given observation $\alpha \in \Omega$, which amounts to solving a nonlinear set covering problem. The problem is NP-hard (BONATES *et al.*, 2008). A generalization of the α-MPP, the *maximum pattern problem* (MPP), asks for a pattern of maximum coverage among all patterns, not only among those covering a particular observation. We point out that we shall not deal in this work with the concept of "fuzzy" patterns, which are patterns that may have a nonempty intersection with both the positive and the negative part of Ω. The main objective of this work is to model and solve both the α-MPP and the MPP using a recently-developed methodology based on decision diagrams (DDs) for solving discrete optimization problems.

Binary decision diagrams (BDDs) were originally used in the analysis and verification of digital circuits (LEE, 1959), and as a graphical data structure to represent Boolean functions (AKERS, 1978). Efficient algorithms for manipulating BDDs were introduced in (BRYANT, 1986), which led to their further use in computer science applications (WEGENER, 2000). The application of BDDs has recently expanded to the field of optimization, with BDDs being used to (approximately) represent the set of feasible solutions of a binary optimization problem. Furthermore, the concepts of BDDs can be generalized to *multivalued decision diagrams* (MDDs), or simply DDs, in order to represent general discrete optimization problems.

DDs have been used either as an auxiliary tool, for instance in constraint programming and integer programming contexts, or as a standalone methodology for dealing with combinatorial optimization problems. In *constraint programming* (CP), Andersen *et al.* (2007), Hoda *et al.* (2010), and Bergman *et al.* (2014) applied MDDs to constraint propagation. In the context of *integer programming* (IP), Hadžić and Hooker (2007), and Serra and Hooker (2017) applied DDs to postoptimality analysis; Becker *et al.* (2005) and Tjandraatmadja and Hoeve (2019) applied DDs to cut generation; while Behle and Eisenbrand (2007) applied BDDs to vertex and facet enumeration. Cire and Hoeve (2013), Kinable *et al.* (2017), and O'Neil and Hoffman (2019) applied MDDs to sequencing problems, which includes scheduling and routing. Bergman and Cire (2017) applied DDs to optimization problems with nonlinear objective functions.

The use of DDs allows for recursive modeling through *dynamic programming* (DP) formulations. Hooker (2013) showed how DDs are closely related to DP but differ in some important ways. In (BERGMAN *et al.*, 2011) and (BERGMAN *et al.*, 2013), the authors used the concept of *relaxed* DDs introduced by Andersen *et al.* (2007) for generating bounds for the set covering and the maximum independent set problems, respectively. Bergman *et al.* (2015) used

the concept of relaxed DDs to generate Lagrangian bounds. Bergman *et al.* (2014) introduced the concept of *restricted* DDs, which were used as primal heuristics for generating bounds for the set covering and set packing problems. Bergman *et al.* (2016) proposed a general-purpose DD-based solver for discrete optimization problems that incorporates restricted and relaxed DDs into a branch-and-bound (BAB) scheme, which proved to be competitive or superior to a state-of-the-art commercial ILP solver for the maximum independent set, the maximum cut, and the maximum 2-satisfiability problems.

1.1 Objectives

Our main objective in this work is to propose a new solution approach for the α-MPP and the MPP and to empirically evaluate its computational behaviour. More specifically, we intend to:

- evaluate the integer linear programming models found in the literature, by measuring their computational performance using a state-of-the art ILP solver;
- to present a detailed description of the DD-based BAB methodology proposed by (BERGMAN *et al.*, 2016), which we use for solving both problems;
- propose a DD model for the α-MPP and MPP problems and discuss all relevant details for its computational implementation;
- use the computational results obtained from the empirical evaluation of the ILP models as a benchmark for an empirical evaluation of our proposed DD-based BAB.

1.2 Organization

In Chapter 2, we present a formal definition of the α-MPP and the MPP, as well as the integer linear programming models found in the literature for these problems. In Chapter 3, we present a detailed description of the general DD-based BAB algorithm proposed by Bergman *et al.* (2016). In Chapter 4, we present a DD model that encompasses both problems, by proposing a dynamic programming model, a merging rule, and relevant heuristics in order to use our proposed DD model within the BAB framework of Bergman *et al.* (2016). In Chapter 5, we perform a series of computational experiments with the ILP models and the proposed DD-based BAB and discuss the results. In Chapter 6, we discuss the conclusions of our work.

2 MAXIMUM PATTERNS

In this chapter, we present the problem of generating patterns from an optimization perspective. In Section 2.1, we introduce the notation and definitions to be used throughout the chapter. In Sections 2.2, 2.3, and 2.4, we present three optimization problems found in the literature that can be used for pattern generation. In Sections 2.5, 2.6, 2.7, and 2.8 we present existing integer programming formulations for these problems from Bonates *et al.* (2008), Ryoo and Jang (2009), Guo and Ryoo (2012), and Yan and Ryoo (2017a), respectively.

2.1 Definitions

Let $\Omega = \{x^1, x^2, \ldots, x^m\} \subseteq \{0,1\}^n$ be a dataset consisting of m binary observations on n attributes, with two classes. Let the class of an observation $x \in \Omega$ be given by the function $y : \Omega \to \{0,1\}$. We refer to Ω^+ as the set of positive observations of Ω:

$$\Omega^+ = \{x \in \Omega : y(x) = 1\}.$$

The set Ω^- of negative observations of Ω is defined in a complementary way. Thus, $\Omega^+ \cup \Omega^- = \Omega$. An example of a dataset of binary observations is given in Table 1. If the observations in Ω are not binary, they can be transformed into binary vectors via a process called *binarization* (BOROS *et al.*, 1997). Datasets with more than two classes can be handled by means of a so-called *one-versus-all* approach, in which one of the classes play the role of the positive class, while the remaining classes play the role of the negative class.

Table 1 – A binary dataset

	k	x_1^k	x_2^k	x_3^k	x_4^k	x_5^k	$y(x^k)$
	1	1	0	1	1	1	1
	2	0	0	0	1	1	1
Ω^+	3	1	1	1	1	1	1
	4	1	1	1	0	1	1
	5	1	1	1	0	0	1
	6	1	0	0	1	0	0
	7	0	0	1	0	1	0
Ω^-	8	1	0	1	0	0	0
	9	1	0	0	0	0	0
	10	0	0	1	0	0	0

Source: Hammer *et al.* (2004)

For each attribute $j \in \{1, \ldots, n\}$, we define a positive *literal* u_j and a negative literal

$\overline{u_j}$ as functions from Ω to $\{0,1\}$, as follows:

$$u_j(x) = \begin{cases} 1, & \text{if } x_j = 1, \\ 0, & \text{if } x_j = 0; \end{cases} \quad \text{and} \quad \overline{u_j}(x) = \begin{cases} 1, & \text{if } x_j = 0, \\ 0, & \text{if } x_j = 1. \end{cases}$$

Let $\mathscr{L} = \{u_1, \overline{u_1}, \ldots, u_n, \overline{u_n}\}$ be the set of all literals for dataset Ω. A *term* T is a product of one or more literals. We shall denote by $\text{Lit}(T)$ the set of literals that compose T. We denote by $\mathscr{T}$ the set of all possible terms. The *degree* $\delta(T)$ of a term T is the number of literals that compose T, i.e., $\delta(T) = |\text{Lit}(T)|$. Thus, we can write T as a function $T : \Omega \to \{0,1\}$ as follows, where ℓ is a generic literal:

$$T(x) = \prod_{\ell \in \text{Lit}(T)} \ell(x).$$

We say that an observation x is *covered* by T, or that T *covers* x, if, and only if, $T(x)$ equals 1. Let $\text{Cov}(T)$ be the set of observations in $\{0,1\}^n$ covered by T,

$$\text{Cov}(T) = \{x \in \{0,1\}^n : T(x) = 1\},$$

and let $\text{Cov}_\Omega(T)$ be the set of observations in Ω covered by T:

$$\text{Cov}_\Omega(T) = \{x \in \Omega : T(x) = 1\}.$$

Unless there is the need to avoid ambiguity, we will simply write $\text{Cov}(T)$ instead of $\text{Cov}_\Omega(T)$. The *coverage* of term T is the number of observations in Ω covered by T, i.e., $|\text{Cov}(T)|$. The subset of $\{0,1\}^n$ covered by a term T is known as a *subcube* of $\{0,1\}^n$, since it is isomorphic to the hypercube $\{0,1\}^{n-\delta(T)}$.

Example 1. *Consider the binary dataset given in Table 1. The term* $T = \overline{u_2}\overline{u_3}u_4$ *is composed by the set of literals* $\text{Lit}(T) = \{\overline{u_2}, \overline{u_3}, u_4\}$*, and its degree is* $\delta(T) = 3$*. Furthermore, the set of observations in* $\{0,1\}^n$ *covered by* T *is* $\text{Cov}(T) = \{(0,0,0,1,0), (1,0,0,1,0), (0,0,0,1,1), (1,0,0,1,1)\}$ *and the set of covered observations by* T *in* Ω *is* $\text{Cov}_\Omega(T) = \{x^2, x^6\}$.

A term P is a *positive pattern* with respect to Ω if it covers at least one positive observation from Ω and no negative observation from Ω:

$$|\text{Cov}(P)| > 0 \text{ and } \text{Cov}(P) \cap \Omega^- = \emptyset.$$

A *negative pattern* is defined analogously. Without loss of generality, we shall refer to a positive pattern simply as a *pattern*, since everything that follows can be defined analogously for negative patterns.

Table 2 – Examples of patterns in the dataset given in Table 1

Examples	Prime	Strong	Spanned	x^2-pattern
$u_2 u_3 \overline{u_4}$, $u_3 u_4 u_5$	No	No	No	No
$\overline{u_1}\,\overline{u_3}$, $\overline{u_1} u_4$, $\overline{u_3} u_5$	Yes	No	No	Yes
$u_1 u_2$, $u_2 u_3$	No	Yes	No	No
$u_1 u_2 u_3 \overline{u_4}$	No	No	Yes	No
u_2, $u_1 u_5$	Yes	Yes	No	No
$u_1 u_2 u_3$, $u_1 u_3 u_5$	No	Yes	Yes	No
$u_4 u_5$	Yes	Yes	Yes	Yes

Source: Adapted from Hammer *et al.* (2004)

Hammer *et al.* (2004) define three types of patterns according to their suitability in a classification context: *prime*, *strong*, and *spanned*. A pattern P is prime if there is no pattern P' such that $\mathrm{Lit}(P') \subset \mathrm{Lit}(P)$, i.e., if the removal of any literal from P results in a term which is not a pattern. A pattern P is *strong* if there is no pattern P' such that $\mathrm{Cov}(P) \subset \mathrm{Cov}(P')$. A pattern is spanned if there is no pattern P' such that $\mathrm{Cov}(P) = \mathrm{Cov}(P')$ and $\mathrm{Lit}(P) \subset \mathrm{Lit}(P')$. Bonates *et al.* (2008) defines yet another type of pattern: an *α-pattern* is a pattern that covers a given positive observation α. Examples of patterns are given in Table 2.

2.2 The maximum α-pattern problem (α-MPP)

A *maximum α-pattern* is a pattern of maximum coverage among those that cover a positive observation α. The *maximum α-pattern problem (α-MPP)* consists in finding a maximum α-pattern among all possible terms:

$$(\alpha\text{-MPP}) \text{ maximize} \quad |\mathrm{Cov}(T)| \tag{2.1}$$

$$\text{subject to} \quad T(x) = 0, \quad \forall\, x \in \Omega^-, \tag{2.2}$$

$$T(\alpha) = 1, \tag{2.3}$$

$$T \in \mathscr{T}. \tag{2.4}$$

The α-MPP was proposed by Bonates *et al.* (2008). A maximum α-pattern is a strong pattern. Strong patterns are useful in the sense that they can be descriptive of large portions of a dataset. The pattern generation step in LAD using maximum α-patterns consists in computing a set of maximum α-patterns in such a way that each positive observation is covered by at least one of the patterns in the set.

The α-MPP is a restricted version of the problem, where a pattern is not allowed to cover any observation of the opposite class. These patterns are referred to as "pure". In practice, a relaxed version of the α-MPP, which allows a pattern to cover a "small" number of

observations of the opposite class, might be more adequate in a classification context. These patterns are called "fuzzy". In this work, however, we shall focus our attention to the restricted version of the problem.

2.2.1 Complexity

In (BONATES *et al.*, 2008), the fact that the α-MPP is a generalization of the unitary-cost set cover problem was assumed to be evident. Here, we discuss this in more detail as we argue the hardness of the α-MPP. Let $S = \{1, \ldots, m\}$ be a set of m elements and $\mathscr{F} = \{S_1, S_2, ..., S_n\}$ be a collection of n subsets of S. *Minimum unitary-cost set cover* is the problem of selecting as few as possible subsets from $\mathscr{F}$ such that every element in S is contained in at least one of the selected subsets. The minimum set cover problem is one of the classical combinatorial optimization problems shown to be NP-hard (KARP, 1972). An IP formulation for it is given by

$$\text{minimize} \quad \sum_{j=1}^{n} x_j \tag{2.5}$$

$$\text{subject to} \quad \sum_{j:\ i \in S_j} x_j \geq 1, \quad i = 1, \ldots, m, \tag{2.6}$$

$$x_j \in \{0,1\}, \quad j = 1, \ldots, n, \tag{2.7}$$

where x_j is a binary decision variable that equals 1 when subset S_j is selected. Constraints (2.6) state that for every element $i \in S$, at least one subset that contains i is selected. In the following, we show that the α-MPP is at least as hard to solve as the unitary-cost set cover problem.

Proposition 2.2.1. *The α-MPP is NP-hard.*

Proof. Let (2.6) be rewritten in matricial form as $Ax \geq \mathbb{1}$, where A is an $m \times n$ matrix, and x and $\mathbb{1}$ are the n-vectors $(x_1, \ldots, x_n)^T$ and $(1, \ldots, 1)^T$, respectively. Let $\Omega^- = \{\gamma^1, \ldots, \gamma^m\}$ be comprised of m observations accounting for each row of A, i.e., $\gamma^i = A_i$, the i-th row of A, for $i = 1, \ldots, m$. Let Ω^+ be comprised of $n+1$ observations, where α is the n-vector $(0, \ldots, 0)$ and the other n observations are the canonical vectors $e_j = (0, \ldots, 0, 1, 0, \ldots, 0)$, for $j = 1, \ldots, n$. We claim that solving the α-MPP for $\Omega^+ \cup \Omega^-$ is equivalent to solving (2.5)-(2.7). The conversion can be done in polynomial time.

Let P^* be an optimal solution for this instance of the α-MPP. Any feasible pattern for this instance must contain only negative literals. Let $J^* = \{j : \overline{u}_j \in P^*\}$ be the set of indexes of the negative literals that compose P^*. We know that for a given negative observation γ^i, there

is at least one attribute $j \in J^*$, such that $\gamma_j^i = 1$. Let x^* be a solution to the set cover instance, where $x_j^* = 1,\ \forall j \in J^*$, and $x_j^* = 0,\ \forall j \notin J^*$. Thus, x^* is feasible for the set cover instance, since each constraint is associated with a negative observation, i.e., $A_i x^* \geq 1$, for $i = 1, \ldots, m$.

It remains to show that x^* is an optimal solution for the set cover problem. We prove this by contradiction. Suppose there exists a better feasible solution $\hat{x}$, i.e., $\mathbb{1}^T\hat{x} < \mathbb{1}^T x^*$. Let $\hat{J} = \{j : \hat{x}_j = 1\}$. It can be seen that the pattern $\hat{P}$, with $\text{Lit}(\hat{P}) = \{\overline{u_j} : j \in \hat{J}\}$, is a feasible solution for the α-MPP instance. The objective function value for $\hat{P}$ is $|\text{Cov}(\hat{P})| = n - \delta(\hat{P}) + 1$. From the optimality of P^*, we know that $|\text{Cov}(\hat{P})| \leq |\text{Cov}(P^*)| \implies n - \delta(\hat{P}) + 1 \leq n - \delta(P^*) + 1$. Thus, $\delta(P^*) \leq \delta(\hat{P}) \implies \mathbb{1}^T x^* \leq \mathbb{1}^T\hat{x}$, a contradiction. Therefore, x^* is optimal. □

2.3 The general case: the maximum pattern problem (MPP)

Let $R = \{T \in \mathscr{T} : T(x) = 0, \forall\ x \in \Omega^-, |\text{Cov}(T)| > 0\}$ be the set of all patterns. A pattern is a *maximum pattern* if its coverage is maximum among all patterns. Let $R^* = \arg\max\{|\text{Cov}(P)| : P \in R\}$ be the set of all maximum patterns. Ryoo and Jang (2009) proposed an MILP formulation that can be easily adjusted for generating patterns with respect to different criteria. We shall call the main problem associated with their formulation as the *maximum pattern problem (MPP)*, which asks for a maximum pattern $P^* \in R^*$. The MPP is a generalization of the α-MPP, which is obtained by leaving out constraint (2.3). Observe that, by definition, a maximum pattern is strong. Furthermore, a spanned pattern P' can easily be obtained from a pattern P by adding all "missing" literals in P, i.e., $P' = P \cup \{\ell \in \mathscr{L} \setminus \text{Lit}(P) : \ell(x) = 1,\ \forall\ x \in \text{Cov}(P)\}$. Using the MPP in the LAD methodology is a more natural approach than using the α-MPP, because it frees the user from having to choose an observation α at each iteration of the process of generating patterns in LAD. Moreover, due to the order in which the α observations are selected, it is possible that no optimal pattern to the MPP is found in the process of computing enough α-patterns to cover Ω^+.

2.4 Optimal prime patterns: the maximum pattern of minimum degree problem (MP-MDP)

As discussed in the previous section, the MPP allows us to find strong patterns and spanned patterns of maximum coverage. Spanned patterns offer a safeguard against "false positives" in LAD, while prime patterns provide a safeguard against "false negatives" (HAMMER

et al., 2004). Thus, we might also be interested in finding prime patterns of maximum coverage. Guo and Ryoo (2012) adapted their formulation for the MPP to generate prime patterns. We hereby define the maximum pattern of minimum degree problem (MPMDP), which asks for a maximum pattern P^* of minimum degree, i.e., $P^* \in \arg\min\{\delta(P) : P \in R^*\}$. We note that P^* is a prime pattern. Prime patterns are useful in the sense that their smaller number of literals could make them, in theory, more easily understood by experts, while also performing well as classification rules.

Example 2. *Consider the dataset given in Table 1. There are 7 maximum patterns,* $R^* = \{u_1u_2,\ u_1u_2u_3,\ u_1u_3u_5,\ u_1u_5,\ u_2,\ u_2u_3,\ u_4u_5\}$*, each covering 3 observations. The maximum pattern of minimum degree is* u_2*, which covers observations* x^3*,* x^4*, and* x^5*.*

2.5 Bonates *et al.* ILP model for the α-MPP

Let $w(\beta) = \{j \in \{1,\ldots,n\} : \beta_j \neq \alpha_j\}$ be the set of indexes of the attributes in which observation $\beta \in \Omega^+$ differs from α. Let y_j be a binary decision variable. If $\alpha_j = 1$, then y_j equals 1 if the pattern contains u_j. If $\alpha_j = 0$, then y_j equals 1 if the pattern contains $\overline{u_j}$. The α-MPP can be formulated as the following integer nonlinear programming model (BHK-NL), which amounts to a nonlinear objective function subject to set covering constraints:

$$\text{(BHK-NL) maximize} \quad \sum_{\beta \in \Omega^+ \setminus \{\alpha\}} \prod_{j \in w(\beta)} (1 - y_j) \tag{2.8}$$

$$\text{subject to} \quad \sum_{j \in w(\gamma)} y_j \geq 1, \qquad \forall\, \gamma \in \Omega^-, \tag{2.9}$$

$$y_j \in \{0,1\} \qquad j = 1,\ldots,n. \tag{2.10}$$

For a given observation β, the product of the terms $1 - y_j$, with $\alpha_j \neq \beta_j$, equals 1 only when the variables y_j are set to 0, i.e., when there are no conflicts between β and the literals selected to belong to the pattern. In such a case, the pattern defined by y_j-variables covers β. Objective function (2.8) is the sum of products defined for each positive observation, except for α, i.e, it amounts to the number of additional observations covered by the pattern defined by the y_j-variables. Set of constraints (2.9) state that for each negative observation γ, at least one of the literals in $w(\gamma)$ is selected, ensuring that γ is not covered by the pattern defined by the y_j-variables.

The BHK-NL model can be linearized via the introduction of another set of decision variables. Let z_β be a binary decision variable, for $\beta \in \Omega^+ \setminus \{\alpha\}$, which equals 1 if β is covered

by the pattern defined by the y_j-variables:

$$z_\beta = \begin{cases} 1, & \text{if } \beta \text{ is covered by the pattern;} \\ 0, & \text{otherwise.} \end{cases}$$

The BHK-NL model can be reformulated as the following integer linear programming model (BHK):

$$\text{(BHK) maximize} \quad \sum_{\beta \in \Omega^+ \setminus \{\alpha\}} z_\beta \tag{2.11}$$

$$\text{subject to} \quad \sum_{j \in w(\gamma)} y_j \geq 1, \qquad \forall\, \gamma \in \Omega^- \tag{2.12}$$

$$|w(\beta)| \cdot z_\beta + \sum_{j \in w(\beta)} y_j \leq |w(\beta)|, \quad \forall\, \beta \in \Omega^+ \setminus \{\alpha\}, \tag{2.13}$$

$$y_j \in \{0,1\}, \qquad j = 1, \ldots, n, \tag{2.14}$$

$$z_\beta \in \{0,1\}, \qquad \forall\, \beta \in \Omega^+ \setminus \{\alpha\}. \tag{2.15}$$

Objective function (2.11) is now straightforward: it amounts to the number of observations covered by the pattern, in addition to α itself. However, the set of constraints (2.13) is added in order to link y_j-variables to z_β-variables. If the pattern defined by y_j-variables covers β, then the sum of the y_j-variables in which α and β differ for attribute j equals 0. In that case, a constraint in (2.13) amounts to $z_\beta \leq 1$. Observe that, in such a case, the sense of the objective function will ensure that z_β is set to 1. On the other hand, if there is a number $k \geq 1$ of literals in the pattern defined by the y_j-variables in which α and β differ for attribute j, then the corresponding constraint in (2.13) amounts to $z_\beta \leq (|w(\beta)| - k)/|w(\beta)|$. In that case, since $z_\beta < 1$, it is set to 0.

2.6 Ryoo and Jang MILP model for the MPP

Let z_β be defined as in Section 2.5. Let y_j^+ and y_j^- be binary decision variables, where y_j^+ equals 1 when the positive literal u_j belongs to the pattern, and y_j^- equals 1 when the negative literal $\overline{u_j}$ belongs to the pattern, respectively:

$$y_j^+ = \begin{cases} 1, & \text{if } u_j \text{ belongs to the pattern;} \\ 0, & \text{otherwise;} \end{cases} \quad \text{and } y_j^- = \begin{cases} 1, & \text{if } \overline{u_j} \text{ belongs to the pattern;} \\ 0, & \text{otherwise.} \end{cases}$$

Ryoo and Jang (2009) formulation for the MPP introduces an integer decision variable d, which indicates the degree of the pattern:

$$\text{(RJ) minimize} \quad \sum_{\beta \in \Omega^+} (1 - z_\beta) \tag{2.16}$$

$$\text{subject to} \quad \sum_{j:\ \gamma_j=1} y_j^+ + \sum_{j:\ \gamma_j=0} y_j^- \leq d-1, \qquad \forall\, \gamma \in \Omega^- \tag{2.17}$$

$$n \cdot (1 - z_\beta) + \sum_{j:\ \beta_j=1} y_j^+ + \sum_{j:\ \beta_j=0} y_j^- \geq d, \quad \forall\, \beta \in \Omega^+ \tag{2.18}$$

$$y_j^+ + y_j^- \leq 1 \qquad j = 1, \ldots, n \tag{2.19}$$

$$\sum_{j=1}^{n} (y_j^+ + y_j^-) = d \tag{2.20}$$

$$y_j^+,\ y_j^- \in \{0,1\}, \qquad j = 1, \ldots, n, \tag{2.21}$$

$$z_\beta \in \{0,1\}, \qquad \forall\, \beta \in \Omega^+, \tag{2.22}$$

$$1 \leq d \leq n. \tag{2.23}$$

Objective function (2.16) corresponds to the number of positive observations not covered by the pattern and is to be minimized. Constraints (2.17) state that, given a negative observation γ and a pattern of degree d, there are at most $d-1$ literals in the pattern that agree with γ, i.e., there is at least one literal in the pattern which is in conflict with γ. Constraints (2.18) state that if less than d literals that are selected agree with a given positive observation β, then β is not covered by the pattern. Otherwise, the minimization of objective function ensures that z_β is set to 1. Constraints (2.19) state that a pattern will not contain a positive literal and its corresponding negative literal simultaneously. Note that constraints (2.19) can be omitted from the formulation. Constraints (2.20) state that the pattern contains d literals. Since d is the sum of binary variables, its integrality condition can be relaxed.

2.7 Guo and Ryoo MILP model for the MPP

Guo and Ryoo (2012) proposed a formulation for the MPP, which can be seen as a generalization of BHK, and which, according to the authors, performs better in practice than RJ. Let y_j^+ and y_j^- be binary decision variables as defined in Section 2.6. Let z_β be defined as in Section 2.5. For a given $\beta \in \Omega$, let $w^+(\beta) = \{j \in \{1, \ldots, n\} : \beta_j = 0\}$ be the set of indexes of the attributes in which β has a conflict with the corresponding positive literal, that is, the set of indexes j in which $u_j(\beta) = 0$. Let $\omega^-(\beta)$ be similarly defined, that is, $w^-(\beta) = \{j \in \{1, \ldots, n\} : \beta_j = 1\}$ is the set of indexes j in which $\overline{u_j}(\beta) = 0$. The MPP can be

formulated as

$$\text{(GR) maximize} \quad \sum_{\beta \in \Omega^+} z_\beta \tag{2.24}$$

$$\text{subject to} \quad \sum_{j \in \omega^+(\gamma)} y_j^+ + \sum_{j \in \omega^-(\gamma)} y_j^- \geq 1, \quad \forall\, \gamma \in \Omega^-, \tag{2.25}$$

$$z_\beta + y_j^+ \leq 1, \qquad \forall\, \beta \in \Omega^+,\ \forall j \in w^+(\beta) \tag{2.26}$$

$$z_\beta + y_j^- \leq 1, \qquad \forall\, \beta \in \Omega^+,\ \forall j \in w^-(\beta) \tag{2.27}$$

$$y_j^+,\ y_j^- \in \{0,1\}, \qquad j = 1,\ldots,n, \tag{2.28}$$

$$z_\beta \in \{0,1\}, \qquad \forall\, \beta \in \Omega^+ \tag{2.29}$$

Set of constraints (2.25) is a generalized version of (2.12). Constraints (2.26) state that given an observation β, if β is covered, i.e., $z_\beta = 1$, then y_j^+ must be set to 0 in those cases in which β does not agree with the corresponding literal, i.e., in those cases where $\beta_j = 0$. Conversely, if $\beta_j = 0$ and the positive literal for attribute j is selected, i.e., $y_j^+ = 1$, then z_β must be set to 0. Thus, z_β and y_j cannot equal 1 simultaneously. The role played by constraints (2.27) is analogous to that of (2.26). Furthermore, given an observation β, if y_j^+ equals 0 for every $j \in w^+(\beta)$ and y_j^- equals 0 for every $j \in w^-(\beta)$, then each constraint in (2.26) and (2.27) amounts to $z_\beta \leq 1$. Since the objective function is to be maximized, z_β is set to 1.

2.7.1 Adaptation to the MPMDP

Let ω be a real number, with $\omega \in [-1/(n+1), 0)$. Guo and Ryoo (2012) adapted GR for generating prime patterns by altering the objective function (2.24) to

$$\sum_{\beta \in \Omega^+} z_\beta + \omega \cdot \sum_{j=1}^{n} (y_j^+ + y_j^-) \tag{2.30}$$

Let z_{3a}^* be the optimal value for GR and let z_{3b}^* be the optimal value for GR with objective function (2.30). Notice that $z_{3b}^* \in [z_{3a}^* - n/(n+1), z_{3a}^*)$. The coefficient ω in the second term of the objective function is chosen in order to guarantee that any optimal solution for GR with objective function (2.30) still has maximum coverage, while penalizing the number of literals in the resulting pattern. Observe that if $\omega \in [-1/(n+1), 0)$, then the formulation solves the MPMDP.

2.8 Yan and Ryoo MILP models for the MPP

Depending on the dataset, the number of constraints defined by (2.26) and (2.27) may be too large. Yan and Ryoo (2017a) proposed two alternative sets of constraints that can replace (2.26) and (2.27) in model (GR). The first set is given by

$$n \cdot z_\beta + \sum_{\omega^+(\beta)} y_j^+ + \sum_{\omega^-(\beta)} y_j^- \leq n, \; \forall \, \beta \in \Omega^+, \tag{2.31}$$

which is a generalization of (2.13). The linear relaxation of an alternative formulation, which we shall refer to as YRa, obtained by replacing (2.26) and (2.27) with (2.31),

$$\text{(YRa) maximize } (2.24), \text{ subject to } (2.25), \; (2.31), \; (2.28), \; (2.29),$$

is weaker than GR. Indeed, observe that a constraint for a given observation β in (2.31) is a linear combination of constraints in (2.26) and (2.27). Thus, every point that satisfies both (2.26) and (2.27), also satisfies (2.31). On the other hand, a point satisfying (2.31) may not satisfy (2.26) and (2.27). For instance, a feasible solution (for the linear relaxation) in which $z_\beta = 1/n$, $y_k^+ = 1$, for some $k \in w^+(\beta)$, $y_j^+ = 0$, $\forall j \in w^+(\beta) \setminus \{k\}$, and $y_j^- = 0$, $\forall j \in w^-(\beta)$, satisfies (2.31) if $n \geq 2$, but does not satisfy (2.26).

Despite YRa having a weaker relaxation than GR, (2.31) contains only $|\Omega^+|$ constraints, while (2.26) and (2.27) contain $n \cdot |\Omega^+|$ constraints. Another set of constraints, which contains only $2n$ constraints, can be defined by means of a different linear combination of constraints (2.26) and (2.27), such that each literal – instead of each positive observation – defines a constraint:

$$|v^+(j)| \cdot y_j^+ + \sum_{\beta \in v^+(j)} z_\beta \leq |v^+(j)|, \; j = 1, \ldots, n, \tag{2.32}$$

$$|v^-(j)| \cdot y_j^- + \sum_{\beta \in v^-(j)} z_\beta \leq |v^-(j)|, \; j = 1, \ldots, n, \tag{2.33}$$

where $v^+(j) = \{\beta \in \Omega^+ : \beta_j = 0\}$ and $v^-(j) = \{\beta \in \Omega^+ : \beta_j = 1\}$. If y_j^+ equals 1, then z_β must equal 0 for every observation $\beta \in \Omega^+$ in which $\beta_j = 0$. The idea is analogous if $y_j^- = 1$. If $y_j^+ = 0$, then the positive literal for attribute j is not selected and the constraint does not forbid the observations $\beta \in |v^+(j)|$ from being covered by the pattern. Since the objective function is to be maximized, as many z_β-variables as possible are set to 1. Depending on the dataset, it may be the case that the number of literals is inferior to the number of positive observations. An alternative formulation YRb can be obtained from GR by replacing both (2.26) and (2.27) with

(2.32) and (2.33):

$$\text{(YRb) maximize } (2.24), \text{ subject to } (2.25),\ (2.32),\ (2.33),\ (2.28),\ (2.29).$$

2.8.1 Valid inequalities

In (YAN; RYOO, 2017b), the authors used graph theoretic analysis of data to discover useful neighborhood properties among data, which allowed them to derive the so-called (extended) hypercube inequalities. These inequalities are stronger than the so-called minimal cover inequalities in (2.25) that yield them.

Example 3. *Observations x^7 and x^{10} in Table 1 define the minimal cover inequalities $y_1^+ + y_2^+ + y_3^- + y_4^+ + y_5^- \geq 1$ and $y_1^+ + y_2^+ + y_3^- + y_4^+ + y_5^+ \geq 1$, respectively. The Hamming distance between observations x^7 and x^{10} is 1, that is, they differ in only one attribute. Thus, x^7 and x^{10} are considered to be neighbours. Furthermore, they compose a hypercube of dimension $d = 1$ and this neighbourhood property allows us to replace the corresponding $2^d = 2^1$ inequalities with a single stronger hypercube inequality, $y_1^+ + y_2^+ + y_3^- + y_4^+ \geq 1$.*

In (YAN; RYOO, 2019), the authors further enhanced the GR model by discovering a new neighborhood property among data, allowing them to derive what we shall refer to as clique inequalities, which are stronger than the so-called McCormick inequalities in (2.26) and (2.27) that yield them.

Table 3 – Pattern generation models

Model name	Base model	Yan and Ryoo (2017b)'s extended hypercube inequalities	Yan and Ryoo (2019)'s clique inequalities
GR	GR	No	No
$\text{GR}^{\texttt{ehi}}$	GR	Yes	No
$\text{GR}_{\texttt{cliques}}$	GR	No	Yes
$\text{GR}^{\texttt{ehi}}_{\texttt{cliques}}$	GR	Yes	Yes
YRa	YRa	No	No
$\text{YRa}^{\texttt{ehi}}$	YRa	Yes	No
YRb	YRb	No	No
$\text{YRb}^{\texttt{ehi}}$	YRb	Yes	No

Source: The author.

Example 4. *Observations x^2 and x^4 in Table 1 define the McCormick inequalities $z_{x^2} + y_5^- \leq 1$ and $z_{x^4} + y_5^- \leq 1$ for literal $\overline{u_5}$, respectively. Let I be the set of literals that assume value 1*

on both observations, i.e., $I = \{u_5\}$. *Since I defines a term* $T = u_5$ *which covers a negative observation, in this case* x^7*, then* x^2 *and* x^4 *are considered to be neighbours. Consider a graph where the nodes are the positive observations and the edges connect the observations that are neighbours. In such a graph,* x^2 *and* x^4 *compose a maximal clique. That allows us to replace the corresponding 2 inequalities with a single stronger clique inequality,* $z_{x^2} + z_{x^4} + y_5^- \leq 1$.

We shall not get into further details into the theory behind these inequalities as this would extend way beyond the focus of our work. However, we use them in our computational experiments, which are reported in Chapter 5. We shall use the nomenclature specified in Table 3. For the GR model, we can use the extended hypercube inequalities, the clique inequalities, or both. For the YRa and YRb models we can only use the extended hypercube inequalities.

3 DECISION DIAGRAMS FOR OPTIMIZATION: AN OVERVIEW

In this chapter, we present the DD-based branch-and-bound proposed by Bergman *et al.* (2016) for solving a discrete optimization problem $\mathscr{P}$. In Sections 3.1 and 3.2, we formally define $\mathscr{P}$ and define the notation for DDs, which we use throughout the chapter. In Sections 3.3 to 3.7, we show how to model $\mathscr{P}$ in order to solve it via DDs, and how to compile exact, restricted and relaxed DDs for $\mathscr{P}$. In Section 3.8, we show how to solve $\mathscr{P}$ by incorporating restricted and relaxed DDs into a branch-and-bound scheme.

3.1 Discrete optimization problems

Let $X = \{x_1, \ldots, x_n\}$ be a set of n decision variables, D_j be the domain of x_j, for $j \in \{1, \ldots, n\}$, $D = D_1 \times \cdots \times D_n$ be the Cartesian product of the domains of the variables, and $f : D \to \mathbb{R}$ be a real-valued function over D. Let a constraint $C_i(x)$ be defined as a pair $(Var(C_i), Val(C_i))$, where $Var(C_i) = \{x_{i_1}, \ldots, x_{i_k}\} \subseteq X$ is a subset of k variables, and $Val(C_i) \subseteq D_{i_1} \times \cdots \times D_{i_k}$ is the set of tuples that *satisfy* the constraint, with $1 \leq i_1 \leq \cdots \leq i_k \leq n$. $C_i(x)$ is *satisfied* by $x = (x_1, \ldots, x_n)$ if $(x_{i_1}, \ldots, x_{i_k}) \in Val(C_i)$, and *violated* otherwise. Let $\{C_1, \ldots, C_m\}$ be a set of m constraints. Let $\mathscr{P}$ be a discrete optimization problem of the form:

$$\begin{aligned} (\mathscr{P}) \text{ maximize} \quad & f(x) \\ \text{subject to} \quad & C_i(x), \quad i = 1, \ldots, m \\ & x \in D. \end{aligned}$$

A *solution* of $\mathscr{P}$ is any $x \in D$, and a *feasible solution* to $\mathscr{P}$ is any solution that satisfies all constraints C_i, for $i = 1, \ldots, m$. The set of feasible solutions of $\mathscr{P}$ is denoted by $Sol(\mathscr{P})$. A feasible solution x^* is *optimal* for $\mathscr{P}$ if $f(x^*) \geq f(x)$ for all $x \in Sol(\mathscr{P})$. The optimal solution value $f(x^*)$ is denoted by z^*.

Example 5. *Let E be a set of m elements and $\mathscr{F} = \{E_1, E_2, ..., E_n\}$ be a collection of n subsets of* E. *Let each subset E_j be associated with a cost c_j. Any element $e \in E_j$ is said to be covered by E_j. Weighted set cover is the problem of selecting, at minimum total cost, subsets from $\mathscr{F}$ such that every element in E is covered by at least one of the selected subsets. A formulation for*

it is given by

$$\textit{minimize} \quad \sum_{j=1}^{n} c_j x_j \tag{3.1}$$

$$\textit{subject to} \quad \sum_{j:\, e\in E_j} x_j \geq 1, \quad \forall e \in E \tag{3.2}$$

$$x_j \in \{0,1\}, \quad j = 1,\ldots,n, \tag{3.3}$$

where x_j is a binary decision variable that equals 1 when subset E_j is selected. Constraints (3.2) state that for every element $e \in E$, at least one subset that covers e is selected.

3.2 Decision diagrams: definitions

A decision diagram (DD) is a graphical structure that encodes a set of solutions to problem $\mathscr{P}$. Formally, a decision diagram $B = (U, A, d)$ is a layered directed acyclic multigraph with node set U, arc set A, and arc labels d. The node set U is partitioned into layers $L_1, \ldots, L_{n+1}$, where layers L_1 and L_{n+1} consist each of a single node, the root node r and the terminal node t, respectively:

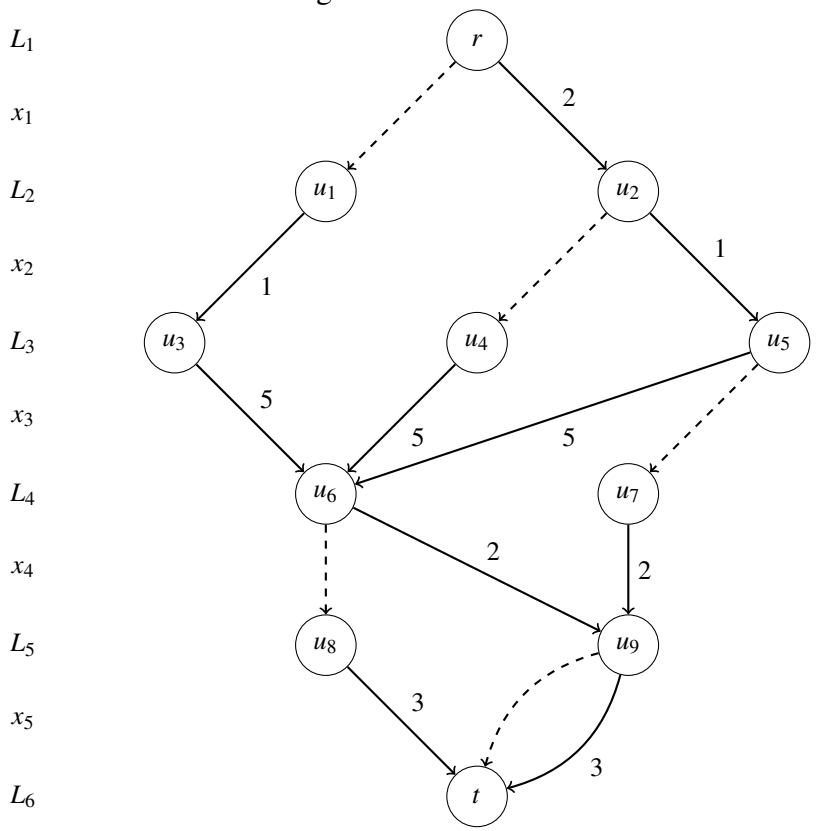
$$U = \bigcup_{j=1}^{n+1} L_j,\ L_1 = \{r\} \text{ and } L_{n+1} = \{t\}.$$

Each arc $a \in A$ is directed from a node in some layer L_j to a node in L_{j+1}, has a label $d(a) \in D_j$, and represents the assignment of value $d(a)$ to variable x_j, for $j = 1,\ldots,n$.

When $D_j = \{0,1\}$, for $j = 1,\ldots,n$, B is called a *binary decision diagram* (BDD). B is called a *multivalued decision diagram* (MDD), otherwise. The index of the layer to which node $u \in L_j$ belongs is denoted by $\ell(u)$, i.e, $\ell(u) = j$. No two arcs leaving the same node have the same label. An arc with label d coming out of a node u is denoted by $a_d(u)$, while $b_d(u)$ denotes the node u' at the endpoint of arc $a_d(u)$. The *width* of layer L_j is the number of nodes in L_j, and the width of B is $\max_j\{|L_j|\}$. The *size* of B is the number of nodes in U. Given $u, w \in U$, let U_{vw} be the set of nodes that belong to some $u-w$ path. We denote by B_{uw} the subgraph of B induced by U_{vw}.

Every arc-specified path $p = \langle a^1, \ldots, a^n \rangle$ from r to t encodes an assignment to the variables $x_1, \ldots, x_n$, namely $x_j = d(a^j)$, for $j = 1,\ldots,n$. This assignment is denoted by x^p. The set of r to t paths of B encodes the set of assignments $Sol(B)$. In *weighted DDs*, each arc $a \in A$ is associated with a weight $v(a)$. In Section 3.3, more details are given on how weighted DDs are used to represent optimization problems. The *length* of p is given by the sum of the weights

Figure 1 – A BDD



Source: The author.
Note: Solid arcs indicate 1-assignments, while dashed arcs indicate 0-assignments. Lengths of solid arcs are specified. Dashed arcs have length 0.

of its arcs:

$$v(p) = \sum_{j=1}^{n} v(a^j).$$

The length of a longest $r-t$ path in B is denoted by $v^*(B)$. Given two nodes u and w, the length of a longest $u-w$ path in B_{uw} is denoted by $v^*(B_{uw})$ Observe that there are no positive-length directed circuits in B, therefore the longest path between two nodes can be computed in polynomial time with the *Dijkstra algorithm* (AHUJA *et al.*, 1988).

Two nodes belonging to the same layer L_j are *equivalent* when the sets of paths from each to the terminal node are the same, i.e., they correspond to the same set of assignments to $(x_j, \ldots, x_n)$, which implies that either of them is redundant in the representation. A *reduced* DD is a DD such that no two nodes of any layer are equivalent (WEGENER, 2000). Reduced DDs can be unsuitable for optimization, because the arc lengths from equivalent nodes may differ (HOOKER, 2013).

Example 6. *The DD of Figure 1 is a BDD with 6 layers and width 3. Each $r-t$ path encodes an assignment to variables x_1 to x_5. For instance, the assignment $\bar{x} = (0,1,1,0,1)$ is encoded by*

the path $r-u_1-u_3-u_6-u_8-t$, which has length 9. The longest $r-t$ path has length 13. The DD is not reduced: observe that nodes u_3 and u_4 are equivalent.

3.2.1 Variable ordering

The ordering of variables can have a significant effect on the size of DDs, and it can also influence the quality of the objective function bounds provided by them (BERGMAN *et al.*, 2012). Let $\sigma : \{1,\dots,n\} \to \{1,\dots,n\}$ be a bijective function that defines a variable ordering. Thus, each arc $a \in A$ directed from a node in some layer L_j to a node in L_{j+1} now represents the assignment of some value $d(a)$ to variable $x_{\sigma(j)}$, for $j = 1,\dots,n$. Without loss of generality, we ignore variable ordering notation in the remaining of this chapter, that is, we define $x_{\sigma(j)} = x_j$.

3.3 Dynamic programming formulations

Formulating $\mathscr{P}$ in order to solve it via DDs requires the formulation of a dynamic programming (DP) model. In this Section, we focus on DP formulations, which are used for compiling exact, restricted, and relaxed DDs. In Section 3.6, we describe the merging operators, which are used for compiling relaxed DDs.

A DP formulation for $\mathscr{P}$ contains three elements: state spaces S_j, transition functions t_j, and transition cost functions h_j. Each state space S_j accounts for the j-th stage of the DP model, for $j = 1,\dots,n+1$. State space S_1 consists of a single state: the root state $\hat{r}$. State space S_{n+1} consist of a finite set of k terminal states $\hat{t_1},\dots,\hat{t_k}$ and an infeasible state $\hat{0}$. Except for S_1, any state space contain an infeasible state $\hat{0}$. Thus,

$$\begin{aligned} S_1 &= \{\hat{r}\}, \\ \hat{0} &\in S_j, \qquad j = 2,\dots,n, \text{ and} \\ S_{n+1} &= \{\hat{t_1},\dots,\hat{t_k},\hat{0}\}. \end{aligned}$$

The transition from state $s^j \in S_j$ to state $s^{j+1} \in S_{j+1}$ is governed by the control variable x_j, i.e., the state s^{j+1} is the result of modifications to the state s^j caused by the assignment of the control variable x_j to some element $d \in D_j$. These modifications are defined by the transition functions $t_j : S_j \times D_j \to S_{j+1}$, for $j = 1,\dots,n$, and their costs are given by the functions $h_j : S_j \times D_j \to \mathbb{R}$, for $j = 1,\dots,n$. A transition from an infeasible state always leads to an infeasible state, i.e., $t_j(\hat{0},d) = \hat{0}$, for all $d \in D_j$.

Let $s = (s^1, \ldots, s^{n+1})$ be a tuple of $n+1$ state variables, $S = S_1 \times \cdots \times S_{n+1}$ be the Cartesian product of the domains of the state variables, i.e., $s^j \in S_j$, for $j = 1, \ldots, n+1$, and let $\hat{f} : S \times D \to \mathbb{R}$ be a real-valued function. A DP formulation for $\mathscr{P}$ can be written as

$$(\mathscr{DP}) \text{ maximize} \quad \hat{f}(s,x) = \sum_{j=1}^{n} h_j(s^j, x_j) \tag{3.4}$$

$$\text{subject to} \qquad s^{j+1} = t_j(s^j, x_j), \qquad j = 1, \ldots, n \tag{3.5}$$

$$s \in S, \tag{3.6}$$

$$x \in D. \tag{3.7}$$

A separable objective function f for $\mathscr{P}$ is a function that can be written as

$$f(x) = \sum_{j=1}^{n} f_j(x_j).$$

A linear function is an example of a separable function. Modeling the transition cost functions when f is separable is simple. It is natural to define

$$f_j(x_j) = h_j(s^j, x_j), \ j = 1, \ldots, n.$$

However, DDs can also accomodate nonseparable objective functions (e.g., nonlinear functions) with the assignment of so-called *canonical* arc costs (HOOKER, 2013).

Example 7. *(BERGMAN et al., 2016) Consider the weighted set cover problem defined in Section 3.1 (Example 5). Let S_j be the following state spaces, for $j = 1, \ldots, n+1$, and let the root state $\hat{r}$ be the set of all elements E:*

$$S_1 = \{E\}, \ S_j = 2^E \cup \{\hat{0}\}, \ j = 2, \ldots, n, \text{ and } S_{n+1} = \{\emptyset\}.$$

Let a state variable s^j represent a subset of elements or the infeasible state. The idea of the DP model is to start from a position where no element is covered. Thus, the root state is the set of all elements. Let x_j, for $j = 1, \ldots, n$, be control variables, such that x_j equals 1 if E_j is selected, and equals 0 otherwise. If a subset is selected, then all elements covered by it are removed in the resulting state. If not, no element is removed from the resulting state. In case there is an element that cannot be covered by any subset whose selection is yet to be decided, the corresponding transition is deemed infeasible, that is, the resulting state is $\hat{0}$. Let the transition

functions $t_j : S_j \times \{0,1\} \to S_{j+1}$, *for* $j = 1,\ldots,n$, *be defined as*

$$t_j(s^j,x_j) = \begin{cases} s^j, & \text{if } x_j = 0 \text{ and } \nexists\, e \in s^j : e \notin E_k,\ \forall k \in \{j+1,\ldots,n\}; \\ s^j \setminus E_j, & \text{if } x_j = 1; \\ \hat{0}, & \text{otherwise.} \end{cases}$$

Recall that the definition of transition function requires that $t_j(\hat{0},x_j) = \hat{0}$, *for all* j. *Given a state* s^j, *a 1-transition is always feasible and leads to the state* $s^j \setminus E_j$. *A 0-transition is feasible when for every element* $e \in s^j$ *there exists a subset* E_k, *with* $k > j$, *such that* $e \in E_k$. *Otherwise, the 0-transition is infeasible. The cost functions* $h_j : S_j \times \{0,1\} \to \mathbb{R}$, *for* $j = 1,\ldots,n$, *are straightforward:*

$$h_j(s^j,x_j) = -c_j x_j.$$

Since the objective function of the problem corresponds to the total weight of the selected subsets, it is natural that a 0-transition has cost 0 and a 1-transition has cost $-c_j$. *The negative sign is introduced in order to be consistent with the definitions for a maximization problem in Section 3.1.*

3.3.1 Validity

A *valid* DP formulation leads to an *exact* DD for $\mathscr{P}$. In Section 3.4, we describe what an *exact* DD is and how it can be compiled. A DP formulation is valid for $\mathscr{P}$ if for every $x \in D$, there is an $s \in S$ such that (s,x) is feasible for $\mathscr{D}\mathscr{P}$, that is, (s,x) satisifies conditions (3.5)-(3.7), and

$$s^{n+1} = \hat{t}_i, \text{ for some } i \in \{1,\ldots,k\},\ \hat{f}(s,x) = f(x), \qquad \text{if } x \text{ is feasible for } \mathscr{P}, \text{ and} \tag{3.8}$$

$$s^{n+1} = \hat{0}, \qquad \text{if } x \text{ is infeasible for } \mathscr{P}. \tag{3.9}$$

Let B_{DP} be the graph obtained from the state-transition graph for the DP model by omitting all occurrences of the infeasible state $\hat{0}$, letting each remaining arc from state s^j to state $t_j(s^j,x_j)$ have length equal to the transition cost $h_j(s^j,x_j)$, and by merging the terminal states $\hat{t}_i$ into a single terminal state $\hat{t}$. B_{DP} is an *exact DD* for $\mathscr{P}$, a concept which we present next in Section 3.4. Each node in B_{DP} is associated with a state from the corresponding DP model. In what follows, no distinction is made between a node in B_{DP} and a state. Furthermore, each layer is associated with an stage, and each arc is associated with a transition, while its weight is associated with the cost of the transition.

3.4 Exact decision diagrams

B is an *exact decision diagram* for $\mathscr{P}$ if the $r-t$ paths in B encode precisely the feasible solutions of $\mathscr{P}$, and the length of any $r-t$ path equals the objective function value of the solution encoded by the path. More formally, B is exact for $\mathscr{P}$ when

$$Sol(\mathscr{P}) = Sol(B) \tag{3.10}$$

$$f(x^p) = v(p), \text{ for all } r-t \text{ paths } p \text{ in } B. \tag{3.11}$$

A longest $r-t$ path in B encodes an optimal solution x^* to $\mathscr{P}$ and its length $v^*(B)$ equals the optimal value $f(x^*)$ of $\mathscr{P}$. Analogously, the length of a shortest path in an exact DD for a minimization problem equals the optimal value of the problem.

Algorithm 1: `compile_exact_DD`

output : decision diagram (U,A,d)

1 $L_1 \leftarrow \{\hat{r}\}$
2 **for** $j = 1,\ldots,n$
3 $\quad L_{j+1} \leftarrow$ `compile_layer`(L_j)
4 $t \leftarrow$ `blend`(L_{n+1})
5 $L_{n+1} \leftarrow \{t\}$
6 **return** (U,A,d)

Given a DP model for $\mathscr{P}$, the compilation of an exact DD $\mathscr{P}$ is straightforward. A procedure for accomplishing this task is described in Algorithm 1. It is assumed that a valid DP model for $\mathscr{P}$ is an input of all algorithms in this chapter. The root node (state) $\hat{r}$ is added to the first layer. Each subsequent layer is built as described in Algorithm 2. Given a layer L_j, for each node $u \in L_j$ and each element $d \in D_j$, if the transition $t_j(u,d)$ is feasible, then a new node u' with state $t_j(u,d)$ is created (if it does not already exist) and added to layer L_{j+1}. Then an arc from u to u', with weight $h_j(u,d)$, is created. Lastly, the nodes in the last layer are blended into the single node t (lines 5-6 of Algorithm 1, which has no particular associated state.

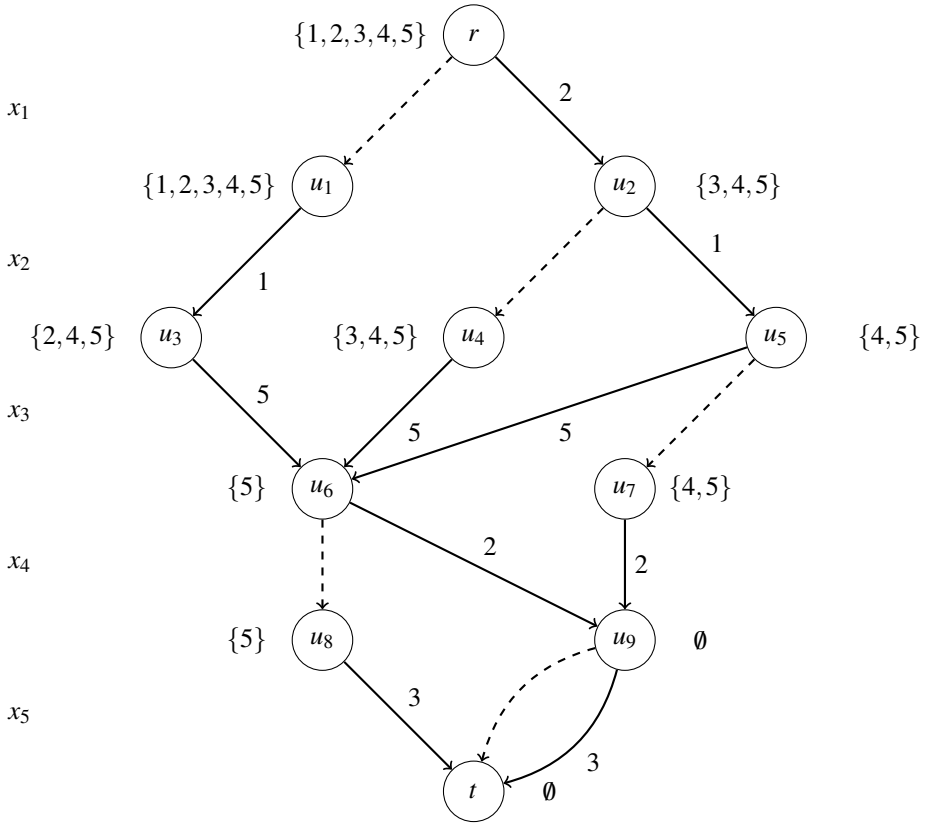
Computing a longest path in an exact DD can be done in polynomial time in the size of the diagram. However, compiling an exact DD for an NP-hard problem may not be a viable option, since the size of the DD can grow exponentially with the size of the problem. In the worst case scenario, a layer L_j in a BDD will contain 2^{j-1} nodes. This is expected, otherwise we would have P = NP. The size of a DD can be controlled by bounding its width, at the cost of no longer satisfying conditions (3.10) and (3.11). In Sections 3.5 and 3.6, we describe how this can be done when compiling so-called restricted and relaxed DDs, respectively.

Algorithm 2: `compile_layer`

input : layer L_j
output : layer L_{j+1}

$L_{j+1} \leftarrow \emptyset$
for $u \in L_j$
  for $i \in D_j : t_j(u,i) \neq \hat{0}$
    $u' \leftarrow t_j(u,i)$ `/* Create node */`
    $L_{j+1} \leftarrow L_{j+1} \cup \{u'\}$ `/* Add node to layer */`
    $b_i(u) \leftarrow u'$ `/* Create arc */`
    $v(a_i(u)) \leftarrow h_j(u,i)$ `/* Set arc weight */`
return L_{j+1}

Figure 2 – An exact DD



Source: The author.
Note: Solid arcs indicate 1-assignments, while dashed arcs indicate 0-assignments. Lengths of solid arcs are specified. Dashed arcs have length 0. The state for each node is also specified.

Example 8. *Consider an instance of the weighted set cover problem defined in Section 3.1 (Example 5), where* $E = \{1,2,3,4,5\}$, $\mathscr{F} = \{\{1,2\},\{1,3\},\{2,3,4\},\{4,5\},\{5\}\}$, *and associated*

costs are $c = (2,1,5,2,3)$. *Its formulation as an ILP problem is given by*

$$\begin{aligned}
\textit{minimize} \quad & 2x_1 + x_2 + 5x_3 + 2x_4 + 3x_5 && && (3.12)\\
\textit{subject to} \quad & x_1 + x_2 && \geq 1 && (3.13)\\
& x_1 \quad + x_3 && \geq 1 && (3.14)\\
& \quad x_2 \quad + x_3 && \geq 1 && (3.15)\\
& \quad\quad x_3 \quad + x_4 && \geq 1 && (3.16)\\
& \quad\quad\quad x_4 \quad + x_5 && \geq 1 && (3.17)
\end{aligned}$$

There are 11 feasible solutions for this instance. The optimal solution is $x^* = (1,1,0,1,0)$ *and the optimal value is 5. The BDD of Figure 2 is an exact DD for this instance obtained by executing Algorithm 1 using the DP model described in Section 3.3 (Example 7), and inverting the sign of the arc costs. Observe that every* $r-t$ *path in the BDD encodes a feasible solution and each feasible solution is encoded by a path. In particular, the shortest path* $r-u_2-u_5-u_7-u_9-t$ *(following the 0-arc from* u_9 *to* t*) encodes the optimal solution and its length equals the optimal value of 5.*

3.5 Restricted decision diagrams

B is a *restricted decision diagram* for $\mathscr{P}$ if the $r-t$ paths in B encode a subset of the feasible solutions of $\mathscr{P}$, and the length of any $r-t$ path is a lower bound on the objective function value for the solution encoded by the path. That is, B is restricted for $\mathscr{P}$ if

$$\begin{aligned}
Sol(\mathscr{P}) &\supseteq Sol(B) && (3.18)\\
f(x^p) &\geq v(p), \text{ for all } r-t \text{ paths } p \text{ in } B. && (3.19)
\end{aligned}$$

A longest $r-t$ path in B encodes a feasible solution $\bar{x}$ to $\mathscr{P}$ and its length $v^*(B)$ is a lower bound on the optimal value z^* for $\mathscr{P}$. Analogously, the length of a shortest path in a restricted DD for a minimization problem provides an upper bound on the optimal value of the problem.

Compiling a restricted DD for $\mathscr{P}$ is similar to compiling an exact DD, the difference being that the width of the DD is controlled so as to not exceed a maximum width parameter W (see lines 3 to 5 of Algorithm 3). After building a layer, a subset M of its nodes is heuristically selected and removed. This procedure is repeated until the width of the layer is at most W. Node selection (function `node_select_removal`) is discussed in Section 3.7.

Algorithm 3: `compile_restricted_DD`

input : node r, maximum width W
output : decision diagram (U,A,d)

$L_{\ell(r)} \leftarrow \{r\}$
for $j = \ell(r),\dots,n$
  while $|L_j| > W$
    $M \leftarrow$ `node_select_removal`(L_j)
    $L_j \leftarrow L_j \setminus M$
  $L_{j+1} \leftarrow$ `compile_layer`(L_j)
$t \leftarrow$ `blend`(L_{n+1})
$L_{n+1} \leftarrow \{t\}$
return (U,A,d)

A slight difference between the algorithms for compiling exact DDs and restricted DDs is that the compilation of a restricted DD does not necessarily start with the first layer (unless the root state of the DP model is passed to the algorithm as an input). This modification is done in order to accommodate the algorithm into a branch-and-bound scheme, where a restricted DD is not compiled for $\mathscr{P}$, but for a subproblem of $\mathscr{P}$. The use of restricted DDs in this context is described in Section 3.8.

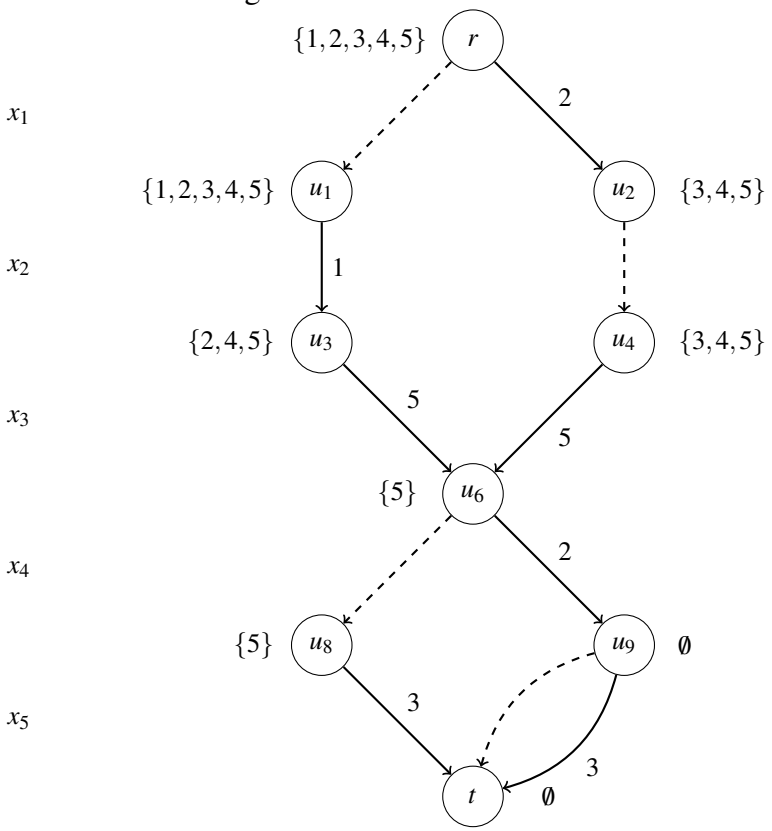
Observe that the DD obtained by executing Algorithm 3 for $\mathscr{P}$ is restricted. Indeed, the procedure of eliminating nodes only causes the loss of feasible solutions, thus condition (3.18) is satisfied. Furthermore, the length $v(p)$ of an $r-t$ path p equals the objective function value $f(x^p)$ for the solution encoded by p, thus condition (3.19) is also satisfied.

Example 9. *Consider the instance for the weighted set cover problem given in Section 3.4 (Example 8). The BDD of Figure 3 is a restricted DD of maximum width 2 for this instance. It is obtained by executing Algorithm 3 using the DP formulation given in Section 3.3 (Example 7), and inverting the sign of the arc costs. The BDD is similar to the exact DD of Figure 2, as it is obtained by starting the construction from the root state and selecting node u_5 to be eliminated when the third layer is compiled. The shortest path in the resulting BDD has length 8, which is an upper bound on the optimal value of 5.*

3.6 Relaxed decision diagrams and merging rules

B is a *relaxed decision diagram* for $\mathscr{P}$ if the $r-t$ paths in B encode a superset of the feasible solutions of $\mathscr{P}$, and the length of any $r-t$ path that encodes a feasible solution of $\mathscr{P}$ is an upper bound on the objective function value of the solution encoded by the path. That is, B is

Figure 3 – A restricted DD



Source: The author.
Note: Solid arcs indicate 1-assignments, while dashed arcs indicate 0-assignments. Lengths of solid arcs are specified. Dashed arcs have length 0. The state for each node is also specified.

relaxed for $\mathscr{P}$ if

$$Sol(\mathscr{P}) \subseteq Sol(B) \tag{3.20}$$

$$f(x^p) \leq v(p), \text{ for all } r-t \text{ paths } p \text{ in } B \text{ for which } x^p \in Sol(\mathscr{P}). \tag{3.21}$$

A longest $r-t$ path in B encodes a solution $\bar{x}$ of $\mathscr{P}$, which may or may not be feasible, and its length $v^*(B)$ is an upper bound on the optimal value z^* of $\mathscr{P}$. Analogously, the length of a shortest path in a relaxed DD for a minimization problem provides a lower bound on the optimal value of the problem. Naturally, if the solution encoded by an optimal path is feasible, and its value equals the length of the optimal path, then the solution is also optimal.

When compiling restricted DDs, selected nodes are eliminated when the width of a layer exceeds the maximum width parameter. When compiling relaxed DDs, however, nodes are merged, instead of eliminated. The merging procedure is defined by two operators: $\oplus$ and Γ_M. These operators are valid if applying Algorithm 4 to $\mathscr{P}$ results in a DD that satisfies conditions (3.20) and (3.21). Validity conditions are discussed in Section 3.6.1. The operator $\oplus$ is a function $\oplus : 2^{S_j} \rightarrow S_j$, for $j = 2, \ldots, n-1$, which takes a subset M of nodes (states) in layer L_j and returns

Algorithm 4: `compile_relaxed_DD`

input : node r, maximum width W
output : decision diagram (U, A, d)

1 $L_{\ell(r)} \leftarrow \{r\}$
2 **for** $j = \ell(r), \ldots, n$
3 **while** $|L_j| > W$
4 $M \leftarrow$ `node_select_merger`(L_j)
5 $u' \leftarrow \oplus(M)$
6 **for** $u \in L_{j-1}$
7 **for** $i \in D_{j-1} :\ b_i(u) \in M$
8 $v(a_i(u)) \leftarrow \Gamma_M(v(a_i(u)), b_i(u))$
9 $b_i(u) \leftarrow u'$
10 $L_j \leftarrow (L_j \setminus M) \cup \{u'\}$
11 $L_{j+1} \leftarrow$ `compile_layer`(L_j)
12 $t \leftarrow$ `blend`(L_{n+1})
13 $L_{n+1} \leftarrow \{t\}$
14 **return** (U, A, d)

a state $\oplus(M) \in S_j$. The operator Γ_M is a function $\Gamma_M : \mathbb{R} \times M \to \mathbb{R}$, which takes the length of an arc coming into a node $u \in M$ and returns a real value $\Gamma_M(v, u)$.

After building a layer L_j, a node merger operation is applied to L_j, which consists in heuristically selecting a set M of nodes in L_j to be merged into the single node $u' = \oplus(M)$. Each incoming arc at some node $u \in M$ is redirected to u', while its cost v is modified to $\Gamma_M(v, u)$. L_j is then updated with the removal of all nodes $u \in M$ from L_j and with the addition of u' to L_j. This procedure is repeated until the width of the layer is at most W (see lines 4 to 10 of Algorithm 4). Node selection (function `node_select_merger`) is discussed in Section 3.7.

We remark that there is another procedure in the literature for compiling relaxed DDs, which is based on node splitting, rather than node merging. The idea of the so-called *compilation by separation algorithm* is to start with a (trivial) relaxed DD and apply operations on the nodes called *filtering* and *refinement* (BERGMAN *et al.*, 2016). Filtering consists in removing arcs for which the corresponding transition functions lead to an infeasible state. Refinement consists of splitting nodes to strengthen the DD representation, as long as the size of the layer does not exceed the maximum width. The algorithm is derived from the *incremental refinement* algorithm proposed by Hadzic *et al.* (2008). As examples of the use of the algorithm we can mention, Hoda *et al.* (2010) applied the algorithm for constraint propagation in constraint programming, while Cire and Hoeve (2013) applied the procedure for sequencing problems.

3.6.1 Validity of relaxation operators

Hooker (2017) proved general conditions under which a node merger operation defined by $\oplus$ and Γ_M yields a valid relaxation. A state $u' \in L_j$ *relaxes* a state $u \in L_j$ if (i) all feasible transitions from state u are also feasible from state u', and (ii) the immediate cost of any feasible transition from u is at most that of its immediate cost from u', that is,

$$\begin{aligned} t_j(u,e) \neq \hat{0} \quad &\Longrightarrow \quad t_j(u',e) \neq \hat{0},\ \forall\, e \in D_j, \\ h_j(u,e) \quad &\leq \quad h_j(u',e),\ \forall\, e \in D_j. \end{aligned}$$

Two conditions must be satisfied in order for a node merger operation to yield a valid relaxation. The first condition establishes that when u' relaxes u, if the transition function is applied to both of them with the same control (and the resulting states are feasible), then the resulting state from u' relaxes the resulting state from u. Recasting this in terms of state $\oplus(M)$, we have

$$t_j(\oplus(M),e) \text{ relaxes } t_j(u,e),\ \ \forall\, u \in M,\ e \in D_j : t_j(u,e) \neq \hat{0}. \tag{3.22}$$

The second condition requires that the resulting state from the merger relaxes all the merged states, that is,

$$\oplus(M) \text{ relaxes } u,\ \forall\, u \in M. \tag{3.23}$$

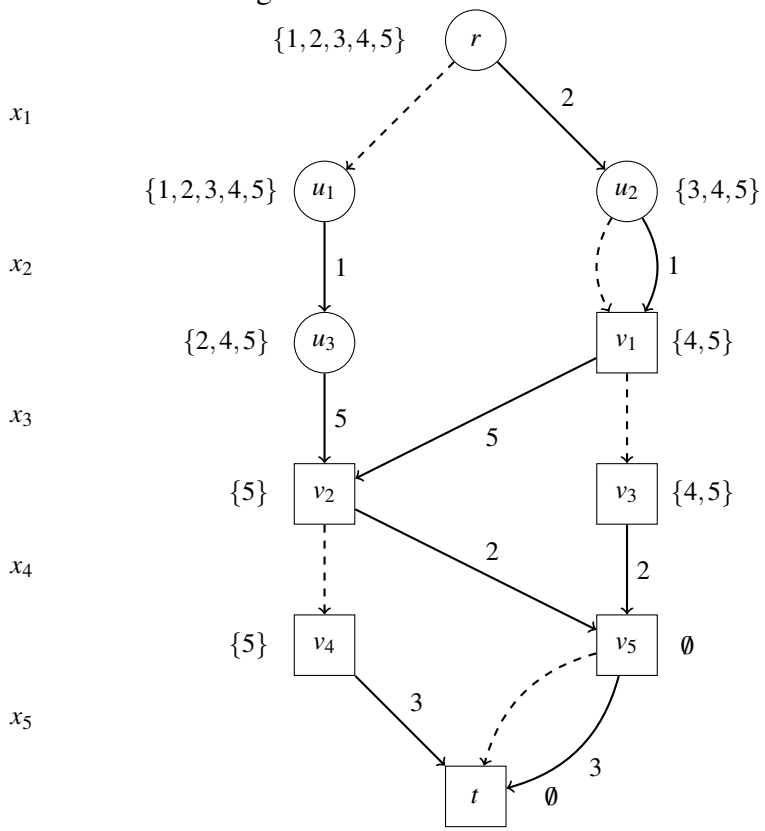
Theorem 3.6.1. *(HOOKER, 2017) If conditions (3.22) and (3.23) are satisfied, the merger of nodes in M within a decision diagram B results in a valid relaxation of B.*

Example 10. *A valid node merger operation for the DP formulation of the weighted set cover problem given in Section 3.3 (Example 7) is*

$$\oplus M = \bigcap_{u \in M} u \text{ and } \ \Gamma_M(v,u) = v.$$

The rationale for the node merger operation is to consider as covered some elements that are not actually covered and, consequently, introduce infeasible solutions without removing feasible ones. By taking the intersection of nodes in M, all those elements that were covered in some states but not in others are considered to be covered in the resulting state. Furthermore, there is no change to the costs of the incoming arcs to the resulting state. It can be verified that these operators satisfy the conditions of Theorem 3.6.1.

Figure 4 – A relaxed DD



Source: The author.

Note: Solid arcs indicate 1-assignments, while dashed arcs indicate 0-assignments. Lengths of solid arcs are specified. Dashed arcs have length 0. The state for each node is also specified. The distinct shapes of nodes will be discussed later in this chapter.

Consider the instance for the weighted set cover problem given in Section 3.4 (Example 8). The BDD of Figure 4 is a relaxed DD of maximum width 2 for that instance. It is obtained by executing Algorithm 4 using the DP formulation given in Section 3.3 (Example 7), and inverting the sign of the arc costs. The BDD is similar to the exact DD of Figure 2, as it is obtained by selecting nodes u_4 and u_5 to be merged when the third layer is compiled, resulting in the node v_1. The shortest path in the resulting BDD has length 4, which is a lower bound on the optimal value of 5. It encodes the infeasible solution $(1,0,0,1,0)$*, which was introduced in the BDD by the merge operation.*

3.7 Node selection for restricted and relaxed DDs

When compiling restricted and relaxed DDs, it is necessary to heuristically select a subset of nodes from a layer. The heuristic can be based on combinatorial properties of $\mathscr{P}$ or it can be a procedure that could be applied to any $\mathscr{P}$. In this Section, we present a case of the latter: a greedy heuristic, described in Algorithm 5, that selects nodes based on the longest path

lengths from the root node up to them. This strategy proved to be effective for the maximum independent set problem (BERGMAN *et al.*, 2013).

Algorithm 5: `node_select`

input : layer L_j, maximum width W
output : set of nodes M

$M \leftarrow \emptyset$
while $|L_j \setminus M| > W$
  Let $\hat{u} \in \arg\min\{v^*(B_{ru}) : u \in L_j \setminus M\}$
  $M \leftarrow M \cup \{\hat{u}\}$
return M

The longest $r-t$ path in a restricted DD provides a lower bound on the optimal value for $\mathscr{P}$, thus it is natural to attempt to build a restricted DD that provides a high lower bound. Therefore, selecting nodes with small values of $v^*(B_{ru})$ to be eliminated is a reasonable greedy heuristic for maximizing the value of the longest path in the resulting restricted DD.

The longest $r-t$ path in a relaxed DD provides an upper bound on the optimal value for $\mathscr{P}$, thus it is natural to attempt to build a relaxed DD that provides a low upper bound. Owing to (3.21), a longest path in a relaxed DD is likely to contain a node that was introduced by the merging operation. Therefore, selecting the nodes u with the smallest value $v^*(B_{ru})$ to be merged is a good greedy heuristic for minimizing the value of the longest path in the resulting relaxed DD.

Example 11. *Observe that the choices made for node selection in Sections 3.5 and 3.6 (Examples 9 and 10) did not yield the tightest possible bounds, despite the fact that the heuristic described in Algorithm 5 was used. Selecting node u_3 or node u_4 (instead of u_5) to be eliminated when compiling the restricted DD, would yield an upper bound of 5, which equals the optimal value. Similarly, selecting nodes u_3 and u_4 (instead of u_4 and u_5) to be merged when compiling the relaxed DD, would yield a lower bound of 5, which equals the optimal value. Thus, since the lower bound equals the upper bound, 5 is the optimal value for the problem and the solution encoded by the optimal path in the restricted DD is optimal for the problem.*

3.8 A decision diagram-based branch-and-bound

The idea of a DD-based branch-and-bound (BAB) comes from the fact that each state in a DP formulation for a problem represents a subproblem of the original problem $\mathscr{P}$. Thus,

each node in the corresponding exact DD also represents a subproblem. Naturally, divide-and-conquer ideas could in turn be used for trying to solve the problem by solving the subproblems represented by the nodes. In a DD-based BAB, that means branching on the nodes. Like in a traditional BAB in integer linear programming (ILP), an implicit enumeration of all feasible solutions is attempted by computing bounds on the optimal value of the subproblems and pruning unpromising nodes.

In an LP-based BAB, linear relaxations of the subproblems provide dual bounds on the value of the objective function, while branching is accomplished by generating subproblems via the addition of linear inequalities that eliminate fractional solutions. Relaxed DDs play the role of linear relaxations in a DD-based BAB by providing bounds and a direction for branching. In this case, branching is achieved by selecting so-called exact nodes in the relaxed DD. Furthermore, in an LP-based BAB, when the solution of the linear relaxation is not integer, primal heuristics can be used in order to generate primal bounds. This role is played by restricted DDs in a DD-based BAB.

Let $\mathscr{P}|_u$ be a restricted version of $\mathscr{P}$, whose feasible solutions are encoded by the $r-t$ paths that contain node u in an exact DD for $\mathscr{P}$. Let $\overline{B}$ be a relaxed DD for $\mathscr{P}$. A node u of $\overline{B}$ is *exact* if all $r-u$ paths in $\overline{B}$ lead to the same state s^j. In other words, if the application of the transition functions over the partial solution encoded by any $r-u$ path leads to the same state. An exact node u of $\overline{B}$ represents the subproblem $\mathscr{P}|_u$. A *cut set* of $\overline{B}$ is a subset S of nodes of $\overline{B}$ such that any $r-t$ path in $\overline{B}$ contains at least one node in S. A cut set is *exact* if all nodes in S are exact. Note that every feasible solution for $\mathscr{P}$ is encoded by a path that contains at least one node from any given exact cut set S. Thus, the nodes in an exact cut set encode a partition of the set of feasible solutions of $\mathscr{P}$. Therefore, an exact cut set of $\overline{B}$ provides an exhaustive enumeration of subproblems for $\mathscr{P}$:

Theorem 3.8.1. *(BERGMAN et al., 2016) Let S be an exact cut set of a relaxed DD $\overline{B}$ for $\mathscr{P}$ compiled by Algorithm* 4 *using a valid DP model. Then*

$$z^*(\mathscr{P}) = \max_{u \in S} \{z^*(\mathscr{P}|_u)\}.$$

A DD-based branch-and-bound implicitly compiles an exact DD B^* for $\mathscr{P}$ by attempting to avoid the exploration of paths that do not lead to an optimal solution for $\mathscr{P}$. A description of such a procedure is shown in Algorithm 6. The algorithm begins by compiling a restricted and a relaxed DD with a root node (state) $\hat{r}$ for $\mathscr{P}$. The relaxed DD can provide an

exact cut set, which corresponds to a decomposition of $\mathscr{P}$ into a series of subproblems. Each of these subproblems is then processed in the same fashion.

Algorithm 6: `DD_based_BAB`

```
   input  : maximum width W
   output : optimal value z_opt
1  Q ← {(r̂, 0)}
2  z_opt ← −∞
3  while Q ≠ ∅
4      (u, v_u) ← select_node_branching(Q)
5      Q ← Q \ {(u, v_u)}
6      B̂ ← compile_restricted_DD(u, W)
7      z_lb ← v_u + v*(B̂)
8      z_opt ← max{z_opt, z_lb}
9      if B̂ is not exact then
10         B̄ ← compile_relaxed_DD(u, W)
11         z_ub ← v_u + v*(B̄)
12         if z_ub > z_opt then
13             S ← exact_cut set(B̄)
14             for w ∈ S
15                 v_w ← v_u + v*(B̄_uw)
16                 Q ← Q ∪ {(w, v_w)}
17 return z_opt
```

At each iteration, a node u, which is associated with a subproblem $\mathscr{P}|_u$, is selected from a queue Q (see line 4 of Algorithm 6). Note that its value $v_u = v^*(B^*_{ru})$ corresponds to the longest $r-u$ path in the implicitly constructed exact DD B^* for $\mathscr{P}$. A restricted DD $\widehat{B}$ with maximum width W and u as the root node is compiled (see line 6 of Algorithm 6). If the length of a longest path in $\widehat{B}$ added to the value of node u provides a higher lower bound value $z_{lb} = v_u + v^*(\widehat{B})$ than that of the best incumbent solution, z_{opt}, then z_{opt} is updated to z_{lb}. If $\widehat{B}$ is exact, then u is *pruned by optimality*, since z_{lb} is the optimal value for $\mathscr{P}|_u$. Otherwise, a relaxed DD $\overline{B}$ with maximum width W and root node u is compiled (see line 10 of Algorithm 6). If the length of a longest path in $\overline{B}$ added to the value of node u provides an upper bound $z_{ub} = v_u + v^*(\overline{B})$ inferior to that of the best incumbent solution, z_{opt}, then u is *pruned by bound*, since $\mathscr{P}|_u$ does not contain a solution with objective function value higher than z_{ub}. Otherwise, a cut set S of exact nodes in $\overline{B}$ is selected and added to Q together with their corresponding values (see lines 13 to 16 of Algorithm 6).

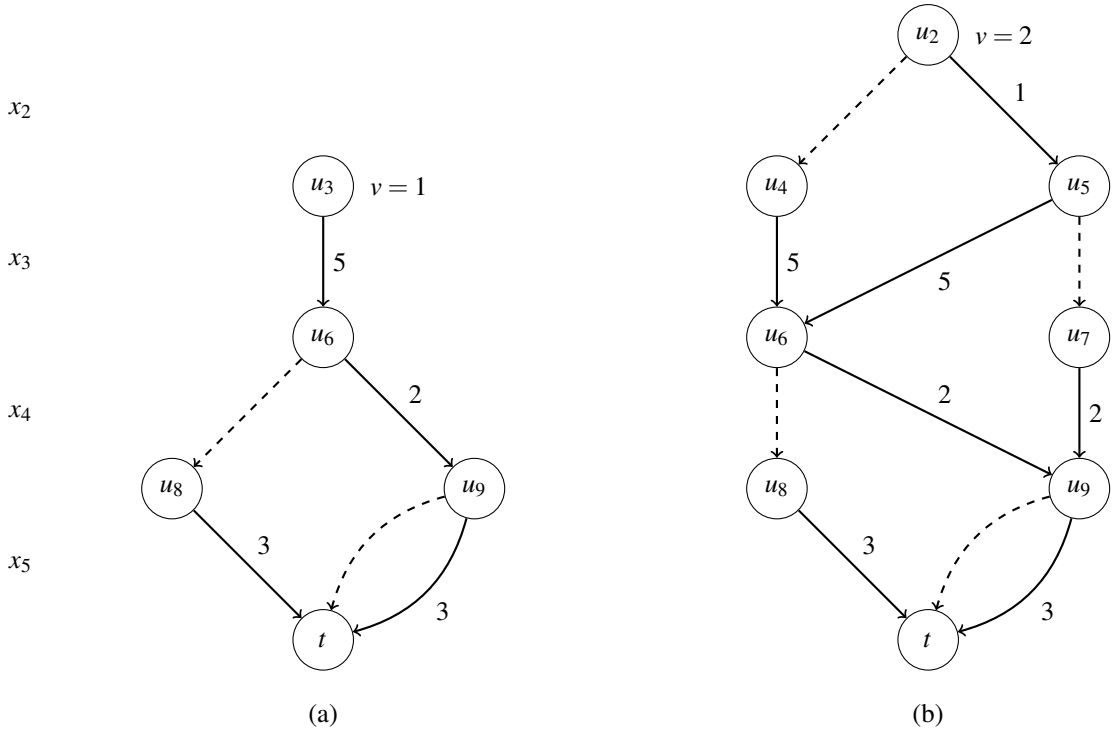
3.8.1 Search node selection and types of exact cut sets

In an implementation of a DD-based BAB the user has to choose a node to be branched (see line 4 of Algorithm 6) and has to select nodes to be part of an exact cut set (see line 13 of Algorithm 6). Bergman *et al.* (2016) selected nodes u in Q with smallest value v_u for maximization problems. The authors also described three different types of cut sets: *traditional branching* (TB), *last exact layer* (LEL), and *frontier cut set* (FC):

- TB: Branching is performed on the nodes from the second layer, i.e., $S = L_2$.
- LEL: Branching is performed on the nodes of the deepest layer containing only exact nodes, i.e., $S = L_{j^*}$, with $j^* = \max\{j : u \text{ is exact } \forall u \in L_j\}$.
- FC: Branching is performed on the exact nodes that are immediate predecessors of nodes that are not exact, i.e., $S = \{u \in U(\overline{B}) : u \text{ is exact and } b_d(u) \text{ is not exact for some } d \in D_{\ell(u)}\}$.

Note that these types of exact cut sets are minimal (with respect to inclusion), so that, for any feasible solution of $\mathscr{P}$, its corresponding path contains exactly one node from S.

Figure 5 – DDs for subproblems in a BAB



Source: The author.

Note: Solid arcs indicate 1-assignments, while dashed arcs indicate 0-assignments. Lengths of solid arcs are specified. Dashed arcs have length 0. The values of the root nodes are specified.

Example 12. *Again, consider the instance for the weighted set cover problem given in Section*

3.4 (Example 8). Suppose the maximum width W is set to 2. Suppose that the first iteration of an execution of the BAB described in Algorithm 6, using the DP formulation given in Section 3.3 (Example 7) and inverting the sign of the arc costs, yielded the restricted and relaxed DDs of Figures 3 and 4, respectively. Thus, at iteration 1, a lower bound bound of 4 and an upper bound of 8 is obtained. z_{opt} is set to the upper bound of 8.

Observe at Figure 4, that circled nodes are exact, and squared nodes are not. The second layer $\{u_1, u_2\}$ corresponds to the last exact layer, while $\{u_2, u_3\}$ corresponds to the frontier cut set. Suppose the FC is chosen to be added to Q in the first iteration of the BAB, i.e., the pairs $(u_2, 2)$ and $(u_3, 1)$ are added to Q.

Suppose that at iteration 2, node u_3 is chosen for branching. The resulting restricted DD is shown in Figure 5(a). The shortest path in it has length 7, which provides a lower bound of 8 by adding the node value of 1. The best current solution was already 8. Thus, z_{opt} is not updated. And since the DD is exact, the node is pruned by optimality and consequently, there is no need to compile a relaxed DD.

Now at iteration 3, node u_2 is chosen for branching. The resulting restricted DD is shown in Figure 5(b). The shortest path in it has length 3, which provides a lower bound of 5 by adding the node value of 2. The best current solution is 8. Thus, z_{opt} is updated from 8 to 5. And since the DD is exact, node u_2 is pruned by optimality and consequently, there is no need to compile a relaxed DD. With the queue Q empty, the search is finished and the optimal value is 5.

4 FINDING MAXIMUM PATTERNS USING DECISION DIAGRAMS

In this chapter, we propose a way of solving the α-MPP and the MPP using a DD-based branch-and-bound. In Sections 4.1 and 4.2, we propose a formulation and a dynamic programming model for the MPP. In Section 4.3, we explore some properties of the resulting exact DD for the MPP. In Section 4.4, we adapt our proposed DP model to the α-MPP case. In Sections 4.5 and 4.6, we describe a valid merging rule and a node selection rule for our model. In Section 4.7, we propose a variable ordering heuristic. In Section 4.8, we introduce a modification to our original model in order to improve the performance of the proposed DD-based branch-and-bound algorithm.

4.1 A coverage-based formulation for the MPP

Recall that a term T covers a set of observations $\text{Cov}(T)$. However, there may be multiple terms that cover exactly the same observations. We can try to solve the MPP in two ways: by finding a term of maximum coverage, or simply by finding a set of covered observations of maximum cardinality. In the latter case, given an optimal set C^* of covered observations, we can easily find the spanned pattern P^* that defines C^*. Observe that $\text{Lit}(P^*)$ is composed by all literals $\ell \in \mathscr{L}$ such that $\ell(x) = 1,\ \forall x \in C^*$. Thus, a formulation for the MPP based on coverage has a smaller solution space than a formulation based on terms, since each solution of the former might encapsulate multiple solutions of the latter.

Without loss of generality, let the set of positive observations be indexed from 1 to m^+, i.e., $\Omega^+ = \{x^1, \ldots, x^{m^+}\}$. Let z_j be binary decision variables, for $j = 1, \ldots, m^+$, each of which indicates whether observation $x^j \in \Omega^+$ is covered or not:

$$z_j = \begin{cases} 1, & \text{if } x^j \text{ is covered;} \\ 0, & \text{otherwise.} \end{cases}$$

Consider an instantiation $\bar{z} \in \{0,1\}^{m^+}$ of variables z. We define as $\Omega^+(\bar{z}) = \{x^j \in \Omega^+ : \overline{z_j} = 1\}$ as the set of positive observations specified by $\bar{z}$. We also define $\text{Span}(\bar{z}) = \{\ell \in \mathscr{L} : \ell(x) = 1,\ \forall x \in \Omega^+(\bar{z})\}$ as the set of all literals that cover every observation in $\Omega^+(\bar{z})$. The term $T = \prod_{\ell \in \text{Span}(\bar{z})} \ell$

is precisely the spanned pattern that covers all observations in $\Omega^+(\bar{z})$. We formulate the MPP as

$$\text{maximize} \quad \sum_{j=1}^{m^+} z_j \tag{4.1}$$

$$\text{subject to} \quad \prod_{\ell \in \text{Span}(z)} \ell(\gamma) = 0, \quad \gamma \in \Omega^- \tag{4.2}$$

$$z \in \{0,1\}^{m^+}. \tag{4.3}$$

Objective function (4.1) equals the number of observations covered and is to be maximized, while constraints (4.2) ensure that no negative observation is covered by the spanned pattern derived from the set of covered observations.

4.2 Dynamic programming formulation for the MPP

Let S_j be state spaces, for $j = 1,\ldots,m^+ + 1$. Let the root state $\hat{r}$ be the set $\mathscr{L}$ of all literals. A state variable $s^j \in S_j$ represents a subset of literals:

$$S_1 = \{\mathscr{L}\} \text{ and } S_j = 2^{\mathscr{L}} \cup \{\hat{0}\},\ j = 2,\ldots,m^+ + 1.$$

The rationale in our model is to start from a position where no observation is covered. By defining the root state as the set of all literals, we represent a term that covers no observation. If we decide to cover a given observation, then we have to remove from the current state all those literals that do not agree with it. If the term composed by the literals from the resulting state covers some negative observation, the corresponding transition is deemed infeasible. Let the transition functions $t_j : S_j \times \{0,1\} \to S_{j+1}$, for $j = 1,\ldots,m^+$, be defined as

$$t_j(s^j, z_j) = \begin{cases} s^j, & \text{if } z_j = 0; \\ \{\ell \in s^j : \ell(x^j) = 1\}, & \text{if } z_j = 1 \text{ and } \nexists\, \gamma \in \Omega^- : \prod\limits_{\substack{\ell \in s^j: \\ \ell(x^j)=1}} \ell(\gamma) = 1; \\ \hat{0}, & \text{otherwise.} \end{cases}$$

Given a state s^j, a 0-transition always leads to the same state s^j. A 1-transition is feasible when the term t defined by the literals $\ell \in s^j$ which are not in conflict with x^j covers no negative observation γ. Otherwise, the 1-transition is infeasible. The cost functions $h_j : S_j \times \{0,1\} \to \mathbb{Z}_+$, for $j = 1,\ldots,m^+$, are straightforward:

$$h_j(s^j, z_j) = z_j.$$

Since the objective function of the problem corresponds to the cardinality of the coverage set and the decision variables are coverage-based, it is natural that a 0-transition has cost 0 and a 1-transition has cost 1.

Example 13. *Consider the instance of the MPP given by Table 1 and an exact DD compiled by executing Algorithm 1 with the DP formulation given in this section. Figure 6 shows the graph of a partially constructed exact DD just before blending the nodes in the last layer into a single node (line 5 of Algorithm 1). Observe that there are 3 longest paths of length 3, corresponding to the three optimal solutions* $z^1 = (0,0,1,1,1)$, $z^2 = (1,1,1,0,0)$, *and* $z^3 = (1,0,1,1,0)$. *The node at the end of the path that encodes* z^1 *is associated with the spanned pattern* $u_1u_2u_3$. *This path implicitly encodes the non-spanned patterns* u_2, u_1u_2, *and* u_2u_3, *which are also optimal. Similarly, the node at the end of the path that encodes* z^2 *is associated with the spanned pattern* u_4u_5, *and the node at the end of the path that encodes* z^3 *is associated with the spanned pattern* $u_1u_3u_5$ *and the non-spanned pattern* u_1u_5.

4.2.1 Validity

Let $\overline{z} \in \{0,1\}^{m^+}$ be a solution (not necessarily feasible) for the MPP formulation given in Section 4.1. Let $\overline{s} \in S_1 \times \cdots \times S_{m^+ +1}$ be such that $\overline{s^1} = \mathscr{L}$ and $\overline{s^{j+1}} = t_j(\overline{s^j}, \overline{z}_j),\ j = 1,\ldots,m^+$. It is clear that $(\overline{z},\overline{s})$ is feasible for the DP formulation given in Section 4.2, and that $\overline{s^{m^+ +1}} = \hat{t}_k$, for some $\hat{t}_k \in 2^{\mathscr{L}}$, or $\overline{s^{m^+ +1}} = \hat{0}$. Observe that if $\overline{z}$ is feasible, then $\hat{t}_k = \text{Span}(\overline{z})$. Also, we have

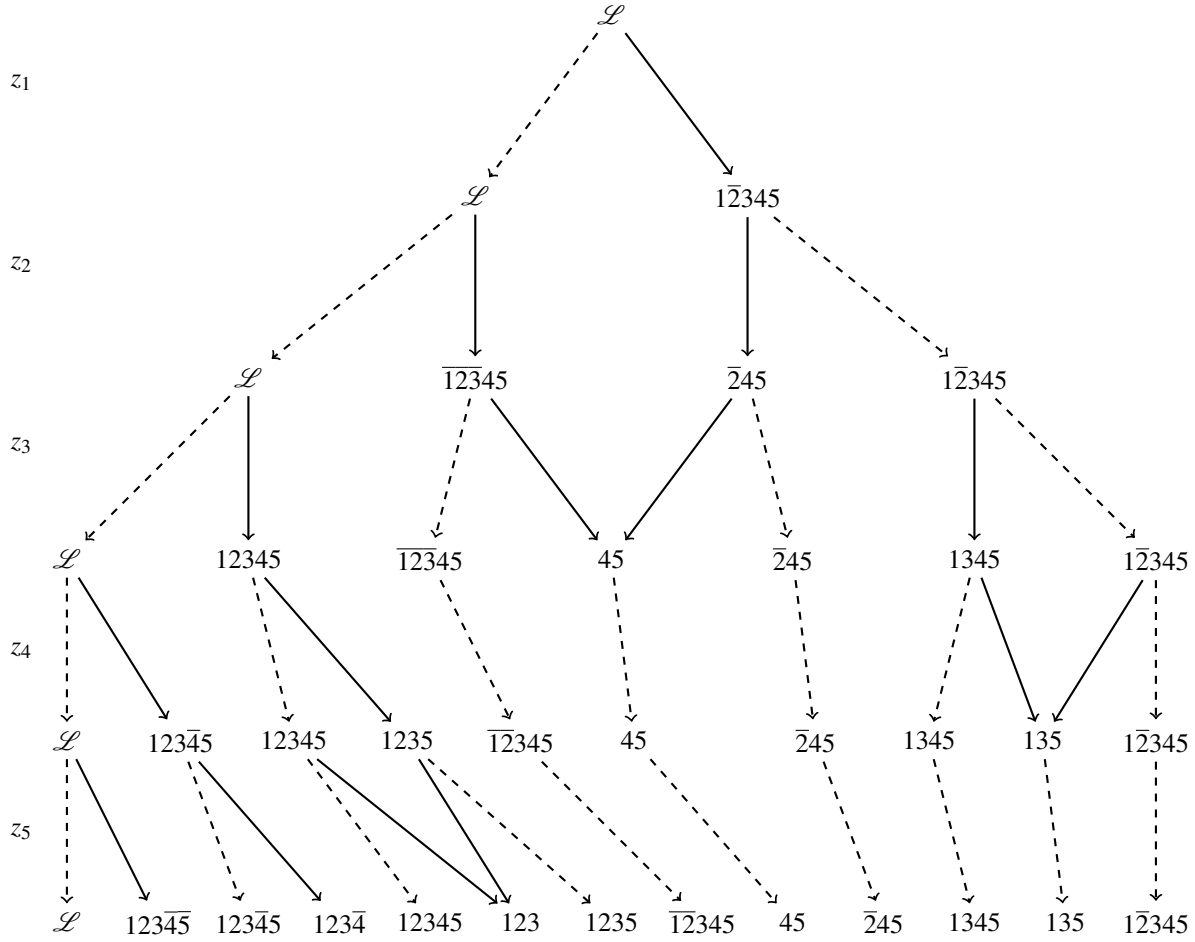
$$\sum_{j=1}^{m^+} h_j(\overline{s^j}, \overline{z_j}) = \sum_{j=1}^{m^+} \overline{z_j},$$

thus condition (3.8) is satisfied. If $\overline{z}$ is infeasible, then there exists $\gamma \in \Omega^-$, such that

$$\prod_{\ell \in \text{Span}(\overline{z})} \ell(\gamma) = 1.$$

By definition of the transition functions of the DP model, there exists some k, such that state $\overline{s^k}$ is infeasible, i.e., $\overline{s^k} = \hat{0}$. Therefore, $t_j(\overline{s^j}, \overline{z_j}) = \hat{0}$, for $j = k, ..., m^+$. Thus, condition (3.9) is also satisfied.

Figure 6 – A partially constructed exact DD for the MPP



Source: The author.

Note: Solid arcs encode 1-assignments and have length 1. Dashed arcs encode 0-assignments and have length 0. The state associated with each node is compactly represented as indexes, possibly with an overbar, each of which corresponds to a literal. For instance, $\overline{2}45$ stands for the set of literals $\{\overline{u_2}, u_4, u_5\}$.

4.3 Properties

Consider a partially constructed exact DD (before merging the nodes of the last layer) for the MPP compiled by executing Algorithm 1 with the DP formulation given in Section 4.2. By definition, transitions that lead to terms which are not patterns are deemed infeasible. Thus, with the exception of the nodes associated with state $\mathscr{L}$, every other node in an exact DD for the MPP defines a term which is a pattern (observe Figure 6). Consider the terminal nodes at the end of longest paths in the DD. We shall refer to them as optimal terminal nodes. These nodes are associated with terms that are strong spanned patterns.

Definition 4.3.1. *Let $r - \hat{t}_k^*$ be a longest path in an exact DD for the MPP compiled by executing Algorithm* 1 *with the DP formulation given in Section 4.2, without merging the nodes of the last layer. We refer to $\hat{t}_k^*$ as an* optimal terminal node*.*

Lemma 4.3.1. *An optimal terminal node $\hat{t}_k^*$ defines a strong pattern.*

Proof. Suppose $\hat{t}_k^*$ defines pattern P_k that is not strong. Then, there is a terminal node $\hat{t}_i$ that defines a pattern P_i, such that $\text{Cov}(P_k) \subset \text{Cov}(P_i)$. Thus, the length L_i of a longest $r-\hat{t}_i$ path is higher than the length L_k of a longest $r-\hat{t}_k^*$ path, i.e., $L_i > L_k$. But $\hat{t}_k^*$ is optimal. Absurd. □

Lemma 4.3.2. *An optimal terminal node $\hat{t}_k^*$ defines a spanned pattern.*

Proof. Let P_k be the pattern associated with state $\hat{t}_k^*$. Suppose that P_k is not spanned. Then, there is a literal ℓ that does not belong to $\hat{t}_k^*$, such that $\ell(x) = 1,\ \forall x \in \text{Cov}(P_k)$. Since $\ell \notin \hat{t}_k^*$, then ℓ was eliminated at some point during construction of the longest $r-\hat{t}_k^*$ path in the DD. Recall from the definition of the transition functions that only 1-transitions can account for the removal of literals from a state. Suppose that ℓ was eliminated in the j-th 1-transition, i.e., $\ell \in s^j$ and $\ell \notin t_j(s^j, 1)$. Then, we have $\ell(x^j) = 0$. A contradiction, since $x^j \in \text{Cov}(P_k)$. □

Theorem 4.3.1. *An optimal terminal node defines a strong spanned pattern.*

4.4 Adaptation to the α-MPP

The DP formulation for the MPP given in Section 4.2 can be easily adapted to the α-MPP. Without loss of generality, let α be the last positive observation in the dataset, i.e., $\alpha = x^{m^+}$. Since α must be covered by any α-pattern, we do not need to define variable z_{m^+}. Thus, the adapted DP formulation contains variables z_j, for $j = 1, \ldots, m^+ - 1$. Therefore, the resulting DD contains m^+ layers. Furthermore, we know that all literals that do not agree with α do not take part in an optimal α-pattern, therefore it is sufficient to define the root state as the set of all literals that agree with α, i.e., $\hat{r} = \{\ell \in \mathscr{L} : \ell(\alpha) = 1\}$. The ideas in the remainder of this chapter are described in terms of our DD formulation for the MPP, but they apply equally to the α-MPP formulation introduced in this section.

Example 14. *Consider the instance of the MPP given by Table 1. If the observation α is $x^2 = (0,0,0,1,1)$, then the root node of the DP model for the α-MPP is $\{\overline{u_1}, \overline{u_2}, \overline{u_3}, u_4, u_5\}$.*

4.5 Merging rule

According to Theorem 3.6.1, in order to define a valid node merger operators, conditions (3.22) and (3.23) must be satisfied. We shall define the following operators

$$\oplus(M) = \bigcup_{u \in M} u \quad \text{and} \tag{4.4}$$

$$\Gamma_M(v,u) = v. \tag{4.5}$$

Lemma 4.5.1. *$\oplus(M)$ relaxes w, $\forall w \in M$.*

Proof. 0-transitions are always feasible, therefore we only need to examine 1-transitions. Observe that

$$t_j(w,1) \neq \hat{0} \implies \prod_{\ell \in t_j(w,1)} \ell(\gamma) = 0,\ \forall \gamma \in \Omega^-.$$

Since $\ell(x^j) = 1,\ \forall \ell \in t_j(w,1)$ and $w \subseteq \oplus(M)$, then each literal in $t_j(w,1)$ also belongs to $t_j(\oplus(M),1)$, i.e., $t_j(w,1) \subseteq t_j(\oplus(M),1)$. Therefore, we have

$$\prod_{\ell \in t_j(\oplus(M),1)} \ell(\gamma) = 0,\ \forall \gamma \in \Omega^-.$$

Thus, any feasible transition from state w is also feasible from state $\oplus(M)$. Moreover, the transition costs are unchanged by Γ. □

Theorem 4.5.1. *Operators $\oplus$ and Γ as defined in (4.4) and (4.5) are valid relaxation operators for the MPP.*

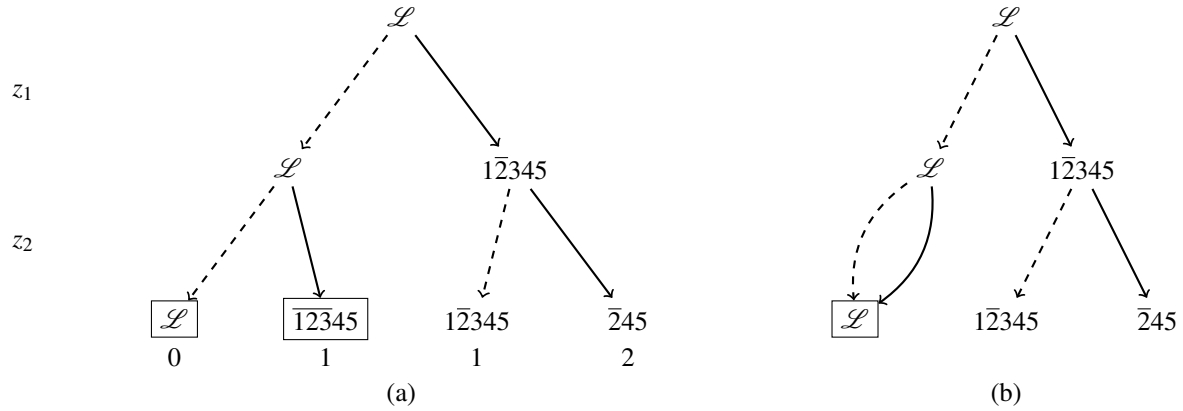
Proof. Since $\oplus(M)$ relaxes w, $\forall w \in M$, we need to show that $t_j(\oplus(M),1)$ relaxes $t_j(w,1)$. The proof is analogous to the proof of Lemma 4.5.1. □

4.6 Node selection

We use the node selection heuristic presented in Section 3.7 (Algorithm 5) when compiling restricted and relaxed DDs for the MPP, that is, in order to select nodes we rank them into a non-decreasing order of the value of the longest path from the root node up to each of them.

Example 15. *Consider the instance of the MPP given by Table 1 and a relaxed DD compiled by executing Algorithm 4 with the DP formulation given in Section 4.2. Figure 7 shows an iteration*

Figure 7 – Construction of a relaxed DD for the MPP with maximum width 3



Source: The author.
Note: Solid arcs indicate 1-assignments, while dashed arcs indicate 0-assignments. Solid arcs have length 1. Dashed arcs have length 0.

of the execution of the algorithm for compiling the relaxed DD. In Figure 7(a), the third layer exceeds the maximum width 3, thus we select 2 nodes to be merged. Node $\mathscr{L}$ is the highest ranked node to be selected, since the longest path from the root node up to it is 0. Nodes $\overline{12\bar{3}}45$ and $1\bar{2}345$ are tied as the second highest ranked nodes, since the longest path from the root node up to each of them is 1. We only need to select one of them. We arbitrarily choose $\overline{12\bar{3}}45$. Figure 7(b) shows the result of the merger, the union of the 2 merged nodes is $\mathscr{L}$.

4.7 Variable ordering

We propose a variable ordering that takes into account how similar a positive observation is from some negative observation. The intuition is that prioritizing variables associated with positive observations that are the most similar to some negative observation leads to infeasible states earlier throughout the construction of a DD, as it is very likely that after a subsequent 1-transition the remaining literals in the resulting state define a term that covers a negative observation. Consider the following metric $dist : \Omega^+ \to \{0,1,\ldots,n\}$ for measuring the dissimilarity between a positive observation and the set of negative observations:

$$dist(\beta) = \min_{\gamma \in \Omega^-} \{|\{j \in \{1,...,n\} : \beta_j \neq \gamma_j\}|\},\ \beta \in \Omega^+.$$

For a given positive observation β, we consider the value for ranking variable z_β to be the dissimilarity value between β and the negative observation which is least dissimilar (or more similar) to β. If given two observations x^i and x^k, with $i < k$, we have $dist(x^i) = dist(x^k)$, then we opt for the least index ranking, that is, x^i before x^k.

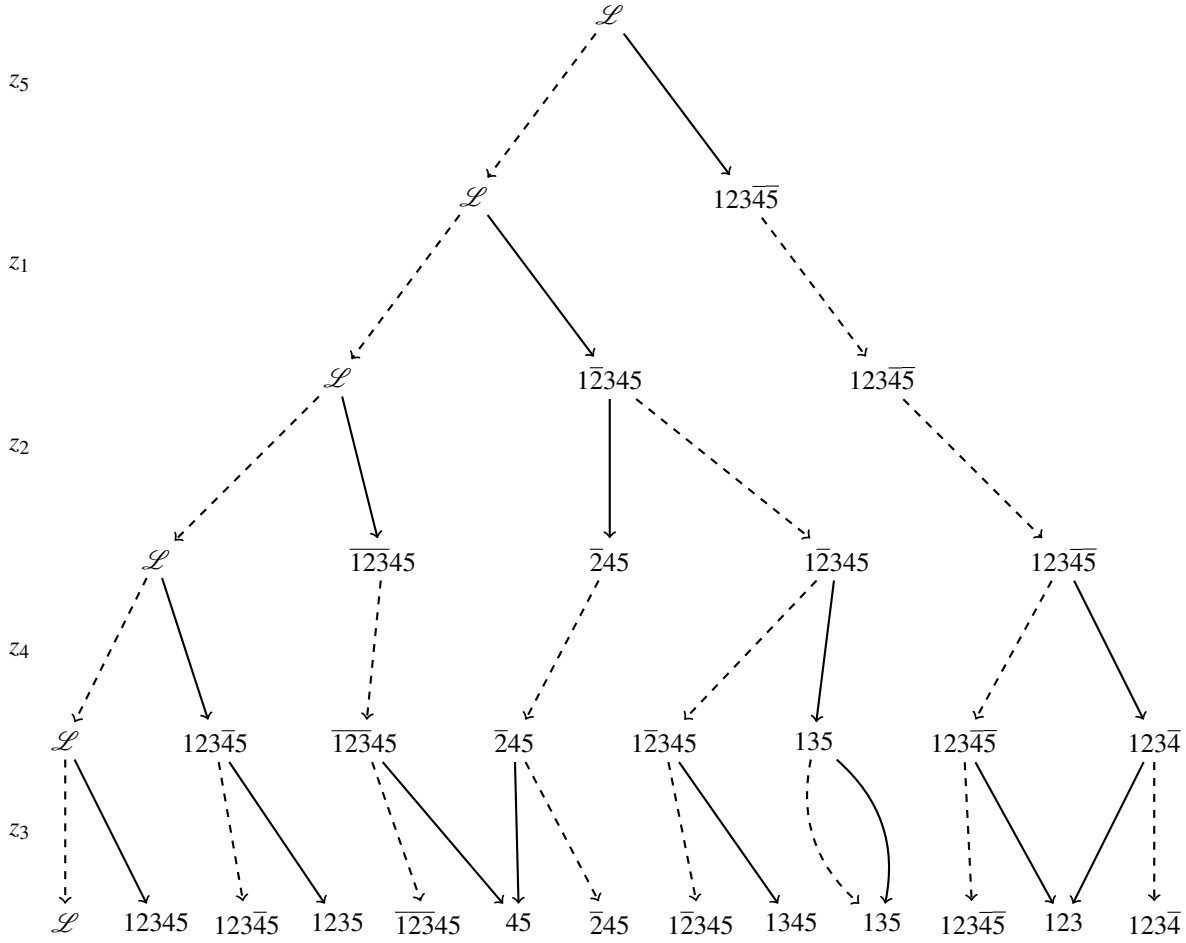
Example 16. *Consider the instance of the MPP given by Table 1. The distance $dist(\beta)$ of each*

Table 4 – Dissimilarity matrix

		Ω^-					
		x^6	x^7	x^8	x^9	x^{10}	$dist(x^k)$
Ω^+	x^1	2	2	2	3	3	2
	x^2	2	2	4	3	3	2
	x^3	3	3	3	4	4	3
	x^4	4	2	2	3	3	2
	x^5	3	3	1	2	2	1

Source: The author.

Figure 8 – A partially constructed exact DD for the MPP with different variable ordering



Source: The author.
Note: Solid arcs encode 1-assignments and have length 1. Dashed arcs encode 0-assignments and have length 0.

positive observation β to the negative set is given in Table 4. The resulting variable ordering is x^5, x^1, x^2, x^4, x^3, which yields the exact DD of Figure 8. Comparing it with the DD of Figure 6, we observe the original DD had 4, 7, and 10 nodes in the third, fourth, and fifth layers, respectively, while the DD with variable ordering had 3, 5, and 8 nodes in the third, fourth, and fifth layers, respectively.

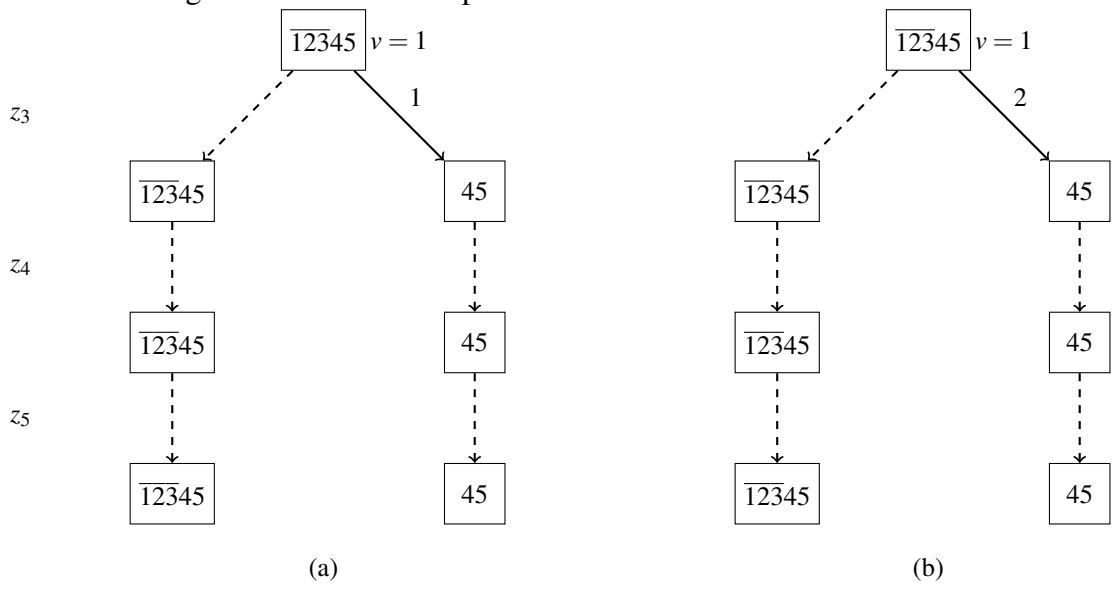
4.8 Improving bounds obtained from restricted DDs

We propose a modification of the cost function of our DP model in order to improve the performance of the branch-and-bound algorithm. Recall that when a choice is made to cover an observation x^j, all literals that do not agree with x^j are removed from the state. Formally, a 1-transition from a state s^j may yield a state $t_j(s^j, 1)$ that defines a term that covers not only x^j, but also some other observation x^k, which was not covered by s^j. Thus, we shall define the following cost function for restricted DDs only:

$$h_j(s^j, 1) = \left| \left\{ x \in \Omega^+ \setminus \mathrm{Cov}(s^j) : \prod_{\substack{\ell \in s^j: \\ \ell(x^j)=1}} \ell(x) = 1 \right\} \right|.$$

Computational experiments showed that this modified cost function enables the branch-and-bound algorithm to find better bounds earlier in the search than it would if the original unitary cost function was used. A possible interpretation of this behavior is that the structure of the DD compiled with this DP model allows the algorithm to take shortcuts to paths that encode better solutions. The following example illustrates the idea:

Figure 9 – DDs for subproblems in a BAB with modified costs



Source: The author.
Note: Solid arcs indicate 1-assignments, while dashed arcs indicate 0-assignments. Lengths of solid arcs are specified. Dashed arcs have length 0. The values of the root nodes are specified.

Example 17. *Consider the instance of the MPP given by Table 1 and an execution of the DD-based branch-and-bound algorithm presented in Section 3.8 (Algorithm 6) with the DP*

formulation given in Section 4.2. Suppose that in some iteration of the branch-and-bound the node $\overline{123}45$ *with cost 1 in the third layer is selected for branching (see Figure 6). The bound provided by the corresponding subproblem is 2 (see Figure 9(a)), accounting for the sum of the node value of 1 with the length of the longest path, which equals 1. A 1-transition from state* $\{\overline{u_1}, \overline{u_2}, \overline{u_3}, u_4, u_5\}$ *leads to the state* $\{u_4, u_5\}$*, corresponding to the decision of covering observation* x^3*. Notice that* $\{\overline{u_1}, \overline{u_2}, \overline{u_3}, u_4, u_5\}$ *does not cover observation* x^1*, but the resulting state* $\{u_4, u_5\}$ *does. Thus, the cost of the corresponding transition using the modified cost function equals 2 (see Figure 9(b)), accounting for the coverage of both* x^1 *and* x^3*. As result, the bound provided by the corresponding subproblem now is 3, accounting for the node value of 1 and the length of the longest path, which equals 2.*

4.9 Speeding up the compilation of relaxed DDs

Suppose that we are at a given iteration of a DD-based branch-and-bound and a relaxed DD with root node u must be compiled (line 10 of Algorithm 6). An optimization that can be made in order to improve the efficiency of the BAB is to interrupt the compilation of the relaxed DD as soon as we can make sure that we will not be able to prune node u by bound (which is possible when $z_{ub} \leq z_{opt}$ in line 12 of Algorithm 6). To the best of our knowledge, this detail has not been explicitly described in the literature.

Observe that the description of the DD-based BAB algorithm presented in Section 3.8 suggests the compilation of a relaxed DD (line 10 of Algorithm 6) and then the computation of the value of its longest path (line 11 of Algorithm 6). However, we can compute a longest path in a relaxed DD simultaneously to its compilation, by using the Dijkstra algorithm. In such a case, by the end of the construction of a layer of the relaxed DD (line 11 of Algorithm 4), the value $v^*(\overline{B}_{uw})$ of the longest path from the root node u to each node w in layer L_{j+1} is available.

In our proposed DD model, each arc cost in a relaxed DD is either 0 or 1 (Section 4.2). Therefore, the value of the longest path being computed simultaneously to the compilation of the relaxed DD never decreases. If a node in L_{j+1} has length value (taking node value v_u into account, see line 11 of Algorithm 6) that already exceeds the value of the best feasible solution found so far by the BAB, then the compilation of the relaxed DD can be stopped prematurely, since we already know that the bound provided by the relaxed DD is strictly higher than z_{opt}. More specifically, if $v_u + \max\{v^*(\overline{B}_{uw}) : w \in L_{j+1}\} > z_{opt}$ after the execution of line 11 of Algorithm 4, then the main loop starting at line 2 can be interrupted (lines 12 and 13 must

be adjusted accordingly, since we may not have compiled L_{n+1}).

In practice, since we do not compute the relaxed DD in its entirety, we do not actually compute the upper bound provided by it. Note that, this approach might prevent us from obtaining a frontier cutset, since there could be nodes in the FC that belonged to deep layers that had not yet been constructed. A LEL cut set can be computed, provided that at least one merging operation has been performed prior to the interruption of the compilation of the relaxed DD. If that is not the case, we simply take the last layer we constructed as a cut set.

Empirically, this strategy proved to be quite effective in reducing the overall computational time. In our implementation of a DD-based BAB, we used this particular type of cut set computation, whenever possible; otherwise, we simply used the LEL cut set, as described in Subsection 3.8.1.

5 COMPUTATIONAL RESULTS

In this chapter, we report the computation results from our experiments. All tests were carried out in a machine with an Intel (R) Core i5-8250U CPU @ 1.60GHz 1.80GHz processor, with 8 GB RAM, running 64-bit Windows 10 Home edition. In Section 5.1, we describe the instances used in the experiments. In Section 5.2, we evaluate the integer programming models found in the literature for the MPP. In Section 5.3, we compare the performance of our DD model for the MPP against the best ILP model for each instance.

5.1 Instances

In order to evaluate the ILP models for the MPP presented in Chapter 2 and the DD model presented in Chapter 4, we selected 10 datasets from the UCI Machine Learning Repository (ASUNCION; NEWMAN, 2007). In the following, we list the chosen datasets and specify the modifications made to them so as to make our results fully reproducible:

1. Abalone
    - the nominal feature in the first column was removed, thus 7 of the 8 features were used;
    - observations from class 9 were considered to be positive observations, while those from the remaining classes were considered to be negative observations.
2. Breast Cancer Wisconsin
    - observations with missing values were removed;
    - the first feature, corresponding to id numbers, was removed;
    - observations from class 2 were considered to be negative observations, while those from class 4 were considered to be positive observations.
3. Statlog (Heart)
    - observations from class 1 were considered to be negative observations, while those from class 2 were considered to be positive observations.
4. MAGIC Gamma Telescope
    - observations from class h were considered to be negative observations, while those from class g were considered to be positive observations.
5. Mammographic Mass
    - observations with missing values were removed;

- the last feature was used as the class, thus 5 of the 6 features were used.

6. Musk
   - the 2 available versions of the dataset were combined into a single one;
   - the first two columns, corresponding to the observations' ids, were removed.
7. Parkinson Speech
   - only the training dataset was used;
   - the first and penultimate columns were removed as they are not features.
8. Phishing Website
   - observations from class -1 were considered to be negative observations, while those from class 1 were considered to be positive observations.
9. QSAR Biodegradation
   - observations from class NRB were considered to be negative observations, while those from class RB were considered to be positive observations.
10. Spambase
    - no changes were made.

Table 5 – Instances

Instance	Observations			Features	
	Total	Positive	Negative	Originally	Binarized
Abalone	4,177	689	3,488	7	3,074
Breast Cancer Wisconsin	683	239	444	9	72
Statlog (Heart)	270	120	150	13	290
MAGIC Gamma Telescope	19,020	12,332	6,688	10	2,197
Mammographic Mass	830	403	427	5	74
Musk	7,074	1,224	5,850	166	35,349
Parkinson Speech	1,040	520	520	26	10,632
Phishing Website	11,055	6,157	4,898	30	38
QSAR Biodegradation	1,055	356	699	41	4,178
Spambase	4,601	1,813	2,788	57	8,006

Source: The author.

Table 5 reports the sizes of the chosen datasets after the reported changes were made and also the number of features of each dataset before and after they are binarized (BOROS *et al.*, 1997).

5.2 Evaluation of the ILP models

In order to evaluate the ILP approach for pattern generation we focused our attention on solving the MPP rather than the α-MPP, since the former is a generalization of the latter.

Table 6 – Results for the ILP models using CPLEX

Instance: Abalone						
	ILP Optimizer			CP Optimizer		
Model	f(z*)	Time(s)	Gap(%)	f(z*)	Time(s)	Gap(%)
GR	2	–	34,350.00	17	–	3,941.18
GR^{ehi}	3	–	22,886.70	15	–	4,480.00
$GR_{cliques}$	1	–	14,122.20	–	–	–
$GR^{ehi}_{cliques}$	3	–	4,640.74	–	–	–
YRa	14	–	4,805.23	**18**	–	**3,716.67**
YRa^{ehi}	11	–	6,161.59	**18**	–	**3,716.67**
YRb	2	–	34,338.80	16	–	4,193.75
YRb^{ehi}	2	–	34,350.00	15	–	4,480.00

Source: The author.

Table 7 – Results for the ILP models using CPLEX

Instance: Breast Cancer Wisconsin						
	ILP Optimizer			CP Optimizer		
Model	f(z*)	Time(s)	Gap(%)	f(z*)	Time(s)	Gap(%)
GR	137	14.27	0	137	5.65	0
GR^{ehi}	137	36.51	0	137	9.15	0
$GR_{cliques}$	137	37.54	0	137	37.94	0
$GR^{ehi}_{cliques}$	137	20.68	0	137	33.91	0
YRa	137	26.79	0	137	5.27	0
YRa^{ehi}	137	35.33	0	137	6.69	0
YRb	**137**	**0.83**	**0**	137	4.26	0
YRb^{ehi}	137	1.51	0	137	4.88	0

Source: The author.

Table 8 – Results for the ILP models using CPLEX

Instance: Statlog (Heart)						
	ILP Optimizer			CP Optimizer		
Model	f(z*)	Time(s)	Gap(%)	f(z*)	Time(s)	Gap(%)
GR	52	–	18.84	52	376.17	0
$GR_{cliques}$	52	1,293.36	0	52	1,295.48	0
YRa	52	339.58	0	52	132.90	0
YRb	**52**	**35.30**	**0**	52	250.59	0

Source: The author.
Note: No extended hypercube inequalities were generated for this instance, as there were no hypercubes in the data.

Thus, we shall not evaluate the BHK model presented in Section 2.5. Furthermore, we shall focus our attention on the GR model, for solving than MPP, rather than the RJ model, since Guo and Ryoo (2012) state that the former is more efficient in terms of computational time than the latter.

We implemented all variations of the GR model presented in Chapter 2, which were specified in Table 3, using the software IBM ILOG CPLEX Optimization Studio 12.9.0, and the C++ programming language with IDE Microsoft Visual Studio 2015. Furthermore, we

Table 9 – Results for the ILP models using CPLEX

Instance: MAGIC Gamma Telescope						
	ILP Optimizer			CP Optimizer		
Model	f(z*)	Time(s)	Gap(%)	f(z*)	Time(s)	Gap(%)
GR	–	–	–	–	–	–
GR^{ehi}	–	–	–	–	–	–
$GR_{cliques}$	–	–	–	–	–	–
$GR^{ehi}_{cliques}$	–	–	–	–	–	–
YRa	21	–	47,644.80	–	–	–
YRa^{ehi}	**28**	–	**38,803.60**	–	–	–
YRb	–	–	–	–	–	–
YRb^{ehi}	–	–	–	–	–	–

Source: The author.

Table 10 – Results for the ILP models using CPLEX

Instance: Mammographic Mass						
	ILP Optimizer			CP Optimizer		
Model	f(z*)	Time(s)	Gap(%)	f(z*)	Time(s)	Gap(%)
GR	74	25.38	0	74	17.73	0
GR^{ehi}	74	43.00	0	74	18.47	0
$GR_{cliques}$	74	36.45	0	74	180.84	0
$GR^{ehi}_{cliques}$	74	44.05	0	74	167.85	0
YRa	**74**	**1.29**	**0**	74	8.01	0
YRa^{ehi}	74	6.67	0	74	4.01	0
YRb	74	1.60	0	74	16.68	0
YRb^{ehi}	74	11.42	0	74	13.18	0

Source: The author.

Table 11 – Results for the ILP models using CPLEX

Instance: Parkinson Speech						
	ILP Optimizer			CP Optimizer		
Model	f(z*)	Time(s)	Gap(%)	f(z*)	Time(s)	Gap(%)
GR	31	–	1,577.42	–	–	–
$GR_{cliques}$	10	–	5,100.00	–	–	–
YRa	**55**	–	**839.82**	47	–	1,002.13
YRb	3	–	17,231.70	49	–	957.14

Source: The author.
Note: No extended hypercube inequalities were generated for this instance, as there were no hypercubes in the data.

also implemented all model variations using the constraint programming solver available in the software library IBM ILOG CPLEX CP Optimizer. We established a time limit of $1,800$ seconds in all of our experiments.

Tables 6 to 14 present the computational results from our experiments with CPLEX for 9 out of 10 instances. We do not present a table for the Musk instance, since we did not obtain a feasible solution within the time limit with any of the model variants. For each of the remaining 9 instances, we present a table that reports the results for each model variation, which is specified

Table 12 – Results for the ILP models using CPLEX

Instance: Phishing Website						
	ILP Optimizer			CP Optimizer		
Model	f(z*)	Time(s)	Gap(%)	f(z*)	Time(s)	Gap(%)
GR	1,465	671.53	0	1,465	–	320.13
GR^{ehi}	1,465	1,472.24	0	1,465	–	294.33
$GR_{cliques}$	1,465	1,340.98	0	–	–	–
$GR^{ehi}_{cliques}$	1,465	–	92.19	–	–	–
YRa	1,465	630.26	0	1,465	–	313.51
YRa^{ehi}	1,465	802.06	0	1,465	1,444.81	0
YRb	**1,465**	**27.96**	**0**	1,465	–	288.12
YRb^{ehi}	1,465	33.36	0	1,465	–	300.61

Source: The author.

Table 13 – Results for the ILP models using CPLEX

QSAR Biodegradation						
	ILP Optimizer			CP Optimizer		
Model	f(z*)	Time(s)	Gap(%)	f(z*)	Time(s)	Gap(%)
GR	34	–	947.05	123	–	187.80
$GR_{cliques}$	57	–	524.56	137	–	158.39
YRa	**146**	–	**135.49**	131	–	170.22
YRb	96	–	269.79	124	–	185.48

Source: The author.
Note: No extended hypercube inequalities were generated for this instance, as there were no hypercubes in the data.

Table 14 – Results for the ILP models using CPLEX

Spambase						
	ILP Optimizer			CP Optimizer		
Model	f(z*)	Time(s)	Gap(%)	f(z*)	Time(s)	Gap(%)
GR	–	–	–	–	–	–
GR^{ehi}	–	–	–	–	–	–
$GR_{cliques}$	–	–	–	–	–	–
$GR^{ehi}_{cliques}$	–	–	–	–	–	–
YRa	**742**	–	**142.54**	–	–	–
YRa^{ehi}	742	–	143.50	–	–	–
YRb	–	–	–	–	–	–
YRb^{ehi}	–	–	–	–	–	–

Source: The author.

in the first column. Columns 2 to 4 are the results using the ILP solver, while columns 5 to 7 are the results using the CP solver. For each solver, we report the best solution found (columns 2 and 5), the elapsed time (columns 3 and 6), and the optimality gap reported by the solver (columns 4 and 7). A "–" in the optimal value columns and the gap columns indicates that a feasible solution was not found. "–" in the time columns indicates that the time limit of 1,800s was exceeded.

We observed that CPLEX was able to solve 4 of 10 instances within the time limit: Breast Cancer Winsconsin, Statlog (Heart), Mammographic Mass, and Phishing Website. The

GR model did not have the best performance when compared to the YRa and YRb models for any of the instances. This stems from the fact that for the instances we tested the number of constraints in the GR model is too large compared to the number of constraints of the similar YRa and YRb models. This is also highlighted by the fact that YRb was much more competitive for the instances having the smallest numbers of features, namely: Breast Cancer Winsconsin, Statlog (Heart), Mammographic Mass, and Phishing Website. We also note that the use of CP Optimizer proved to be the best option for the Abalone instance with any of the models.

5.3 Evaluation of the DD-based algorithm

In order to evaluate the performance of our implementation of a DD-based branch-and-bound (BAB) for the MPP we compare its computational results to those of the best ILP model-solver combination presented in Section 5.2, which were highlighted in bold at Tables 6 to 14. Our goal is to evaluate whether a straightfoward implementation of the DD-based BAB is competitive against CPLEX with its default settings. We implemented the proposed DD-based BAB in C++ using Microsoft Visual Studio 2015. Again, we established a time limit of $1,800$ seconds in all of our experiments.

Table 15 presents the computational results from our experiments. Columns 2 to 5 report the results obtained using CPLEX. For the CPLEX results, we extracted the best results from Tables 6 to 14, which are highlighted in bold at each table. In columns 2 and 3, we report the model-solver combination with best performance. In columns 4 and 5, we report the objective function value and running time, respectively. For comparison, we report the results obtained using our BAB implementation as well. In columns 6 and 7, we report the objective function value and running time, respectively. In appendix B, we provide an overview of how the objective function value improved over time for each instance.

Our implementation of the BAB algorithm is based on the description presented in Section 3.8 using the DP model presented in Section 4.2. While implementing a DD-based branch-and-bound, we are required to make several choices. In order to tune the algorithm, we conducted several experiments to empirically determine a best strategy. Our implementation features:

- the maximum width of 10;
- the node selection heuristic for restricted and relaxed DDs discussed in Section 4.6;
- the BAB node selection criteria of prioritizing nodes with lower values, as mentioned in

Table 15 – Comparison CPLEX vs. DD

	CPLEX				DD	
Instance	Model	Solver	f(z^*)	Time(s)	f(z^*)	Time(s)
Abalone	YRa	CP	18	–	17	–
Breast Cancer Wisconsin	YRb	ILP	137	0.83	137	186.89
Statlog (Heart)	YRb	ILP	52	35.30	52	64.68
MAGIC Gamma Telescope	YRa$^{\texttt{ehi}}$	ILP	28	–	176	–
Mammographic Mass	YRa	ILP	74	1.29	74	2.45
Musk	–	–	–	–	647	–
Parkinson Speech	YRa	ILP	55	–	65	–
Phishing Website	YRb	ILP	1,465	27.96	1,465	–
QSAR Biodegradation	YRa	ILP	146	–	143	–
Spambase	YRa	ILP	742	–	701	–

Source: The author.
Note: – in the optimal value columns indicates that a feasible solution was not found. – in the time columns indicates that the time limit of 1,800s was exceeded.

Section 3.8.1;

- the use of the last exact layer as the exact cutset of choice, as described in Section 3.8.1;
- the use of the modified cost function for restricted DDs proposed in Section 4.8; and
- the use of the optimization technique for relaxed DDs suggested in Section 4.9.

We observed that the DD method had similar computational results to the CPLEX approach for 4 instances: Abalone, Statlog (Heart), Mammographic Mass, and QSAR biodegradation. The DD was inferior for Breast Cancer Wisconsin, Phishing Website, and Spambase instances, but was superior for the MAGIC Gamma Telescope, Musk, and Parkinson Speech instances. Interestingly, a DD-based branch-and-bound seems more robust than an ILP approach for the larger instances, which may reflect the fact that the ILP models require the insertion of constraints that bind observations-based and literals-based decision variables, while the DP model naturally makes such association by embedding the literals-based decision variables into states.

Overall, the performance of the DD-based BAB was competitive, and occasionally significantly superior, to that of the best model-solver combination obtained with CPLEX. Given the relative simplicity of the implementation of the DD-based BAB and the relatively small number of parameters and implementation choices involved, we believe that it is safe to conclude that our proposed approach constitutes an attractive alternative for solving the MPP and α-MPP models in practice. In Appendix A, we provide an empirical evaluation of the quality of the bounds provided by the relaxed DD, as compared to those obtained from the linear relaxation of the YRa model.

5.3.1 Evaluation of the variable ordering heuristic

We attempted to further improve our DD-based BAB implementation by incorporating the variable ordering heuristic proposed in Section 4.7. Table 16 reports the computational results with the variable ordering heuristic turned on (columns 2, 3, and 4) and turned off (columns 5 and 6) for comparison. In column 2, we report the objective function value of the best solution found. In column 3, we report the time for ordering the variables. In column 4, we report the search time, which includes the time spent ordering the variables. In columns 5 and 6, we report the objective function value and running time, respectively, obtained with the variable ordering heuristic turned off, which were taken directly from Table 15.

Table 16 – DD results with and without the use of a variable ordering heuristic

	Var. Order. ON			Var. Order. OFF	
Instance	f(z^*)	Ordering Time(s)	Total Time(s)	f(z^*)	Time(s)
Abalone	18	13.84	–	17	–
Breast Cancer Wisconsin	137	0.01	99.50	137	186.89
Statlog (Heart)	52	0.01	25.98	52	64.68
MAGIC Gamma Telescope	92	343.74	–	176	–
Mammographic Mass	74	0.02	2.06	74	2.45
Musk	657	474.17	–	647	–
Parkinson Speech	62	5.38	–	65	–
Phishing Website	1,465	2.39	–	1,465	–
QSAR Biodegradation	137	1.94	–	143	–
Spambase	330	76.71	–	701	–

Source: The author.
Note: – in the time columns indicates that the time limit of 1,800s was exceeded.

We observed that we were able to reduce the total time to about 50% of the original time for 2 instances: Breast Cancer Wisconsin and Statlog (Heart). We were also able to slightly improve the solutions for 2 instances: Abalone and Musk. However, we obtained worse solutions for 4 instances: Parkinson Speech, QSAR Biodegradation, MAGIC Gamma Telescope, Spambase. The solutions for the last two were significantly worse: their coverage was around 50% of the coverage we obtained originally. Overall, the effect of using our proposed variable ordering heuristic does not seem advantageous. Therefore, we recommend the use of the natural order of variables, or possibly another ordering heuristic.

6 CONCLUSIONS AND FUTURE WORK

In this work, we studied the pattern generation problem which arises in the supervised classification methodology called Logical Analysis of Data. In particular, we focused on the several integer linear programming approaches found in the literature for solving the optimization problem of finding maximum patterns. Furthermore, we also studied and described in details a decision diagram-based branch-and-bound for solving discrete optimization problems also found in the literature.

We proposed a decision diagram-based model and solution approach for the maximum pattern problem. Specifically, we proposed a valid dynamic programming model and a merging rule, in order to use the aforementioned DD-based branch-and-bound. We also suggested conditions (Section 5.3) under which an implementation of our DD model was shown to perform most efficiently in terms of computational time and solution quality.

We conducted several computational experiments in order to compare our DD-based approach against the various MILP approaches found in the literature. The results showed that our proposed approach was competitive with each of the ILP individual models from the literature. We believe it is a viable approach for the maximum pattern problem in practice.

There are a few potential directions that can be followed from this work. One direction is to adapt our DD model for solving the maximum pattern of minimum degree problem (Section 2.4), which was not explored here. Another direction is to find a way of incorporating the neighborhood properties among data found in the literature ((YAN; RYOO, 2017b) and (YAN; RYOO, 2019)) into the DD model, in order to enhance its performance.

6.1 Contributions

The main contributions of this work can be summarized as follows:

- a thorough empirical evaluation of the ILP models found in the literature for the maximum pattern problem (MPP) using a state-of-the art ILP solver;
- a DD model for both the α-MPP and the MPP;
- merging rule and heuristics in order to solve the model via a DD-based branch-and-bound;
- an empirical evaluation of the performance of the proposed DD model, comparing its computational time and solution quality against state-of-the-art ILP approaches for the MPP.

6.2 Acknowledgements

We gratefully acknowledge the financial support provided by FUNCAP (Fundação Cearense de Apoio do Desenvolvimento Científico e Tecnológico) during the development of this work.

REFERENCES

AHUJA, R. K.; MAGNANTI, T. L.; ORLIN, J. B. **Network flows**. [S.l.]: Cambridge, Mass.: Alfred P. Sloan School of Management, Massachusetts, 1988.

AKERS, S. B. Binary decision diagrams. **IEEE Transactions on Computers**, IEEE, n. 6, p. 509–516, 1978.

ALEXE, G.; ALEXE, S.; HAMMER, P. L. Pattern-based clustering and attribute analysis. **Soft Computing**, Springer, v. 10, n. 5, p. 442–452, 2006.

ALEXE, G.; HAMMER, P. L. Spanned patterns for the logical analysis of data. **Discrete Applied Mathematics**, Elsevier, v. 154, n. 7, p. 1039–1049, 2006.

ANDERSEN, H. R.; HADZIC, T.; HOOKER, J. N.; TIEDEMANN, P. A constraint store based on multivalued decision diagrams. In: SPRINGER. **International Conference on Principles and Practice of Constraint Programming**. [S.l.], 2007. p. 118–132.

ASUNCION, A.; NEWMAN, D. **UCI machine learning repository**. 2007.

BECKER, B.; BEHLE, M.; EISENBRAND, F.; WIMMER, R. BDDs in a branch and cut framework. In: SPRINGER. **International Workshop on Experimental and Efficient Algorithms**. [S.l.], 2005. p. 452–463.

BEHLE, M.; EISENBRAND, F. 0/1 vertex and facet enumeration with BDDs. In: SIAM. **2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)**. [S.l.], 2007. p. 158–165.

BERGMAN, D.; CIRE, A. A. Discrete nonlinear optimization by state-space decompositions. **Management Science**, INFORMS, v. 64, n. 10, p. 4700–4720, 2017.

BERGMAN, D.; CIRE, A. A.; HOEVE, W.-J. v.; HOOKER, J. N. Optimization bounds from binary decision diagrams. **INFORMS Journal on Computing**, INFORMS, v. 26, n. 2, p. 253–268, 2013.

BERGMAN, D.; CIRE, A. A.; HOEVE, W.-J. V.; HOOKER, J. **Decision diagrams for optimization**. [S.l.]: Springer, 2016.

BERGMAN, D.; CIRE, A. A.; HOEVE, W.-J. van. Lagrangian bounds from decision diagrams. **Constraints**, Springer, v. 20, n. 3, p. 346–361, 2015.

BERGMAN, D.; CIRE, A. A.; HOEVE, W.-J. van; HOOKER, J. N. Variable ordering for the application of BDDs to the maximum independent set problem. In: SPRINGER. **International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming**. [S.l.], 2012. p. 34–49.

BERGMAN, D.; CIRE, A. A.; HOEVE, W.-J. van; YUNES, T. BDD-based heuristics for binary optimization. **Journal of Heuristics**, Springer, v. 20, n. 2, p. 211–234, 2014.

BERGMAN, D.; CIRE, A. A.; HOEVE, W.-J. van; HOOKER, J. N. Discrete optimization with decision diagrams. **INFORMS Journal on Computing**, INFORMS, v. 28, n. 1, p. 47–66, 2016.

BERGMAN, D.; CIRÉ, A. A.; HOEVE, W. van. MDD propagation for sequence constraints. **Journal of Artificial Intelligence Research**, v. 50, p. 697–722, 2014.

BERGMAN, D.; HOEVE, W.-J. V.; HOOKER, J. N. Manipulating MDD relaxations for combinatorial optimization. In: SPRINGER. **International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems**. [S.l.], 2011. p. 20–35.

BOCCIA, M.; SFORZA, A.; STERLE, C. Simple pattern minimality problems: Integer linear programming formulations and covering-based heuristic solving approaches. **INFORMS Journal on Computing**, INFORMS, 2020.

BONATES, T. O. Large margin rule-based classifiers. **Wiley Encyclopedia of Operations Research and Management Science**, Wiley Online Library, 2010.

BONATES, T. O.; HAMMER, P. L.; KOGAN, A. Maximum patterns in datasets. **Discrete Applied Mathematics**, Elsevier, v. 156, n. 6, p. 846–861, 2008.

BOROS, E.; HAMMER, P. L.; IBARAKI, T.; KOGAN, A. Logical analysis of numerical data. **Mathematical Programming**, Springer, v. 79, n. 1-3, p. 163–190, 1997.

BOROS, E.; HAMMER, P. L.; IBARAKI, T.; KOGAN, A.; MAYORAZ, E.; MUCHNIK, I. An implementation of logical analysis of data. **IEEE Transactions on knowledge and Data Engineering**, IEEE, v. 12, n. 2, p. 292–306, 2000.

BRYANT, R. E. Graph-based algorithms for boolean function manipulation. **Computers, IEEE Transactions on**, IEEE, v. 100, n. 8, p. 677–691, 1986.

CASERTA, M.; REINERS, T. A pool-based pattern generation algorithm for logical analysis of data with automatic fine-tuning. **European Journal of Operational Research**, Elsevier, v. 248, n. 2, p. 593–606, 2016.

CHOU, C.-A.; BONATES, T. O.; LEE, C.; CHAOVALITWONGSE, W. A. Multi-pattern generation framework for logical analysis of data. **Annals of Operations Research**, Springer, v. 249, n. 1-2, p. 329–349, 2017.

CIRE, A. A.; HOEVE, W.-J. van. Multivalued decision diagrams for sequencing problems. **Operations Research**, INFORMS, v. 61, n. 6, p. 1411–1428, 2013.

CRAMA, Y.; HAMMER, P. L.; IBARAKI, T. Cause-effect relationships and partially defined boolean functions. **Annals of Operations Research**, Springer, v. 16, n. 1, p. 299–325, 1988.

GUO, C.; RYOO, H. S. Compact MILP models for optimal and pareto-optimal LAD patterns. **Discrete Applied Mathematics**, Elsevier, v. 160, n. 16-17, p. 2339–2348, 2012.

HADŽIĆ, T.; HOOKER, J. N. Cost-bounded binary decision diagrams for 0-1 programming. In: SPRINGER. **International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming**. [S.l.], 2007. p. 84–98.

HADZIC, T.; HOOKER, J. N.; O'SULLIVAN, B.; TIEDEMANN, P. Approximate compilation of constraints into multivalued decision diagrams. In: SPRINGER. **International Conference on Principles and Practice of Constraint Programming**. [S.l.], 2008. p. 448–462.

HAMMER, P. L.; KOGAN, A.; SIMEONE, B.; SZEDMÁK, S. Pareto-optimal patterns in logical analysis of data. **Discrete Applied Mathematics**, Elsevier, v. 144, n. 1-2, p. 79–102, 2004.

HANSEN, P.; MEYER, C. A new column generation algorithm for logical analysis of data. **Annals of Operations Research**, Springer, v. 188, n. 1, p. 215–249, 2011.

HODA, S.; HOEVE, W.-J. V.; HOOKER, J. N. A systematic approach to MDD-based constraint programming. In: SPRINGER. **International Conference on Principles and Practice of Constraint Programming**. [S.l.], 2010. p. 266–280.

HOOKER, J. N. Decision diagrams and dynamic programming. In: SPRINGER. **International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems**. [S.l.], 2013. p. 94–110.

HOOKER, J. N. Job sequencing bounds from decision diagrams. In: SPRINGER. **International Conference on Principles and Practice of Constraint Programming**. [S.l.], 2017. p. 565–578.

KARP, R. M. Reducibility among combinatorial problems. In: **Complexity of Computer Computations**. [S.l.]: Springer, 1972. p. 85–103.

KINABLE, J.; CIRE, A. A.; HOEVE, W.-J. van. Hybrid optimization methods for time-dependent sequencing problems. **European Journal of Operational Research**, Elsevier, v. 259, n. 3, p. 887–897, 2017.

LEE, C.-Y. Representation of switching circuits by binary-decision programs. **The Bell System Technical Journal**, Nokia Bell Labs, v. 38, n. 4, p. 985–999, 1959.

LEJEUNE, M.; LOZIN, V.; LOZINA, I.; RAGAB, A.; YACOUT, S. Recent advances in the theory and practice of logical analysis of data. **European Journal of Operational Research**, Elsevier, v. 275, n. 1, p. 1–15, 2019.

LEJEUNE, M. A. Pattern-based modeling and solution of probabilistically constrained optimization problems. **Operations Research**, INFORMS, v. 60, n. 6, p. 1356–1372, 2012.

O'NEIL, R. J.; HOFFMAN, K. Decision diagrams for solving traveling salesman problems with pickup and delivery in real time. **Operations Research Letters**, Elsevier, v. 47, n. 3, p. 197–201, 2019.

PAWLAK, Z. Rough sets. **International Journal of Computer & Information Sciences**, Springer, v. 11, n. 5, p. 341–356, 1982.

QUINLAN, J. R. Induction of decision trees. **Machine Learning**, Springer, v. 1, n. 1, p. 81–106, 1986.

RYOO, H. S.; JANG, I.-Y. MILP approach to pattern generation in logical analysis of data. **Discrete Applied Mathematics**, Elsevier, v. 157, n. 4, p. 749–761, 2009.

SERRA, T.; HOOKER, J. Compact representation of near-optimal integer programming solutions. **Mathematical Programming**, Springer, p. 1–34, 2017.

TJANDRAATMADJA, C.; HOEVE, W.-J. van. Target cuts from relaxed decision diagrams. **INFORMS Journal on Computing**, INFORMS, v. 31, n. 2, p. 285–301, 2019.

WEGENER, I. **Branching programs and binary decision diagrams: theory and applications**. [S.l.]: SIAM, 2000.

YAN, K.; RYOO, H. S. 0-1 multilinear programming as a unifying theory for LAD pattern generation. **Discrete Applied Mathematics**, Elsevier, v. 218, p. 21–39, 2017.

YAN, K.; RYOO, H. S. Strong valid inequalities for boolean logical pattern generation. **Journal of Global Optimization**, Springer, v. 69, n. 1, p. 183–230, 2017.

YAN, K.; RYOO, H. S. Cliques for multi-term linearization of 0–1 multilinear program for boolean logical pattern generation. In: SPRINGER. **World Congress on Global Optimization**. [S.l.], 2019. p. 376–386.

APPENDIX A – COMPARISON OF LINEAR RELAXATION AND DD-BASED BOUNDS

In this appendix, we provide an empirical evaluation of the relative strenght of the upper bounds provided by the linear relaxation of model YRa and our proposed relaxed DD. We also include the lower bounds provided by the restricted DD and running times. The second and third columns in Table 17 indicate the elapsed time and the objective function value (lower bound) obtained from an execution of the restricted DD algorithm. The fourth and fifth columns indicate the elapsed time and the objective function value (upper bound) obtained from an execution of the relaxed DD algorithm. A maximum width of 10 and no variable ordering was used in the execution of both algorithms. The sixth and seventh columns indicate the elapsed time and the linear relaxation value (upper bound) obtained at the root of the search tree from an execution of the branch-and-bound algorithm using the ILP solver CPLEX for solving the YRa model. The eighth column indicates the value of the best known solution. The values in bold indicate that the solution is optimal.

Table 17 – Bounds comparison

	Restricted DD		Relaxed DD		Linear Relaxation		Best known
Instance	Time(s)	Bound	Time(s)	Bound	Time(s)	Bound	solution
Abalone	0.21	7	0.83	190	1093.81	686.73	18
Breast Cancer Wisconsin	0.02	78	0.01	166	1.47	226.20	**137**
Statlog (Heart)	0.02	41	0.01	74	1.03	118.19	**52**
MAGIC Gamma Telescope	7.10	82	50.61	1,044	841.91	11,162.88	176
Mammographic Mass	0.02	55	0.01	89	1.28	74.00	**74**
Musk	94.88	255	88.08	1,214	–	–	657
Parkinson Speech	2.35	45	4.94	401	149.38	519.94	65
Phishing Website	0.71	471	2.12	2,389	82.06	5,594.13	**1,465**
QSAR Biodegradation	1.19	101	0.58	266	29.59	355.92	146
Spambase	34.13	361	97.03	1,335	220.73	1,811.82	742

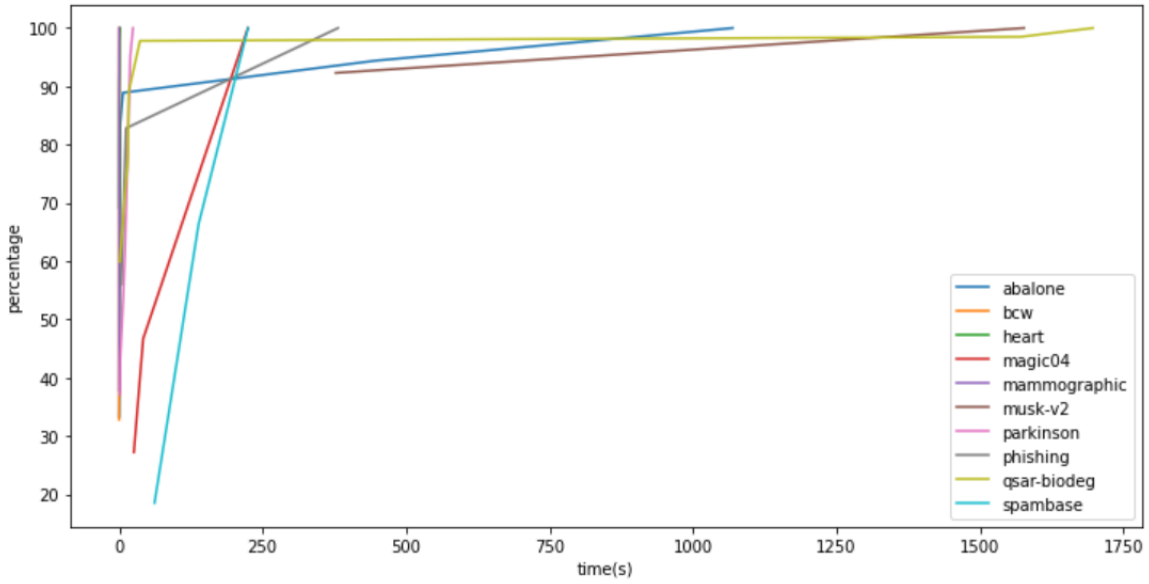
Source: The author.
Note: A "–" in the time columns and the bound columns indicate that a bound was not obtained within the time limit of 1,800s.
Further notes:

Notice that, compared to CPLEX, the relaxed DD provided a better upper bound in less time for 9 of the 10 instances. We point out that we did not turn off the default cuts and heuristics CPLEX uses at the root node of the search tree, which further attests the quality of the bound provided by the relaxed DD.

APPENDIX B – LOWER BOUND EVOLUTION IN AN EXECUTION OF THE DD-BASED BRANCH-AND-BOUND

In this appendix, we provide an overview of how the lower bound improved over time for each instance throughout an execution of the DD-based branch and bound. A maximum width of 10 and no variable ordering was used in the execution of the algorithm. In the y axis we report the relative value of the bound to the best bound obtained within the time limit of 30 minutes, while the x axis indicates the instant in time the bound was obtained during the search. Observe in Figure 10 that we obtain quality bounds (greater than 85%) within 8 minutes for every instance.

Figure 10 – Lower bound evolution for each instance



Source: The author.