University of Exeter
Faculty of Environment, Science and Economy

Improved cut generation algorithms for the acceleration of Logic-Based Benders Decomposition

Submitted by **Aigerim Saken**
to the University of Exeter as a thesis for the degree of
Doctor of Philosophy in Mathematics
September, 2023

Abstract

Logic-based Benders' decomposition (LBBD) is a solution method that integrates mixed-integer programming (MIP) and constraint programming (CP). LBBD solution scheme is a finite iterative algorithm, the central element of which are Benders' cuts. It is crucial for the convergence of the algorithm to strengthen the generated Benders' cuts. This thesis aims to improve cut generation algorithms for the acceleration of LBBD.

The cut generation algorithms are the steps of the LBBD solution scheme that can include cut-strengthening techniques and subproblem separation. As the initial step, this thesis provides an extensive computational evaluation of cut-strengthening techniques in LBBD. The evaluation also includes subproblem separation and its influence on the effectiveness cut-strengthening techniques and the overall acceleration of LBBD. The computational experiments solve cumulative facility scheduling, single-facility scheduling, and vehicle routing problems, which are representative of LBBD problems and are routinely solved with benchmark datasets available. The results of this study indicate that cut-strengthening techniques can benefit from variable sorting. Another observation from this study is that cut-strengthening techniques and subproblem separation can be used interchangeably. Three heuristics based on variable sorting are proposed to improve efficiency of cut-strengthening techniques. The main features of the proposed heuristics are simplicity and no additional computational cost. The computational results show that improved cut-strengthening techniques lead to reduction in overall solution time of the LBBD solution scheme. A novel way of separating the subproblem is developed in this thesis. The subproblem separation is based on the connected components algorithm and can be applied to subproblems that do not separate naturally. The computational results solving vehicle routing problem with local congestion show that substantial acceleration is achieved by subproblem separation.

This thesis shows that improvements to cut generation algorithm can significantly accelerate LBBD. The proposed improvements can be implemented as a part of general LBBD framework.

Acknowledgements

This PhD has been a truly life changing experience for me. There are a lot of people who made my PhD journey a very enjoyable time. I would like to give special thanks to some of them.

First of all, I would like to thank my supervisor, Dr. Stephen J. Maher, who has been a great source of guidance, support, and inspiration. Thank you for your kindness and big trust in my research abilities and ideas. I am forever grateful for the time that you have given me. I would also like to thank my co-supervisor, associate Professor Elina Rönnberg. Thank you for your warm support, mentorship, advice, and belief in me. I am very thankful for the opportunity to work with you.

I would also like to extend my gratitude to Professor Peter Challenor for his support towards the end of my PhD.

I would like to thank my friends in Exeter. A special thank you to Andrea for all the inspiring discussions, bike rides to Topsham, bouldering sessions, and many other fun activities, which are highlights of my time in Exeter. Thank you for being there during the most difficult times in my life. You are wonderful! A special thanks to Amber, you were the best flatmate I could ask for. I am very grateful for all of our chats, your kindness, and your friendship. Thank you for the support during tough times. Thank you Cassie and Nell, I am grateful for our close friendship. Thanks to everyone in Laver for all the lunches, pub excursions, and beach trips.

Thanks to all of my friends from Almaty. Aidos, Ainur, Zhamilya, and Ilyas, although you are far away, you always enrich my life with your support, humour, and kindness!

I want to thank everyone in my extended family. I want to thank my Dad. You saw me starting this journey, and you were very proud and supportive. It breaks my heart that you will never see me finish it. Thank you for everything you have done for me. I will always love you. I want to thank my Mum. I would never achieve any of this without you. Thank you for being my friend, my mentor, and my support. Thank you for all the sacrifices you have made for my happiness! I love you! Menin zhanymsiz.

I dedicate this thesis to my parents.

Funding

This research project was funded by the Faculty of Environment, Science, and Economy. Thank you for giving me the possibility to do a PhD at the University of Exeter.

Associated publications

1 A. Saken, E. Karlsson, S. J. Maher, E. Rönnberg (2023). 'Computational Evaluation of Cut-Strengthening Techniques in Logic-Based Benders' Decomposition'. *Operations Research Forum, Vol. 4, Article number: 62*.

2 A. Saken, S. J. Maher (2023). 'Subproblem Separation in Logic-Based Benders' Decomposition for the Vehicle Routing Problem with Local Congestion'. *Open Access Series in Informatics (OASIcs), Vol. 115, pp 16:1–16:12, 23rd Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2023)*.

3 A. Saken, S. J. Maher. 'Increasing computational efficiency of depth-first binary search in LBBD'. *In Preparation.*

Contents

List of Figures

Chapter 1

Introduction

Optimisation problems abound in everyday life. The practical applications of optimisation include operational research problems like resource management, facility location, machine scheduling, and airline planning. The applications in science include statistics, physics, mathematics, and cryptography (Jünger et al., 2009; Winston, 2004; Wolsey & Nemhauser, 1999). Due to the rapid development of computational resources and increased data availability, there is high demand for optimisation methods that can take advantage of this growth.

Although every optimisation problem requires individual attention, there are problem-solving methods that can be developed into general solution schemes. One class of solution schemes are decomposition methods. Decomposition strategies take advantage of the structure of problems by separating them into more tractable subproblems. Classical Benders' decomposition (BD) proposed by Benders (1962) is one of the most successful decomposition strategies. It is generally applied to problems that become linear programs when the values of some of the variables are fixed.

Logic-based Benders' decomposition (LBBD) introduced by Hooker (2000) and Hooker and Ottosson (2003) is an extension of classical BD. Both LBBD and BD separate a given problem into a master problem and one or more subproblems. Both BD and LBBD use cut generation algorithms to generate Benders' cuts and gradually reduce the solution space of the master problem. Classical BD generates Benders' cuts by using linear programming duals of the subproblems. Whereas, LBBD generates Benders' cuts by using inference duals extracted from subproblem solutions. An inference dual can be defined for any optimisation problem, therefore the subproblem does not need to take a specific form. When subproblem is a linear program, the inference dual is reduced to the linear dual. In this light, classical BD can be seen as a special case of LBBD (Hooker & Ottosson, 2003).

The LBBD solution scheme is an algorithm that converges to optimality after finitely many steps. In its standard form, the generated Benders' cuts are no-good cuts that are fully dense and only eliminate one solution from the solution space. This leads to the slow convergence of the scheme. It is therefore crucial to strengthen cuts to accelerate LBBD. Cut-strengthening techniques strengthen fully dense Benders' cuts by reducing their size and generating sparse cuts, which attempt to eliminate multiple solutions. This thesis proposes improvements to cut generation algorithms, including cut-strengthening techniques, to accelerate LBBD.

1.1 Aim of thesis

LBBD is a solution scheme, which is often implemented as an iterative algorithm. In each iteration, Benders' cuts are generated by solving the subproblem parametrised by the master problem solution. The convergence of the solution scheme is strongly influenced by the strength of the generated Benders' cuts. The main interest of this thesis are cut generation algorithms. We consider such algorithms to be the steps of the LBBD solution scheme after the master problem solution is obtained until the generated cuts are added to the master problem. Once a master problem solution is obtained, various cut generation algorithms can be applied within the solution scheme to generate Benders' cuts. The acceleration of the solution scheme leads to the reduced overall number of iterations, reduction in time required for each iteration, and therefore reduction in overall solution time. The aim of this thesis is to improve cut generation algorithms in order to accelerate the LBBD solution scheme.

As mentioned earlier, a fully dense Benders' cut eliminating a single solution is generated in the standard form of LBBD. However, only a subset of variables having certain values causes the infeasibility or suboptimality of a solution. Moreover, there are other solutions which will also be either infeasible or suboptimal because they contain such variable values. Eliminating all such solutions by generating a Benders' cut only containing a subset of variables is expected to accelerate the solution process.

A cut generation algorithm for the LBBD solution scheme can involve application of cut-strengthening techniques, subprolem separation, or both. Cut-strengthening techniques studied in this thesis strengthen the cuts by identifying a subset of variables causing infeasibility or suboptimality of the solutions. Every cut-strengthening technique employs a different search strategy for identifying such subset. One of the foci of this thesis is to improve cut-strengthening techniques by introducing novel search strategies.

Subproblem separation allows generating a Benders' cut per each separated subproblem. Therefore, it can be used to generate multiple sparse cuts instead of one fully dense cut. Karlsson and Rönnberg (2021) demonstrate the importance of subproblem separation in their study on strengthening of feasibility cuts in LBBD. However, not all subproblems separate naturally. This thesis proposes a novel way to separate subproblems that are not inherently separable.

1.2 Overview of study

There is no restriction on the type of problem LBBD can be applied to. However, in the literature it is mostly implemented as a hybrid method to combine mixed-integer programming (MIP) and constraint programming (CP). The modelling approaches and the main concepts of MIP and CP are discussed in Chapter 2. Cut-strengthening techniques considered in this thesis are based on filtering algorithms for identifying infeasible or irreducible infeasible subsets (IIS) of variables. A brief overview of research on IIS and the definition of filtering algorithms are presented in Chapter 2.

Classical Benders' decomposition is discussed in detail in Chapter 2 to demonstrate the main ideas of Benders-like algorithms. The focus of this thesis and the solution method employed throughout the work is logic-based Benders' decomposition, which is presented in Chapter 2.

The first step in improving cut-strengthening techniques is their evaluation. A detailed overview of cut-strengthening techniques studied in this thesis is presented in Chapter 3. The cut-strengthening techniques presented in Chapter 3 are the greedy algorithm, deletion filter, additive method, additive/deletion filter, and depth-first binary search (DFBS). The LBBD solution schemes and Benders' cuts for three problem formulations with various objective functions are then presented in Chapter 3. The problem formulations include cumulative facility scheduling with fixed costs, single-facility scheduling with a segmented timeline, and vehicle routing with local congestion.

The main contribution of Chapter 3 is the first evaluation of cut-strengthening techniques for both feasibility and optimality Benders' cuts. Computational experiments include separate experiments for different cut generation algorithms: applying no cut strengthening, applying cut-strengthening techniques only, applying subproblem separation only, and applying subproblem separation and cut-strengthening techniques together. Computational results in Chapter 3 showed that cut-strengthening techniques that generate irreducible cuts outperform the greedy algorithm and no application of cut strengthening. The results also demonstrate that deletion filter and DFBS have the highest computational effectiveness for all problem types. Chapter 3 highlights that cut-strengthening techniques and subproblem separation can be used interchangeably for the acceleration of LBBD.

Chapter 3 showed that the efficiency of cut-strengthening techniques can be affected by the random order of the variables. An investigation into possible variable orderings is performed in Chapter 4. Based on this investigation, Chapter 4 proposes three new heuristics for variable sorting in order to improve efficiency of cut-strengthening techniques. As one of the best-performing techniques, the new heuristics have been proposed for DFBS. The new heuristics are applied to solve the cumulative scheduling problem that minimises total tardiness, which is found to be one of the most difficult problems in Chapter 3. The computational results in Chapter 4 show that applying variable sorting increases the efficiency of DFBS and leads to lower subproblem solution time, and consequently, the lower overall solution time.

Chapter 5 proposes a novel way of subproblem separation on the example of vehicle routing problem with local congestion, for which the standard formulation is not inherently separable. Subproblem separation proposed in Chapter 5 is based on the connected components algorithm. New types of Benders' cuts for cut generation with the separated subproblem are introduced in Chapter 5. The results in Chapter 5 show that subproblem separation significantly accelerates the solution scheme. Importantly, the results also show that the type of generated Benders' cuts affects the acceleration of LBBD.

Chapter 6 will discuss conclusions from each of the previous chapters and detail the key contributions. The limitations of the proposed improvements are discussed in this chapter. The conclusions will demonstrate how the improvements to the cut generation algorithms developed in this thesis accelerate the LBBD solution scheme.

Chapter 2

Background

Logic-based Benders' decomposition (LBBD) is a hybrid method that integrates mixed-integer programming (MIP) and constraint programming (CP) in one solution scheme. Although they are different disciplines, MIP and CP share common problem-solving strategies. The main advantage of LBBD, however, is to bring together the relative strengths of MIP and CP. While MIP is known for its robust models, strong relaxation techniques, and concepts of duality, constraint programming brings powerful propagation methods, global modelling approach, and inference.

The focus of this chapter is to introduce mathematical underpinnings that are crucial for solving problems using LBBD. Sections 2.1.1 and 2.1.2 introduce problem formulations and the main concepts in MIP and CP, respectively. The problem formulations do not only demonstrate how the problem structure influences the solution process, but they are also crucial for demonstrating the range of practical applications. A comparison of MIP and CP approaches is presented in Section 2.1.3. The concepts of irreducible infeasible subsets (IIS) and filtering algorithms are presented in Section 2.1.4.

LBBD can be seen as an extension of classical Benders' decomposition (BD)—a widely used MIP problem-solving method. The classical BD and the solution scheme are given in Section 2.2.1. LBBD is then formally presented in Section 2.2.2.

In order to keep the individual chapters mostly self-contained, I will allow some redundancies and occasionally reintroduce some of these concepts.

2.1 Foundations

2.1.1 Mixed-integer programming

A rich variety of problems are solved as mixed-integer programs on a regular basis. The areas of application include management of resources, logistics, planning, portfolio analysis, network design, combinatorics, logic, data analysis, and many other problems.

Mixed-integer programming emerged as an optimisation discipline in the late 1950's. Markowitz and Manne (1957) and Dantzig (1957) showed benefits of modelling practical applications as linear programming problems, with some of the variables constrained to be integer. The term "program" used for mathematical programs is different in meaning from computer programs. Optimisation problems are called programs because of George Dantzig's application of linear programming to "programming" (planning) in the military.

MIPs are optimisation problems of the form

$$\begin{aligned} \min \quad & c^{\mathsf{T}}x + h^{\mathsf{T}}y, \\ \text{s.t.} \quad & Ax + Gy \geq b, \\ & x \in \mathbb{Z}^n_+, \\ & y \in \mathbb{R}^p_+, \end{aligned} \tag{2.1}$$

where $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_p)$ are the variables, and $\mathbb{Z}^n_+$ is a n-dimensional vector, $\mathbb{R}^p_+$ is a p-dimensional vector. Sets $\mathbb{Z}^n_+$ and $\mathbb{R}^p_+$ are the *domains* of variables x and y, respectively. The inequalities $Ax + Gy \geq b$ are the *linear constraints*. An *instance* of the problem is specified by vectors $c \in \mathbb{R}^n$, $h \in \mathbb{R}^p$, $b \in \mathbb{R}^m$, matrices $A \in \mathbb{R}^{n \times m}$ and $G \in \mathbb{R}^{p \times m}$, which represent the *problem data*. Problem (2.1) is called mixed-integer because of the presence of both integer and continuous variables.

The set $S = \{x \in \mathbb{Z}^n_+, y \in \mathbb{R}^p_+, \mid \; Ax + Gy \geq b\}$ is called the set of feasible solutions, or the *feasible set*. An instance is said to be *feasible* if the feasible set S is non-empty. The function

$$z = c^{\mathsf{T}}x + h^{\mathsf{T}}y$$

is called the *objective function*. The variables, the set of constraints, and the objective function must be defined to exactly characterize the given problem, therefore the formulation of practical problems as MIPs is not trivial.

A feasible solution (x^*, y^*) is called an *optimal solution* of problem (2.1) if it results in the smallest possible value of the objective function, that is

$$c^{\mathsf{T}}x^* + h^{\mathsf{T}}y^* \leq c^{\mathsf{T}}x + h^{\mathsf{T}}y, \quad \forall (x,y), (x^*, y^*) \in S.$$

If (x^*, y^*) is an optimal solution, $c^{\mathsf{T}}x^* + h^{\mathsf{T}}y^*$ is called the *optimal value* of the problem.

If a feasible instance of MIP does not have an optimal solution, it is *unbounded*. An instance is unbounded, if for any $u \in \mathbb{R}^1$ there is an $(x,y) \in S$ such that $c^{\mathsf{T}}x + h^{\mathsf{T}}y < u$. The notation $z = -\infty$ denotes an unbounded instance.

Throughout this thesis I assume that all of the data sets are rational. Using this assumption, every feasible instance of an MIP either has an optimal solution or is unbounded (Wolsey & Nemhauser, 1999). Thus, to solve a MIP instance means to find an optimal solution, or to show the instance is either infeasible or unbounded.

Many approaches for solving MIPs, including Benders' decomposition, are based on the fundamental concepts of *relaxation* and *duality*. A relaxation of an optimisation problem is a problem obtained by enlarging the feasible set S and/or decreasing the value of the objective function over S. A *linear relaxation* of problem (2.1) is a *linear program* (LP) obtained by removing integer restrictions on variables x.

A general linear program is a problem of the form

$$\begin{aligned} \min \quad & c^{\mathsf{T}}x, \\ \text{s.t.} \quad & Ax \geq b, \\ & x \in \mathbb{R}^n_+, \end{aligned} \tag{2.2}$$

where $A \in \mathbb{R}^{n\times m}$ is a matrix, and $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ are vectors. Theoretically and common in practice, LPs are much easier problems to solve than MIPs, and the theory and the solution methods for solving LPs are more developed. Generally, LP algorithms are often used as subroutines in MIP algorithms to obtain lower bounds on the optimal value.

Every LP is associated with another LP — its dual. The dual LP of problem (2.2), which is the primal LP, is the following problem

$$\begin{aligned} \max \quad & u^{\mathsf{T}}b, \\ \text{s.t.} \quad & u^{\mathsf{T}}A \leq c, \\ & u \in \mathbb{R}^m_+. \end{aligned} \tag{2.3}$$

The dual is formulated to find a conic combination of all constraints of the primal LP, such that the combination is a maximal underestimator of the primal objective. Each constraint in the primal LP corresponds to a variable in the dual LP, and each variable in the primal LP becomes a constraint in the dual LP. One can note that the dual of problem (2.3) is the primal LP (2.2). The relationship between the two LPs can be described based on the following theorems.

Theorem 1 *Weak duality theorem. If x^* is a feasible solution of problem* (2.2) *and u^* is a feasible solution of problem* (2.3)*, then*

$$u^{*\mathsf{T}}b \leq c^{\mathsf{T}}x^*.$$

By Theorem (1), if the dual problem has an unbounded optimal value, the primal problem is infeasible.

Theorem 2 *Strong duality theorem. If problem* (2.2) *has a finite optimal value $c^\intercal x^*$ or problem* (2.3) *has a finite optimal value $u^{*\intercal} b$, then both problems have a finite optimal value and*

$$c^\intercal x^* = u^{*\intercal} b.$$

Theorem (2) is considered one of the most important and influential theorems in optimisation. By the strong duality theorem, the primal problem has a finite optimal value if and only if the dual problem has a finite optimal value. Moreover, there are four possibilities for the primal and dual problems:

- both problems have finite optimal values that are equal,
- the primal problem is unbounded and the dual is infeasible,
- the dual problem is unbounded and the primal is infeasible,
- both dual and primal problems are infeasible.

The duality theory is a crucial result for the classical Benders' decomposition, which uses the linear duality to generate Benders' cuts. Logic-based Benders' decomposition is based on the concept of inference duality, for which the linear duality can be considered a special case.

2.1.2 Constraint programming

Constraint programming takes roots in the area of *logic programming* and *artificial intelligence*. One of the early works on logic programming is by Kowalski (1974). Kowalski (1974) introduces predicate logic as a programming language with the idea of formalising the properties of rational human thought in man-to-machine communication. Based on Kowalski's work, Colmerauer et al. (1972) introduced one of the first logic programming languages PROLOG. PROLOG has been used for theorem proving, automated planning, and language processing.

The *constraint logic programming* paradigm emerged in the works by Jaffar and Lassez (1987), Dincbas, Simonis and Van Hentenryck (1988), and Colmerauer (1990) when logic programming was integrated with constraint solving. Eventually, the term *constraint programming* emerged. The classic sources in this area are Marriott and Stuckey (1998), Van Hentenryck (1989), Tsang (1993). Some of the practical applications of constraint programming include scheduling, verification, planning, vehicle routing, and resource allocation.

A constraint program can be modelled as

$$\begin{aligned} \min \quad & f(x), \\ \text{s.t.} \quad & C(x), \\ & x \in D_x, \end{aligned} \tag{2.4}$$

where $x = (x_1, \dots, x_n)$ are decision variables. Each variable x_j can take a value v_j from a finite set D_j, which is called the *domain* of x_j. $D_x = D_1 \times \ldots \times D_n$ is the set of domains of variables. $C = \{C_1, \dots, C_m\}$ is the constraint set, where a constraint $C_i \in C$ is a relation $R_i \subset D_x$ defined on a subset of variables. A constraint C_i is satisfied if $(v_1, \dots, v_n) \in R_i$. $f : D_x \mapsto \mathbb{R}$ is an objective function that can take any form.

A constraint program is *feasible* or *satisfiable* if there exists a tuple $v_1, \dots, v_n$ that satisfies all constraints in C. If such a tuple does not exist, the program is *unsatisfiable* or *infeasible*. If a satisfiable solution $v = (v_1, \dots, v_n)$ results in the smallest possible value of the objective function, that is,

$$f(v) \leq f(x), \quad \forall x \in D_x,$$

then v is the optimal solution and $f(v)$ is the optimal value.

A feasibility version of problem (2.4), i.e., a version with no objective function is called a *constraint satisfaction problem* (CSP). The problem is to find a solution $x \in D_x$ satisfying $C(x)$, or to prove that no such solution exists. No distinction between problem-solving methods for CP and CSP is maintained in this thesis.

The main algorithm to solve constraint programs is the domain reduction algorithm. The original problem is usually split into subproblem by variables' domains. The domains of each subproblem are then reduced until a feasible solution is found. For each subproblem, domain reductions are inferred from constraints, implications of one constraint are then propagated to other constraints. If a variable domain is found to be empty, the corresponding subproblem is abandoned and different subproblem is solved. CP solvers usually have a library of specific propagators for different constraint types.

Constraints in constraint programming can be in general defined by any relation $R \subset D_1 \times \ldots \times D_n$. This includes linear or nonlinear equations and inequalities, which are often referred to as *arithmetic constraints*. However, an important feature of constraint programming is that it offers a variety of other constraints, referred to as *symbolic constraints* (Bockmayr & Hooker, 2005). Symbolic constraints obtained by grouping together a number of simple constraints, each involving a number of variables, into a single constraint involving all of these variables are called *global constraints*. Global constraints are a fundamental concept of constraint programming. All of the problems considered in this thesis feature global constraints. The following is an example of a global scheduling constraint.

Consider the following general problem. Multiple tasks must be scheduled to be processed on a facility, which can process jobs simultaneously. A *cumulative scheduling* constraint requires the tasks to be scheduled on a facility so that the total rate of resource consumption does not exceed a given limit at any point of time. The constraint is written as

$$\textsc{Cumulative}(x, p, r, C),$$

where $x = (x_1, \ldots, x_n)$ is a tuple of real-valued variables representing the start times of tasks. The parameter $p = (p_1, \ldots, p_n)$ is a tuple of processing times for each task, $r = (r_1, \ldots, r_n)$ are resource consumption rates of the tasks, and C is the resource capacity of the facility. The constraint requires that the total rate of resource consumption of the tasks to not exceed the capacity C at any time. The mathematical representation of the cumulative constraint is the following

$$\sum_{j \in T} r_j \leq C, \quad \forall t, \; T = \{j | x_j \leq t \leq x_j + p_j\}. \tag{2.5}$$

If only one task can be processed at a facility at any time, the cumulative scheduling constraint becomes a *disjunctive scheduling* constraint.

Disjunctive scheduling is the problem of scheduling tasks one a single facility, where only one task at a time can be processed. The disjunctive scheduling constraint is written as

DISJUNCTIVE(x, p),

where $x = (x_1, \ldots, x_n)$ is a tuple of real-valued variables indicating the start times of tasks, and the parameter $p = (p_1, \ldots, p_n)$ is a tuple of processing times for each task. The constraint enforces the disjunction

$$(x_i + p_i \leq x_j) \vee (x_j + p_j \leq x_i) \tag{2.6}$$

for all i, j with $i \neq j$.

This examples of global constraints show the expressive power of CP language. Instead of defining all of the inequalities (2.5) or (2.6), a single constraint can be defined. More importantly, a global constraint automatically invokes a powerful solution procedure that exploits the structure of the constraint.

2.1.3 MIP and CP: comparison of approaches

Although MIP and CP are different disciplines, they solve similar problems. Both disciplines share the idea of implicitly enumerating all potential solutions in order to find a feasible one. This implies solving numerous subproblems. MIP and CP approaches differ in the way they process these subproblems.

Most of the subproblems solved in a MIP solution process are linear programs. That allows MIP to address subproblems as a whole, and utilise LP algorithms. Constraint programming cannot take a similar approach because individual constraints do not communicate with each other. The only shared data between constraints are the variable domains. Nevertheless, using more specific constraints allows CP methods to obtain valuable implications during the solution process. Since MIPs often rely on linear relaxations, they lack this kind of structure, but strong implications can still be made by studying the polyhedral structure. Both CP and MIP apply various inference methods to obtain these implications.

One particular form of inference, which occurs in both CP and MIP, is generation of *no-goods*. Both in CP and MIP, no-goods or no-good cuts are generated when a solution of a problem is found to be infeasible or suboptimal. The no-goods eliminate the solution from the search. No-goods are widely used in LBBD when a solution is infeasible or suboptimal.

Generally, the difference between MIP and CP starts with the language used at the problem formulation stage. The programming language used to formulate MIPs is declarative. The declarative nature of the language means that the model does not specify how it should be solved. Whereas CPs are formulated more expressively: constraints used in problem formulations imply specific procedures that will be used in solution. This is the main idea of constraint programming. This comes from the computer science background of CP, where every statement is associated with a procedure (Hooker, 2000). The more direct formulation allows constraint programming to exploit the structure of subsets of constraints. Whereas most MIP methods require the whole problem to exhibit a structure.

Despite the differences, MIP and CP can be used simultaneously. One of the ways to combine MIP and CP and exploit their advantages is to use logic-based Benders' decomposition (LBBD). Basics of LBBD are discussed in Section 2.2.2 of this chapter.

2.1.4 Irreducible infeasible subset and filtering algorithms

As mentioned previously, it is not trivial to model mixed-integer programs. The same can be said about constraint programs, despite the expressive power of their global constraints. The problem formulations in both disciplines, especially large models, may turn out to be infeasible due to an infeasible subset of constraints. This can occur, for example, when integrating several smaller models into a larger one, or when modifying an existing large model. These models are referred to as *over-constrained* in CP. In the remainder of this section, the term ”model” implies both MIP and CP models, unless stated otherwise.

When a problem is infeasible, it is important to know the subset of constraints causing infeasibility in order to proceed repairing the model. Alternatively, a modeller can be interested in a subset of constraints that have a solution. This information gives insight into the structure of the problem. However, obtaining such information can be more difficult than solving the problem. This is because analysing infeasible models requires solving numerous variations of the original model (Guieu & Chinneck, 1999). Therefore, the search for such subsets needs to be assisted by an efficient algorithm.

The issue of identifying infeasible subsets of constraint in LP inspired an early work by Van Loon (1981). Van Loon (1981) introduced the term ”irreducibly inconsistent system” and proposed a simplex-like algorithm to identify such systems in LP. However, the proposed search is undirected, and therefore it is not computationally viable. The term introduced by van Loon was transformed into *”irreducible infeasible subset”* applicable to all types of optimisation problems.

Definition 1 *An infeasible subset (IS) is a subset of constraints of a problem, for which there is no solution.*

Definition 2 *An infeasible subset of constraints C is said to be an irreducible infeasible subset (IIS) if any proper subset of C is feasible.*

In other words, IIS is an IS, which becomes feasible if any of its constraints is removed.

Van Loon's work on identifying in LPs was followed by Gleeson and Ryan (1990), Chinneck and Dravnieks (1991), and Greenberg and Murphy (1991). Guieu and Chinneck (1999) then proposed *filtering algorithms* to identify IIS in MIPs. Most of the cut-strengthening techniques evaluated in this thesis are based on filtering algorithms proposed by Guieu and Chinneck (1999).

Definition 3 *(Chinneck & Dravnieks, 1991) An algorithm is called filtering if it gradually eliminates constraints (or variables) from the original set until the remaining subset constitutes an irreducible infeasible subset.*

The filtering algorithms in this thesis are adapted to identify an irreducible infeasible subset of variables. An IIS of variables is defined in a similar way to an IIS of constraints.

Definition 4 *An infeasible subset of variables S is said to be an irreducible infeasible subset (IIS) if any proper subset of S is feasible.*

One of the filtering algorithms first proposed by Chinneck and Dravnieks (1991), and then modified by Guieu and Chinneck (1999), is deletion filter. A simple deletion filter for an infeasible problem is described in Algorithm 1. The main idea of the algorithm is to iteratively remove each constraint, and check if it belongs to an IIS. If once the constraint is removed, the remaining problem becomes feasible, the constraint is returned to the original set. If the remaining problem is infeasible, the constraint is permanently removed. The algorithm guarantees to identify an IIS. Various cut-strengthening techniques based on filtering algorithms, including the deletion filter cut-strengthening technique, are presented in detail and evaluated in Chapter 3.

It is important to note that, all of the research on infeasibility of LPs and MIPs inspired the studies on finding the infeasible set in CP problems and satisfiability

Algorithm 1 The deletion filter

```
Input: an infeasible set of constraints
Output: a single IIS
1: for each constrain in the set do
2:     Temporarily drop the constraint from the set.
3:     Test the feasibility of the reduced set
4:     if feasible then
5:         return the dropped constraint to the set
6:     else
7:         drop the constraint permanently
8:     end if
9: end for
```

(SAT) problems. Numerous methods have been proposed to identify the Minimal Unsatisfiable Subformula/Core (MUS/MUC) in CP and SAT instances. The terms IIS, MUS, and MUC are often used interchangeably, and the methods developed in one discipline can be adapted to other disciplines.

2.2 Benders' decomposition

One of the MIP problem-solving strategies that employs the concepts of duality and relaxation is Benders' decomposition. This section first presents the classical Benders' decomposition and its solution scheme. The extension of Benders' decomposition — logic-based Benders' decomposition is then presented.

2.2.1 Classical Benders' decomposition

Classical Benders' decomposition (BD) is a widely used mixed-integer programming method that can be applied to exploit the problem structure of a program. Benders' decomposition was first proposed by Benders (1962) to solve mixed-integer programs with a bordered *block-diagonal structure*. This structure can arise from real-life applications that include facility location, network design, airline planning, chemical process design, where the Benders' decomposition is the best technique to handle the large scale of the problems.

Let us consider MIP of the form

$$\min c^{\mathsf{T}} x + h^{\mathsf{T}} y, \tag{2.7}$$

$$\text{s.t. } Ax \geq b, \tag{2.8}$$

$$Bx + Gy \geq d, \tag{2.9}$$

$$x \in D_x \subseteq \mathbb{Z}^n_+, \tag{2.10}$$

$$y \in D_y \subseteq \mathbb{R}^p_+, \tag{2.11}$$

where variables x and y are integer and continuous variables, respectively. Vectors $c \in \mathbb{R}^n$, $h \in \mathbb{R}^p$, $b \in \mathbb{R}^m$, $d \in \mathbb{R}^k$ and matrices $A \in \mathbb{R}^{n\times m}$, $B \in \mathbb{R}^{n\times k}$, $G \in \mathbb{R}^{p\times k}$ are given. The matrices A and B constitute a linking *block*, or a *border*. In this light, the integer variables x can be viewed as complicating variables. The complicating variables, when temporarily fixed, render the remaining optimisation problem comprising only the block of variables G considerably more tractable. Constraints (2.8) only contain variables x. Constraints (2.9) provide the link between variables x and y. The block-diagonal structure of Constraints (2.8)–(2.9) makes problem (2.7)–(2.11) suitable for Benders' decomposition.

Fixing the integer variables x to the trial values $\bar{x}$ in problem (2.7)–(2.11) results in the following decomposition:

$$\begin{aligned} \min \quad & c^{\mathsf{T}} x + f(x), \\ \text{s.t.} \quad & Ax \geq b, \\ & x \in D_x \subseteq \mathbb{Z}^n_+, \end{aligned} \tag{2.12}$$

where

$$\begin{aligned} f(\bar{x}) = \min \quad & h^{\mathsf{T}} y, \\ \text{s.t.} \quad & Gy \geq d - B\bar{x}, \\ & y \in D_y \subseteq \mathbb{R}^p_+, \end{aligned} \tag{2.13}$$

is a linear program parametrised by $\bar{x}$, and its *dual* is

$$\begin{aligned} \max \quad & u^{\mathsf{T}}(d - B\bar{x}), \\ \text{s.t.} \quad & u^{\mathsf{T}} G \leq h, \\ & u \in \mathbb{R}^m_+. \end{aligned} \tag{2.14}$$

Based on the strong duality theorem, we can indicate if subproblem (2.13) is infeasible, unbounded, or has a bounded optimal value by using the solution to the dual subproblem (2.14). The feasible region $F = \{u \in \mathbb{R}^m_+ | \ \pi^{\mathsf{T}} G \leq h\}$ does not depend on values of trial variables $\bar{x}$. Therefore, if F is not empty, subproblem (2.14) is either unbounded or feasible for any choice of $\bar{x}$.

Let Q and E be the sets of extreme rays and extreme points of subproblem (2.14), respectively. If subproblem (2.14) is unbounded, an extreme ray v_r with $v_r^{\mathsf{T}}(d - B\bar{x}) > 0$ is obtained. The direction of ray v_r must be avoided, because it indicates infeasibility of problem (2.13). The following constraint is then added to problem (2.12)

$$v_r^{\mathsf{T}}(d - Bx) \leq 0, \quad v_r \in Q.$$

If subproblem (2.14) has a feasible solution, the solution is one of the extreme points u_e. The following constraint is added to problem (2.12)

$$u_e^{\mathsf{T}}(d - B\bar{x}) \leq f(\bar{x}), \quad u_e \in E,$$

to restrict the direction of suboptimality $u_e^{\mathsf{T}}(d - B\bar{x}) > f(\bar{x})$.

Problem (2.12) can be linearised using a continuous variable $\eta \in \mathbb{R}^1$, to give the master problem (MP)

$$\min_x c^{\mathsf{T}} x + \eta, \tag{2.15}$$

$$\text{s.t. } Ax \geq b, \tag{2.16}$$

$$u_e^{\mathsf{T}}(d - Bx) \leq \eta, \quad e \in E, \tag{2.17}$$

$$v_r^{\mathsf{T}}(d - Bx) \leq 0, \quad r \in Q, \tag{2.18}$$

$$x \in D_x \subseteq \mathbb{Z}^n_+. \tag{2.19}$$

Constraints (2.17)–(2.18) are referred to as *optimality cuts* and *feasibility cuts*, respectively. Since the sets Q and E are exponential in size, problem (2.15) has an exponential number of constraints. Therefore, a natural approach is to consider a relaxation of the problem. In this case, a relaxation is obtained by generating only those constraints corresponding to a small number of extreme points and extreme rays. The Benders' decomposition algorithm, described below (see Algorithm 2) is an algorithm based on such relaxation.

Algorithm 2 Iterative Benders' Decomposition Algorithm

```
1: Initialize master problem variables $x$, subproblem variables $y$, and tolerance $\varepsilon$;
2: while True do
3:     Solve the master problem to obtain trial values $\bar{x}$ and the LB=$c^\intercal\bar{x}+\eta^*$;
4:     if  the master problem is infeasible then
5:         stop, the original problem is infeasible;
6:     end if
7:     Solve the subproblem  (2.13) with fixed master problem variables;
8:     if  (2.13) is infeasible then
9:         A feasibility cut $v_r^\intercal(d-Bx) \leq 0$ is added to the master problem;
10:        UB $\leftarrow \infty$;
11:    else if  (2.13) is feasible and $f(\bar{x}) > \eta^*$ then
12:        An optimality cut $u_e^\intercal(d-Bx) \leq \eta$ is added to the master problem;
13:        UB $\leftarrow c^\intercal\bar{x}+f(\bar{x})$;
14:    else if  (2.13) is feasible and $f(\bar{x}) \leq \eta^*$ then
15:        UB $\leftarrow c^\intercal\bar{x}+f(\bar{x})$;
16:        stop, the algorithm terminates;
17:    else if $|$UB-LB$| \leq \varepsilon$ then
18:        stop, the algorithm terminates;
19:    end if
20: end while
```

The traditional implementation of BD is an iterative algorithm. The two main steps of the algorithm are solving the master problem to obtain trial values $\bar{x}$ and solving the subproblem to generate Benders' cuts. In each iteration of Benders' algorithm, the master problem is first solved to obtain trial values $\bar{x}$ and a lower bound on the optimal objective value of problem (2.7)–(2.11), given by $LB = c^\intercal\bar{x}+\eta^*$. The objective function of the master problem gives a valid lower bound on the optimal objective value of the original problem, because it is a relaxation of the original problem. The solution of the master problem $(\bar{x}, \eta^*)$ is then verified by solving the subproblem (2.13), see Algorithm 2. If the solution $\bar{x}$ yields a feasible subproblem, then the sum of $c^\intercal \bar{x}$ and the objective value of the subproblem $f(\bar{x})$ provides a valid upper bound $UB = c^\intercal x + f(\bar{x})$ on the original problem. If an iteration of the algorithm does not generate a cut, or the difference between the lower and upper bounds is within the given tolerance ε, the algorithm terminates. The optimal solution for the original problem (2.7)–(2.11) is $(\bar{x}, \bar{y}(\bar{x}))$, where $\bar{y}(\bar{x})$ is found by solving (2.13) as input. The optimal objective value of the original problem is $c^\intercal \bar{x} + f(\bar{x})$. Note that Benders' algorithm is an exact method that guarantees optimality theoretically. But the optimality is not always achievable in practice. Therefore, when the difference between upper and lower bounds becomes small enough, it suggests that the algorithm has converged to a near-optimal solution. The choice of tolerance depends on the problem's characteristics, the precision required, and the available computational resources.

The presented implementation is the most straightforward, and easy to understand. This implementation is also similar to the general LBBD algorithm. However, in practice, problems can also be solved by the alternative branch-and-cut implementation.

2.2.2 Logic-based Benders' decomposition

One of the drawbacks of classical Benders' decomposition is that one can not always obtain a linear program subproblem. Logic-based Benders' decomposition was introduced as an extension to the classical BD method, with the advantage of not restricting the subproblem type. Logic-based Benders decomposition was introduced by Hooker and Yan (1995) in the context of logic circuit verification. The idea was then formally developed by Hooker (2000), and applied to 0-1 programming by Hooker and Ottosson (2003).

Similar to the classical BD, LBBD decomposes the original problem into master problem and one or many subproblems. Then, the subproblem (or its dual) is iteratively solved to generate cuts and gradually reduce the feasible set of the master problem. However, the dual of the subproblem is not a linear dual, but an 'inference dual'. The name comes from logic inference: the tightest bound on the master problem is inferred from the current solution and the constraints of the subproblem.

LBBD can be applied to problems of the form

$$\begin{aligned} \min \quad & f(x,y), \\ \text{s.t.} \quad & A(x), \\ & C(x,y), \\ & x \in D_x, \\ & y \in D_y, \end{aligned} \tag{2.20}$$

where function $f(x,y)$ can be any arbitrary mapping $f:(D_x,D_y)\mapsto\mathbb{R}$. Constraint set $A(x)$ only contains variables x, Constraints $C(x,y)$ are the linking constraints containing variables x and y. Fixing x to $\bar{x}$ defines the subproblem

$$\begin{aligned} \min \quad & f(\bar{x},y), \\ \text{s.t.} \quad & C(\bar{x},y), \\ & y \in D_y. \end{aligned} \tag{2.21}$$

The inference dual of the subproblem is

$$\begin{aligned} & \max \ v \\ & \text{s.t.}\ C(\bar{x},y) \overset{P}{\Longrightarrow} f(\bar{x},y) \geq v, \\ & v \in \mathbb{R}, \\ & P \in \mathscr{P}, \end{aligned} \tag{2.22}$$

where $\mathscr{P}$ is a family of proofs, and $C(\bar{x},y) \overset{P}{\Longrightarrow} f(\bar{x},y) \geq v$ indicates that proof P deduces bound $f(\bar{x},y) \geq v$ from $C(\bar{x},y)$. The solution of the inference dual is the proof P. An infeasible inference dual is assumed to have an optimal value of ∞.

Therefore, the strong duality holds: the optimal value of inference dual is equal to the optimal value of the original problem.

Although, LBBD can be applied to any kind of optimisation problem, the original problem is often decomposed into a MIP master problem and a CP subproblem. This allows LBBD to integrate the two approaches. The master problem and the subproblem are solved separately, but it is crucial for the effectiveness of the solution scheme to establish strong communication between them through the Benders' cuts.

The exact implementation of LBBD varies for different problems, but the main idea is the same. The general LBBD algorithm is as follows, see the pseudo-code in Algorithm 3. Let v^* be the optimal value of subproblem (2.21), and let P be the solution of the inference dual (2.22) deducing the bound $f(\bar{x},y) \geq v^*$ for $\bar{x}$. By applying the same logical deductions that are used to obtain the bound v^*, it is possible to deduce valid bounds for values of x other than $\bar{x}$. The bound v^* can be expressed through a bounding function $B_{\bar{x}}(x)$, in particular $B_{\bar{x}}(\bar{x}) = v^*$. We denote the objective value of problem (2.20) by z. A Benders' cut $z \geq B_{\bar{x}}(x)$ is derived by identifying a bound that proof P^* yields for a given x. The cut is added to the master problem, which in iteration k of LBBD algorithm takes the form

$$\begin{aligned} \min \quad & z, \\ \text{s.t.} \quad & A(x), \\ & z \geq B_{x^i}(x), \quad i = 1,\ldots,k-1, \\ & x \in D_x, \end{aligned} \tag{2.23}$$

where $x_i, \; i = 1,\ldots,k-1$ are previously obtained trial values of x. In each iteration, the optimal value z^* of the master problem provides a lower bound on the optimal value of the original problem (2.20), and the optimal value v_k^* of the subproblem provides an upper bound. The optimal values z^* increase monotonically, while the subproblem values v_k^* can increase or decrease. The algorithm terminates when the optimal value of the master problem equals to the optimal value of the subproblem. Specifically, it terminates when $z^* = \min\{v_i^* | \ i = 1, \dots, k\}$, or when the problem is infeasible with $z^* = \infty$.

Algorithm 3 Logic-Based Benders' Decomposition Algorithm

```
 1: Initialize master problem variables $x$, subproblem variables $y$, and $v_{min}$
 2: while True do
 3:     Solve the master problem to obtain trial values $x_k$ and the optimal value $z*$;
 4:     if  the master problem is infeasible then
 5:         stop, the original problem is infeasible;
 6:     end if
 7:     Solve the subproblem  (2.21) with fixed master problem variables
 8:     if  (2.21) is unbounded then
 9:         stop, the original problem is unbounded;
10:     end if
11:     let $v_k$ be the optimal value of (2.21), where $v_k = \infty$ if (2.21) is infeasible;
12:     generate a Benders cut $z \geq B_{x^k}(x)$ such that $B_{x^k}(x^k) = v_k$;
13:     if  $v_k < \infty$ then
14:         let $y^k$ be the optimal solution of (2.21);
15:         if $v_k < v_{min}$ then
16:             $v_{min} \leftarrow v_k$, $y^{best} \leftarrow y^k$
17:         end if
18:     end if
19:     if $z^* = v_{min}$ then
20:         stop, the algorithm terminates
21:     end if
22: end while
```

The practical implementation of LBBD depends on the inference method used to prove optimality when solving the subproblem. Compared to classical BD, there is no standard way of generating cuts within LBBD. The cuts must be tailored for the given problem, which allows LBBD to exploit the structure of the problem. When the subproblem (2.21) is a linear programming problem, the inference method is the nonnegative linear combination of inequalities, and the inference dual is the linear programming dual. In this light, classical Benders' decomposition can be viewed as a special case of logic-based Benders' decomposition.

2.3 Summary

LBBD is a solution method that integrates mixed-integer programming and constraint programming. The two approaches differ in the problem formulation languages, in the solution methods and procedures, and to a certain extent practical applications. However, both MIP and CP share the idea of implicitly enumerating the possible solutions. The concept of irreducible infeasible subsets is also an important overlap between the disciplines. Identifying IS or IIS of variables is the main idea of cut-strengthening techniques in LBBD. The computational evaluation of these techniques is given in Chapter 3.

While Benders' decomposition can be a powerful solution method, it is limited due to the restricted type of the subproblem. LBBD extends the classical Benders' decomposition by extending the concept of linear duality to inference duality. This allows LBBD to be applied to any kind of optimisation problem. All of the problems studied in Chapters 3, 4, and 5 are solved by applying LBBD.

Chapter 3

Computational evaluation of cut-strengthening techniques in Logic-Based Benders' decomposition

3.1 Introduction

Logic-based Benders' decomposition (LBBD) is a generalisation of classical Benders' decomposition. Removing the requirement of linear programming subproblems in classical Benders' decomposition, LBBD extends this popular mathematical programming approach to be applied to problems where the subproblem is an optimisation problem of any form. First proposed by Hooker and Ottosson (2003), LBBD handles this generalisation by making use of logical deductions from subproblem solutions to generate Benders' cuts. This is particularly useful when integrating mathematical optimisation and constraint programming. Logical deductions from solving a constraint program can be used to generate cuts in the form of no-good inequalities for addition to a mathematical optimisation problem. While no-good inequalities are general in their application, they are typically weak—causing slow convergence of the LBBD scheme. Numerous cut-strengthening techniques have been proposed to address this limitation of the LBBD scheme (Coban & Hooker, 2013; Hooker, 2007; Lam et al., 2020). However, no systematic investigation into the effectiveness of cut-strengthening techniques for both feasibility and optimality no-good Benders' cuts has previously been performed.

The computational effectiveness of the LBBD is strongly dependent on the cuts generated during the search procedure. The number of cuts and their quality has an impact on the solution times for the master problem and the number of LBBD iterations (Ciré, Çoban & Hooker, 2016). Numerous works have shown that the use of cut-strengthening techniques significantly improves the computational effectiveness of the LBBD scheme. A systematic analysis performed by Karlsson and Rönnberg (2021)—covering a selection of cut-strengthening techniques and application areas—highlighted the computational benefits to the LBBD scheme when applying cut strengthening to *feasibility cuts*. An important result from Karlsson and Rönnberg (2021) is that there is no single best technique for all problem types. However, the greedy cut-strengthening approach was typically outperformed by most of the other considered techniques.

This chapter aims to act as the first step in improving cut-strengthening techniques. To this end, the main contributions are:

- An in-depth discussion and evaluation of cut-strengthening techniques applied to both feasibility and optimality cuts.
- An investigation into the computational effectiveness of five cut-strengthening techniques commonly used to enhance the LBBD scheme. In particular, the greedy algorithm, deletion filter, additive method, additive/deletion filter, and the depth-first binary search will be evaluated.
- Detailed computational experiments covering three different problem types—cumulative facility scheduling with fixed costs, single-facility scheduling with a segmented timeline, and vehicle routing with location congestion—will provide a broad overview of the cut-strengthening techniques. These problems can be naturally decomposed using LBBD, and they are routinely solved in the LBBD literature (see Table 3.1).
- The first systematic investigation into how the efficacy of cut-strengthening techniques is strongly correlated to the problem type.
- The code related to the LBBD schemes for each of the problem types and

the cut-strengthening techniques is freely available.

This chapter is structured as follows: An overview of the literature related to the strengthening of feasibility and optimality cuts in LBBD will be presented in Section 3.2. Section 3.3 presents a brief introduction to LBBD and describes the cut-strengthening techniques investigated and evaluated in this paper. The problem types under investigation will be presented in Section 3.4. In addition to no-good optimality cuts, Analytic Benders' cuts are also generated for the considered problem types. A brief derivation of the Analytic Benders' cuts is presented in Section 3.5. The main contributions of this chapter are the results from computational experiments. Section 3.6 demonstrates the effectiveness of the cut-strengthening techniques by evaluating the solution run times and the average size of the generated cuts. Finally, concluding comments are given in Section 3.7.

3.2 Literature background

Cut strengthening is one of the most common acceleration techniques for Benders-like algorithms (Rahmaniani et al., 2017). The benefit of applying cut strengthening in an LBBD scheme has been demonstrated in works by Karlsson and Rönnberg (2021), Karlsson and Rönnberg (2022), Lam et al. (2020), Lindh, Olsson and Rönnberg (2022), Hooker (2007) and Hooker (2019), Riedler and Raidl (2018), Benini et al. (2008), Sadykov (2008). The literature shows that applying cut strengthening reduces the computational time of the solution process.

Both feasibility and optimality cuts can be generated in an LBBD scheme. Strengthening feasibility cuts in the context of LBBD can be described as finding a subset of master variables that cause infeasibility of the subproblem in the current solution. Strengthening optimality cuts within LBBD means finding a subset of variables that contribute to the optimal value of the subproblem.

A greedy approach to strengthen feasibility cuts is used by Hooker (2007), Ben-

Reference	Application	Cut strengthening	Benders’ cuts
Hooker (2007)	Cumulative facility scheduling	Greedy algorithm	Feas. and opt.
Benini et al. (2008)	Allocating and scheduling processors in multicore architecture	Greedy algorithm	Feas.
Coban and Hooker (2013)	Single-facility scheduling with segmented timeline	Greedy algorithm	Feas. and opt.
Riedler and Raidl (2018)	Selective dial-a-ride problem	Deletion filter	Feas.
Lam et al. (2020)	Planning and scheduling, vehicle routing, facility location	Deletion filter	Feas. and opt.
Lindh, Olsson and Rönnberg (2022)	Scheduling in mining	Greedy algorithm, Deletion filter	Feas.
Cambazard et al. (2004)	Hard real-time task allocation	DFBS	Feas.
Sadykov (2008)	Single-machine scheduling	Carlier algorithm	Feas.
Karlsson and Rönnberg (2022)	Avionic scheduling	DFBS	Feas.
Karlsson and Rönnberg (2021)	Cumulative scheduling, Single-facility scheduling, Vehicle routing with local congestion	DFBS, Deletion Filter, Greedy algorithm	Feas.
Evaluation in this thesis	Cumulative facility scheduling, Single-facility scheduling, Vehicle routing with local congestion	DFBS, Deletion Filter Additive method, Greedy Additive/ Deletion filter	Feas. and opt. Analytic

Table 3.1: Table summarising the literature

ini et al. (2008), and Coban and Hooker (2013). Hooker (2007) solves a cumulative facility scheduling problem, where the master problem assigns jobs to facilities and the subproblem schedules them. A single facility scheduling problem with a segmented timeline is solved in the work by Coban and Hooker (2013), where the master problem assigns jobs to time segments and the subproblem schedules them. This chapter evaluates the greedy algorithm, solving both problems from works by Coban and Hooker (2013) and Hooker (2007) for different objective types. While fast and easy to implement, computational experiments in this chapter will show that the greedy approach is often outperformed by other cut-strengthening techniques.

Riedler and Raidl (2018) and Lam et al. (2020) apply a cut-strengthening algorithm that has the same structure as a deletion filter. Riedler and Raidl (2018) solve a selective dial-a-ride problem and suggest using the cut-strengthening algorithm twice, the second time in reverse order, in an effort to increase chances of obtaining a strengthened feasibility cut of smaller cardinality. Lam et al. (2020) present results from applying the deletion filter algorithm to strengthen feasibility cuts to a range of problems, including planning and scheduling, vehicle routing with location congestion, and facility location. The vehicle routing with location congestion problem is used for the computational study in this chapter, where it is shown that deletion filter is one of the best-performing techniques.

Lindh, Olsson and Rönnberg (2022) apply cut strengthening in their LBBD scheme to solve a short-term scheduling problem for a mining application. The paper presents two algorithms for finding irreducible feasibility cuts and both of them rely on first using a greedy strategy for finding a strengthened cut, and thereafter applying a deletion filter to obtain an irreducible cut. In the first algorithm, the greedy strategy is inspired by an additive/deletion filter. In the second algorithm, the greedy strategy is problem-specific and designed to quickly find a rather strong cut that can then be further strengthened.

Karlsson and Rönnberg (2022) use the depth-first binary search (DFBS) to strengthen feasibility cuts. The authors propose a new acceleration technique for an LBBD scheme to solve an avionic scheduling problem. The acceleration technique extends the use of DFBS to aid the heuristic search for feasible solutions. The output of the cut-strengthening technique is an irreducible cut containing the variables that cause infeasibility. This information is used to select the subsets of variables that were not included in the strengthened cut and can be part of a feasible solution. The computational evaluation in this thesis demonstrates that the DFBS algorithm is one of the best-performing cut-strengthening algorithms.

Cambazard et al. (2004) use the iterative conflict detection algorithm QUICKXPLAIN (Junker, 2001) to strengthen feasibility cuts. QUICKXPLAIN first identifies an irreducible set of constraints, from which the set of variables needed to form a no-good cut is extracted. QUICKXPLAIN has limited use when the infeasibility of the subproblem is caused by a global constraint. If a global constraint is in the obtained irreducible set of the constraints, all variables connected to a global constraint are included in the no-good cut, making the strengthened cut less effective.

Sadykov (2008) proposes a branch-and-bound-type algorithm to strengthen no-good cuts. The proposed algorithm is a modified version of the Carlier algorithm (Carlier, 1982). The algorithm is implemented within a hybrid branch-and-check (Thorsteinsson, 2001) scheme to solve a scheduling problem to minimise the weighted number of late jobs on a single machine. The limitation of the modified Carlier algorithm is that the feasibility checks used by the algorithm are only valid for the single-machine scheduling problem. The study in this chapter is focused on general cut-strengthening techniques, therefore the modified Carlier algorithm is not within the scope of this thesis.

In the conference paper (Karlsson & Rönnberg, 2021), Karlsson and Rönnberg evaluate various cut-strengthening techniques on feasibility cuts within an LBBD scheme. The authors provide computational results based on three different problem formulations, including cumulative facility scheduling with fixed costs, single machine scheduling with sequence-dependent setup times and multiple time windows, and vehicle routing with location congestion. The DFBS, deletion filter, and the greedy algorithms are evaluated. The computational results, based on over 2000 instances, show that applying the DFBS algorithm and the deletion filter algorithm achieves the best computational time for the chosen applications. This chapter extends the contribution in (Karlsson & Rönnberg, 2021) by including the strengthening of optimality cuts and considering additional problem formulations. The table summarising the literature review is provided in Table 3.1.

3.3 Logic-based Benders' scheme and cut strengthening

The cut-strengthening techniques evaluated in this thesis can be applied to a variety of LBBD schemes. To facilitate the general discussion of cut-strengthening techniques a generic problem formulation and decision scheme are presented. Further, this formulation will be used in the mathematical description of applications, highlighting the general nature of the evaluated approaches.

3.3.1 Logic-based Benders' decomposition

LBBD can be applied to problems of the form

$$\begin{aligned}
[\mathsf{P}] \quad \min\ & f(x,y), \\
\text{s.t. } & A(x), \\
& H(y), \\
& C(x,y), \\
& x \in D_x, \\
& y \in D_y.
\end{aligned} \tag{3.1}$$

The feasible set of the problem is given by constraint sets $A(x)$, $H(y)$, and $C(x,y)$ and the domains of the variables x and y, given by D_x and D_y, respectively. An important characteristic of problem (3.1) is that upon fixing the values of x, the remaining problem, which is only with respect to y, becomes ‘easy’ to solve. The resulting problem, denoted by [SP($\bar{x}$)], is termed the subproblem, where $\bar{x}$ is a fixed solution for x.

The subproblem takes the form

$$\begin{aligned} [\text{SP}(\bar{x})] \qquad & \min f(\bar{x},y), \\ & \text{s.t. } C(\bar{x},y), \\ & \quad H(y), \\ & \quad y \in D_y. \end{aligned}$$

To obtain a bound on $f(\bar{x},y)$ an inference dual of the subproblem is defined and solved. The inference dual is the problem of obtaining the tightest possible lower bound on $f(\bar{x},y)$ from $C(\bar{x},y), H(y)$, and D_y:

$$\begin{aligned} & \max \ v \\ & \text{s.t.} \quad \left. \begin{aligned} & C(\bar{x},y), \\ & H(y), \\ & y \in D_y, \end{aligned} \right\} \overset{P}{\Longrightarrow} f(\bar{x},y) \geq v, \\ & \qquad v \in \mathbb{R}, \\ & \qquad P \in \mathscr{P}, \end{aligned}$$

where $A \overset{P}{\Longrightarrow} B$ means that proof P deduces B from A, and $\mathscr{P}$ is a family of proofs. The solution of the dual is a proof P, that gives the tightest possible bound $\bar{v}$ on $f(x,y)$ when $x = \bar{x}$, for details see (Hooker, 2007). Since there are no restrictions on the type of the constraints $C(x,y)$ and $H(y)$, and the function $f(x,y)$, the inference dual can be obtained for any kind of optimisation problem. When the function $f(x,y)$ is restricted to depend only on the variables x, the objective function $f(\bar{x},y)$ of the subproblem [SP($\bar{x}$)] becomes a constant, making the subproblem a feasibility problem. When function $f(x,y)$ depends on both variables x and y, the subproblem is an optimisation problem.

A lower bound on $f(x,y)$ can be provided by a bounding function $B_{\bar{x}}(x)$, that is defined using a proof P. The main idea of LBBD is to apply the reasoning used for obtaining the value $\bar{v}$ to deduce the bounding function $B_{\bar{x}}(x)$ on $f(x,y)$ for any values of x. The subscript $\bar{x}$ indicates the solution used to obtain the bounding function. The bounding function $B_{\bar{x}}(x)$ has two properties (Hooker, 2007):

Property 1 *$B_{\bar{x}}(x)$ provides a valid lower bound on $f(x,y)$ for any given $x \in D_x$. That is, $f(x,y) \geq B_{\bar{x}}(x)$ for any feasible (x,y) in problem* (3.1)*.*

Property 2 *$B_{\bar{x}}(\bar{x}) = \bar{v}$.*

It is convenient to regard $\bar{v}$ as an infinite value if the subproblem [SP$(\bar{x})$] is infeasible. Using this assumption, a strong duality property holds for the dual: the optimal value of the subproblem is always equal to the optimal value of its inference dual (Hooker & Ottosson, 2003).

3.3.1.1 Solution procedure

The solution procedure iterates between solving the master problem and the subproblem. The master problem is solved to obtain trial values of x, and the subproblem is solved to give feedback in the form of cuts. Let z^* and $\bar{v}_k$ be the optimal objective values of the master problem and the subproblem, respectively, in iteration k. In any iteration of the solution procedure, z^* provides a lower bound on the objective value of (3.1), and $\bar{v} = \min\{\bar{v}_1, ..., \bar{v}_{k-1}\}$ provides an upper bound. The value of z^* increases monotonically for each iteration, while values $\bar{v}_k$ can increase or decrease. The algorithm repeats until z^* is equal to $\bar{v}$.

The master problem in iteration k of the solution procedure is

$$
\begin{aligned}
[\text{MP}^k] \qquad & \min z, \\
& \text{s.t. } A(x), \\
& \quad z \geq B_{x^i}(x), \quad i = 1, ..., k-1, \\
& \quad [\text{Valid inequalities}], \\
& \quad x \in D_x, \\
& \quad z \in \mathbb{R},
\end{aligned}
$$

where $x^1, ..., x^{k-1}$ are the solutions of the master problems in iterations $1, ..., k-1$. The inequalities $z \geq B_{x^i}(x)$ are Benders' cuts added to the master problem in iterations $i = 1, ..., k-1$. The inclusion of [Valid inequalities] in [MP^k] highlights that it may be possible to strengthen the master problem with appropriate inequalities. In the applications considered in this chapter, the [Valid inequalities] include relaxations of constraints present in the subproblems, but also auxiliary variables and constraints.

3.3.1.2 Problem structure

The cut-strengthening techniques investigated in this thesis assume a problem structure where (i) the master problem variables are binary and (ii) only the variable values $x_j = 1$, $j \in \mathscr{J} = \{j | x_j = 1\}$ enforce constraints $C(x, y)$ on variables y. Constraints $C(x, y)$ then take the form

$$x_j = 1 \rightarrow C^j(y), \quad j \in \mathscr{J},$$

meaning that a value $x_j = 1$ enforces constraints $C^j(y)$ on variables y. This allows to obtain a relaxation of the subproblem by changing the value of an x_j to 0. Therefore, the constraints facilitate cut-strengthening techniques based on the evaluation of subproblem relaxations obtained by iteratively changing variable values $x_j = 1$, $j \in \mathscr{J} = \{j | x_j = 1\}$ to $x_j = 0$. When constraints $C^j(y)$ are connected through master variables only, the subproblem can be separated.

To simplify notation, we use the objective function formulation given by

$$f(x,y) = h(x) + v(y).$$

The formulation allows to separate the objective functions of the master problem and the subproblem.

3.3.1.3 Cut generation

Let x^k be the master problem solution in iteration k. Based on the subproblem solution, Benders' cuts are generated for both infeasible and feasible subproblems. If the subproblem is infeasible, the assignment x^k is eliminated by the following disjunction:

$$\vee_{j \in \mathscr{J}(x^k)} x_j^k \neq 1.$$

This disjunction can be formulated as a Benders' feasibility cut in the form of a linear inequality, also commonly referred to as a no-good cut

$$\sum_{j \in \mathscr{J}(x^k)} (1 - x_j) \geq 1, \tag{3.2}$$

where $\mathscr{J}(x^k)$ is a subset of $\mathscr{J}$, such that $\mathscr{J}(x^k) = \{j \in \mathscr{J} | x_j^k = 1\}$.

If the subproblem has an optimal solution, the optimality cuts, referred to as value cuts, can also be formulated analogous to no-good cuts. Let $v^* = h(x^k) + v(y^*)$, be the optimal value of the subproblem in iteration k, where y^* is the optimal solution to the subproblem for x^k. Then, a value cut $z \geq B_{x^k}(x)$ must bound $f(x,y)$ for any solution x by the value $B_{x^k}(x^k) = v^*$. The no-good formulation of this cut is given by

$$z \geq v^* \Big(1 - \sum_{j \in \mathscr{J}(x^k)} (1 - x_j)\Big). \tag{3.3}$$

The right-hand side of the inequality (3) is a bounding function $B_{x^k}(x)$. The function $B_{x^k}(x)$ satisfies Property 1 because for any feasible solution (x,y) it provides a valid lower bound on $f(x,y)$ (Hooker, 2007; Hooker & Ottosson, 2003). The lower bound is valid only because the optimal solution $f(x,y)$ is positive. The function $B_{x^k}(x)$ satisfies Property 2 because $B_{x^k}(x^k) = v^*$. Any feasible solution (x,y) of the problem then satisfies $f(x,y) \geq B_{x^k}(x)$. The inequality (3) provides the tightest bound when all the values of x_j for $j \in \mathscr{J}(x^k)$ are equal to 1.

3.3.1.4 Subproblem separation

Subproblem separation is a common strategy used to accelerate the LBBD scheme. Separation is possible when problem exhibits a bordered block diagonal structure, such that the master variables define the border. Fixing variables x to trial values makes blocks separable. The subproblem [SP(x^k)] then decouples into a separate problem [SP$_i$(x^k)] for each such block i:

$$\begin{aligned} [\text{SP}_i(x^k)] \qquad & \min f(x^k, y), \\ & \text{s.t. } C^j(y), \quad \{j \in \mathscr{J}_i | x_{ji} = 1\}, \\ & \quad y \in D_y. \end{aligned}$$

Solving subproblem i generates cuts that only include variables from $\mathscr{J}_i$. It has been highlighted in (Karlsson & Rönnberg, 2021) and (Ciré, Çoban & Hooker, 2016) that separating the subproblem significantly improves the runtime of the LBBD scheme. Not all of the problems considered in this chapter are separable.

3.3.2 Cut-strengthening techniques

The cut-strengthening algorithms presented in this section attempt to strengthen feasibility and optimality cuts by reducing the number of variables included in the corresponding constraint. The main idea behind all of the algorithms is to find a subset of $\mathscr{J}(\bar{x})$, denoted by $\mathscr{J}(\hat{x})$, where the corresponding variables induce a subproblem with an optimal objective equal to $\bar{v}$. The set $\mathscr{J}(\hat{x})$ is identified by systematically solving subproblem [SP($\bar{x}$)] with trial values corresponding to different subsets of $\mathscr{J}(\bar{x})$. Different trial values lead to different relaxations of subproblem [SP($\bar{x}$)]. The problem structure described in Section 3.3.1 allows to obtain a relaxation of the subproblem by changing trial values of decision variables to 0. Since setting variables to zero obtains a relaxation of the subproblem, the objective value of the subproblem is expected to decrease (in case of feasibility subproblems, the feasible region is expected to increase). The choice of trial values depends on the cut-strengthening algorithm.

Cut-strengthening techniques can be divided into two groups by the types of the cuts that they provide, irreducible or not. The cuts are categorised using the following definition (Karlsson & Rönnberg, 2021).

Definition 5 *Let $\bar{v}$ be the optimal value of subproblem [*$\mathrm{SP}(\bar{x})$*]. A subset $\mathscr{J}(\hat{x})$ of $\mathscr{J}(\bar{x})$ is irreducible if subproblem [*$\mathrm{SP}(\hat{x})$*] has an optimal value $\hat{v}$, such that $\hat{v} = \bar{v}$, and if for each $\tilde{x}$ such that $\mathscr{J}(\tilde{x}) \subset \mathscr{J}(\hat{x})$, it holds that [*$\mathrm{SP}(\tilde{x})$*] has an optimal value $\tilde{v} < \hat{v}$.*

Note that there can be multiple sets $\mathscr{J}(\hat{x})$ meeting the irreducibility criteria, and the sets often share overlapping subsets $\mathscr{J}(\tilde{x})$. Hence, the number of variables in an irreducible cut does not need to be of smallest cardinality, and it is typically possible to derive more than one irreducible cut from a single master problem solution.

In the following, we present the greedy algorithm, deletion filter, additive method, additive/deletion filter, and the DFBS cut-strengthening algorithms. The deletion filter, additive method, additive/deletion filter, and the DFBS algorithm will ensure that the strengthened cut is irreducible, while the greedy algorithm will not. The cut-strengthening algorithms are described for the strengthening of value cuts. The same search principle can be applied to strengthening feasibility cuts. The special case of feasibility cuts is treated in (Karlsson & Rönnberg, 2021).

Note that most cut-strengthening algorithms can be tailored to a specific problem by specifying the order in which the variables $(\bar{x}_j)_{j \in \mathscr{J}(\bar{x})}$ get selected to form subsets. To maintain generality in this study,this possibility is not exploited and a random order is used.

3.3.2.1 Greedy algorithm

The greedy algorithm searches for a subset of variables by evaluating a single index at a time until the objective value of the relaxed subproblem becomes less than $\bar{v}$, and the search stops. An index $j \in \mathscr{J}(\bar{x})$ is selected in each iteration, and the assignment $\bar{x}_j = 0$ is made, for the indices that have not been selected the assignments remain equal to 1. The subproblem is then solved. If the optimal objective value of the relaxed subproblem is equal to $\bar{v}$, the assignment $\bar{x}_j = 0$ becomes permanent, and the next iteration is performed. Otherwise, if the optimal objective value is less than $\bar{v}$, the value of x_j is restored to 1 and the greedy algorithm terminates. Thereafter, the resulting cut is returned. The resulting cut is not guaranteed to be irreducible since the algorithm does not evaluate all of the indices. The pseudo-code for the greedy algorithm is given in Algorithm 4.

Algorithm 4 The greedy cut-strengthening algorithm

```
Input: A set $\mathscr{J}(\bar{x})$
Output: A subset $\mathscr{J}(\hat{x})$
 1: let $v_k$ be the optimal value of $[SP(\bar{x})]$, where $v_k = \infty$ if $[SP(\bar{x})]$ is infeasible
 2: while True do
 3:     Select an index $j \in \mathscr{J}(\bar{x})$
 4:     $\bar{x}_j \leftarrow 0$
 5:     let $v'_k$ be the new optimal value of $[SP(\bar{x})]$
 6:     if $v'_k \neq v_k$ then
 7:         $\bar{x}_j \leftarrow 1$
 8:         $\mathscr{J}(\hat{x}) = \{j \in \mathscr{J}(\bar{x}) | \bar{x}_j = 1\}$
 9:         return $\mathscr{J}(\hat{x})$
10:     end if
11: end while
```

3.3.2.2 Deletion filter

The deletion filter is a cut-strengthening algorithm that returns an irreducible cut. The algorithm is based on the deletion filter for finding an IIS in linear programs (Chinneck & Dravnieks, 1991). There is one iteration for each $j \in \mathscr{J}(\bar{x})$ where the subproblem with the assignment $\bar{x}_j = 0$ is evaluated. If the optimal value of the subproblem is equal to $\bar{v}$, the assignment $\bar{x}_j = 0$ is made permanent in the remaining iterations and in the final subset. If the optimal value of the subproblem is not equal to $\bar{v}$, the assignment is permanently changed to $\bar{x}_j = 1$, both in the remaining iterations and in the final subset. Since there can be more than one irreducible subset, the order in which the assignments are evaluated determines the resulting subset (Chinneck & Dravnieks, 1991). Note that the deletion filter is the only cut-strengthening technique that is guaranteed to evaluate a subproblem relaxation for each $j \in \mathscr{J}$. A pseudo-code for the deletion filter algorithm that finds an irreducible value cut is given in Algorithm 5.

Algorithm 5 The deletion filter algorithm

```
Input: A set $\mathscr{J}(\bar{x})$
Output: An irreducible subset $\mathscr{J}(\hat{x})$
 1: let $v_k$ be the optimal value of $[SP(\bar{x})]$, where $v_k = \infty$ if $[SP(\bar{x})]$ is infeasible
 2: for $j \in \mathscr{J}(\bar{x})$ do
 3:     $\bar{x}_j \leftarrow 0$
 4:     let $v'_k$ be the new optimal value of $[SP(\bar{x})]$
 5:     if $v'_k \neq v_k$ then
 6:         $\bar{x}_j \leftarrow 1$
 7:     end if
 8: end for
 9: $\mathscr{J}(\hat{x}) = \{j \in \mathscr{J}(\bar{x}) | x_j = 1\}$
10: return $\mathscr{J}(\hat{x})$
```

3.3.2.3 Additive method

The additive method was introduced for detecting IIS in linear programming by Tamiz, Mardle and Jones (1996). Chinneck (1997) provided a simplified version of the algorithm that can be applied to a general set of constraints.

The search starts with three sets. An empty set I is introduced to store variables that are identified to belong to an IIS. An initially empty set T is a test set that comprises the set I and the candidate variables in each iteration. The set S is equal to the set of all variables at the start of the search. Then, in each iteration, one variable $\bar{x}_j = 1,\ j \in S$ is added to the set T and subproblem $[SP(T)]$ is evaluated, where $[SP(T)]$ is a subproblem relaxation corresponding to assignments $\bar{x}_j = 1,\ j \in T$. If the optimal value of subproblem is less than $\bar{v}$, the variable is added to the set T and the next iteration starts. Otherwise, the variable is added to the set I and removed from the set S. Subproblem $[SP(I)]$ is then evaluated. If the optimal value of subproblem $[SP(I)]$ is equal to $\bar{v}$, the search terminates with an irreducible subset of variables I, otherwise, the next iteration starts with T equal to the set I. A pseudo-code for the additive method that finds an irreducible value cut is given in Algorithm 6.

Algorithm 6 The additive method algorithm

```
Input: A set $\mathscr{J}(\bar{x})$
Output: An irreducible subset $\mathscr{J}(\hat{x})$
 1: $S \leftarrow \mathscr{J}(\bar{x}); T \leftarrow \varnothing; I \leftarrow \varnothing$
 2: let $v_k$ be the optimal value of $[SP(\bar{x})]$, where $v_k = \infty$ if $[SP(\bar{x})]$ is infeasible
 3: $\bar{x}_j \leftarrow 0, \quad j \in S$
 4: $x'_j \leftarrow 0, \quad j \in S$
 5: while True do
 6:     for $j \in S$ do
 7:         $T \leftarrow T \cup j$
 8:         $\bar{x}_i \leftarrow 1, \quad i \in T$
 9:         let $v'_k$ be the new optimal value of $[SP(\bar{x})]$
10:         if $v'_k = v_k$ then
11:             $I \leftarrow I \cup j$
12:             $S \leftarrow S \setminus j$
13:             $T \leftarrow I$
14:             $x'_i \leftarrow 1, i \in I$
15:             let $v'_k$ be the optimal value of $[SP(x')]$
16:             if $v'_k = v_k$ then
17:                 $\mathscr{J}(\hat{x})$=$I$
18:                 return $\mathscr{J}(\hat{x})$
19:             end if
20:         end if
21:     end for
22: end while
```

3.3.2.4 Additive/Deletion filter

The additive/deletion filter is a hybrid method based on an additive method and a deletion filter. The algorithm returns an irreducible cut. The additive/deletion filter can be considered as a way of removing a large set of assignments before applying a deletion filter. The first step of the additive part of the algorithm is to make the assignment $\bar{x}_j = 0$ for all $j \in \mathscr{J}(\bar{x})$. Then, in each iteration, the assignment $\bar{x}_j = 1$ is made for one index j until the optimal value of the subproblem is equal to $\bar{v}$. Thus, some of the assignments not contributing to the optimal value are removed. The resulting subset is denoted as $\bar{x}'$. The deletion filter is then applied to $\bar{x}'$. A pseudo-code for the additive/deletion algorithm is presented in Algorithm 7.

Algorithm 7 The additive/deletion algorithm

```
Input: A set $\mathscr{J}(\bar{x})$
Output: An irreducible subset $\mathscr{J}(\hat{x})$
1: let $v_k$ be the optimal value of $[SP(\bar{x})]$, where $v_k = \infty$ if $[SP(\bar{x})]$ is infeasible
2: let $\bar{x}_j \leftarrow 0, j \in \mathscr{J}$
3: for $j \in \mathscr{J}(\bar{x})$ do
4:     $\bar{x}_j \leftarrow 1$
5:     let $v'_k$ be the new optimal value of $[SP(\bar{x})]$
6:     if $v'_k = v_k$ then
7:         $\mathscr{J}(\bar{x}') = \{j \in \mathscr{J}(\bar{x}) | x_j = 1\}$
8:         return (Deletion filter ($\mathscr{J}(\bar{x}')$)
9:     end if
10: end for
```

3.3.2.5 Depth-first binary search (DFBS)

The DFBS cut-strengthening algorithm is similar to the deletion filter algorithm, but instead of evaluating only a single index at a time, subsets of indices are evaluated.

The output from the search is an irreducible subset of variables I that defines an irreducible cut. The search starts with the set I being empty. Let v_k be the optimal value of the subproblem in iteration k. The set of variables that are the current candidates for being included in I is denoted by T and initially $T = \mathscr{J}(\bar{x})$. The variables that are not among the current candidates are stored in an auxiliary set S. Initially S is an empty set. In each iteration, the goal is to identify a single index to add to set I by reducing set T until it contains only one index. Set T is split into sets T_1 and T_2 in each iteration. The algorithm then evaluates $[SP(T_1 \cup I \cup S)]$, if the optimal value is equal to the original value, the indices that belong to set T_2 are not considered in the subsequent iterations and the update $T = T_1$ is made. Otherwise, T is set to be equal T_2, S stores T_1. Whenever an index is added to I, $[SP(I)]$ is evaluated and if the optimal value of $[SP(I)]$ is equal to v_k the algorithm terminates. Otherwise, the next iteration is performed.

The final subset is guaranteed to be irreducible. By exploring a subset of variables at a time there is a possibility to decrease the number of subproblems that need to be solved. Note that the way T is split into two subsets is not specified by the algorithm. Defining the strategy of splitting the set can influence the practical performance of the algorithm. The pseudo-code for DFBS is given in Algorithm 8 and it is based on the presentation in (Atlihan & Schrage, 2008) for finding an IIS for a mathematical program. This type of algorithm is one of the components of the infeasibility analyser QUICKXPLAIN, described in (Junker, 2004), and is used to strengthen cuts in (Cambazard et al., 2004).

Algorithm 8 The DFBS cut-strengthening algorithm

```
Input: A set $\mathscr{J}(\bar{x})$
Output: An irreducible subset $\mathscr{J}(\hat{x})$
 1: $T \leftarrow \mathscr{J}(\bar{x}); S \leftarrow \varnothing; \bar{x}_j \leftarrow 0, \quad j \in \mathscr{J}(\bar{x})$
 2: let $v_k$ be the optimal value of $[SP(\bar{x})]$, where $v_k = \infty$ if $[SP(\bar{x})]$ is infeasible
 3: while True do
 4:     if $|T| \leq 1$ then
 5:         $I \leftarrow I + T$
 6:         $\bar{x}_j \leftarrow 1, \quad j \in I$
 7:         let $v'_k$ be the new optimal value of $[SP(\bar{x})]$
 8:         if $v'_k = v_k$ then
 9:             $\mathscr{J}(\hat{x}) = I$
10:             return $\mathscr{J}(\hat{x})$
11:         end if
12:         $T \leftarrow S; S \leftarrow \varnothing$
13:         if $|T| \geq 2$ then go to Line 4
14:         end if
15:         $T_2 \leftarrow T; T_1 \leftarrow \varnothing$
16:     else
17:         Split $T$ into $T_1$ and $T_2$
18:     end if
19:     $\bar{x}_j \leftarrow 1, \quad j \in S \cup T_1 \cup I$
20:     let $v'_k$ be the new optimal value of $[SP(\bar{x})]$
21:     if $v'_k = v_k$ then
22:         $T \leftarrow T_1$
23:     else
24:         $S \leftarrow S + T_1; T \leftarrow T_2$
25:     end if
26:     $\bar{x}_j \leftarrow 0, \quad j \in \mathscr{J}(\bar{x})$
27: end while
```

Example 1 *The example presented in Figure 3.1 illustrates DFBS applied to the set* $\mathscr{J} = \{1,2,3,4,5,6,7,8\}$*. Let* v *be the objective value of [*$SP(\mathscr{J})$*]. Set* T *is initially equal to* $\mathscr{J}$*, and sets* S *and* I *are empty. Set* T *is randomly split into* $T_1 = \{3,4,5,6\}$ *and* $T_2 = \{1,2,7,8\}$*. First, we evaluate the objective value* v' *of* $[SP(T_1 \cup I \cup S)]$*. Since* v' *is equal to the original objective* v*, set* T_1 *is stored* *as the new set T and set T_2 is not considered in the following iterations. Next, $T = \{3,4,5,6\}$ is again randomly split into $T_1 = \{3,4\}$ and $T_2 = \{5,6\}$. The objective value v' of $[SP(T_1 \cup I \cup S)]$ is not equal to v, therefore set T_1 is stored as set S, and set T_2 is the new set T. Next, set $T = \{5,6\}$ is split into $T_1 = \{5\}$ and $T_2 = \{6\}$. The objective value of $[SP(T_1 \cup I \cup S)] = [SP(3,4,5)]$ is equal to v, therefore T_1 is stored as a new set $T = \{5\}$. In the following iteration, since $T =$ only contains a single index, the index is permanently added to $I = \{5\}$. The objective value of [$SP(I)$] is not equal to v, hence the search continues. The values of $S = \{3,4\}$ are first stored in T, then set S is emptied. Set $T = \{3,4\}$ is then split into $T_1\{3\}$ and $T_2 = \{4\}$, and the objective value of [$SP(T_1 \cup I \cup S)$] is evaluated and is equal to v. Therefore, set T now stores $T_1 = \{3\}$. In the next iteration, set T containing a single index is permanently added to I. The objective value of $[SP(I)] = [SP(\{3,5\})]$ is equal to v, hence the irreducible subset is found. The search is complete with $I = \{3,5\}$.*

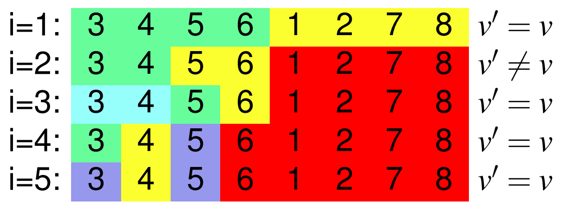

i=1: 3 4 5 6 1 2 7 8 $v' = v$
i=2: 3 4 5 6 1 2 7 8 $v' \neq v$
i=3: 3 4 5 6 1 2 7 8 $v' = v$
i=4: 3 4 5 6 1 2 7 8 $v' = v$
i=5: 3 4 5 6 1 2 7 8 $v' = v$

Figure 3.1: DFBS example: T_1= 🟩, T_2=🟨, S=🟦, I=🟪

3.4 Problems and modelling

The cut-strengthening techniques are evaluated using problems arising from manufacturing and supply chain management contexts. Specifically, we consider the cumulative facility scheduling, single facility (disjunctive) scheduling, and vehicle routing problems. An important feature of these problems is that they can be separated into assignment and scheduling components. LBBD is well suited for these kinds of problems since the assignment problem, which is routinely solved as a MIP, forms the master problem, and the scheduling problem, which is particularly amenable to CP, forms the subproblem.

3.4.1 Cumulative facility scheduling with fixed costs

An LBBD scheme for cumulative facility scheduling with fixed costs was introduced in (Hooker, 2007). The problem is to first allocate a set of jobs $\mathscr{J} = \{1,...,n\}$ to a set of facilities $\mathscr{F}$ where the jobs are then scheduled for processing. Each job $j \in \mathscr{J}$ is assigned to exactly one facility $f \in \mathscr{F}$. It takes processing time p_{jf} to finish job j at facility f and uses resources at the rate c_{jf}. The scheduled jobs can run simultaneously within one facility, but their total resource consumption at facility f cannot exceed capacity $\mathscr{C}_f$ at any time. Each job can only be scheduled to start after its release time r_j and the job must be finished before its deadline d_j.

Let the variable x_{jf} take the value 1 if job j is assigned to facility f, and 0 otherwise. Let y_{jf} be the start time of job j at facility f. The cumulative facility scheduling problem, in the form of problem (3.1), is given by

$$\min h(x) + v(y), \tag{3.4}$$

$$\text{s.t.} \sum_{f \in \mathscr{F}} x_{jf} = 1, \quad j \in \mathscr{J}, \tag{3.5}$$

$$\textsc{Cumulative}((y_{jf}|j \in \mathscr{J}),(p_{jf}|j \in \mathscr{J}),(c_{jf}|j \in \mathscr{J}),\mathscr{C}_f), \quad f \in \mathscr{F}, \tag{3.6}$$

$$x_{jf} \to r_j \leq y_{jf} \leq d_j - p_{jf}, \quad j \in \mathscr{J}, f \in \mathscr{F}, \tag{3.7}$$

$$x_{jf} \in \{0,1\}, \quad j \in \mathscr{J}, f \in \mathscr{F}, \tag{3.8}$$

$$y_{jf} \in [r_j, d_j], \quad j \in \mathscr{J}, f \in \mathscr{F}. \tag{3.9}$$

Constraints (3.5), corresponding to $A(x)$ in problem (3.1), ensure that each job is assigned to exactly one facility. Constraints (3.6) correspond to $H(y)$ and Constraints (3.7) correspond to $C(x,y)$. Constraints (3.6) control the resource consumption and Constraints (3.7) ensure that the jobs are processed within a specified time window (see the definition in Section 2.1.2). The domain D_x is given by Constraints (3.8) and domain D_y is given by Constraints (3.9).

The master problem in iteration k is given by the following assignment problem

$$\begin{aligned} &\min z, \\ &\text{s.t.} \sum_{f \in \mathscr{F}} x_{jf} = 1, \quad j \in \mathscr{J}, \\ &z \geq B_{x^i}(x), \quad i = 1, ..., k-1, \\ &\text{[Valid inequalities]}, \\ &x_{jf} \in \{0,1\}, \quad j \in \mathscr{J}, f \in \mathscr{F}. \end{aligned} \tag{3.10}$$

Let x^k_{jf} be the solution of the master problem in iteraiton k. The subproblem is then given by a scheduling problem

$$\begin{aligned} &\min v(y), \\ &\text{s.t. } \textsc{Cumulative}((y_{jf}|j \in \mathscr{J}), (p_{jf}|j \in \mathscr{J}), (c_{jf}|j \in \mathscr{J}), \mathscr{C}_f), \quad f \in \mathscr{F}, \\ &\quad x^k_{jf} = 1 \rightarrow r_j \leq y_{jf} \leq d_j - p_{jf}, \quad j \in \mathscr{J}, f \in \mathscr{F}, \end{aligned} \tag{3.11}$$

which can be separated into a scheduling problem for each facility $f \in \mathscr{F}$.

Different objective functions are considered for the cumulative scheduling problem by defining the two components $h(x)$ and $v(y)$ of the objective function (3.4): minimising the total cost of production, minimising the makespan of jobs, and minimising total tardiness of jobs. In the case where the objective function comprises only master variables, the subproblem is a feasibility problem, otherwise it is solved as an optimisation problem.

3.4.1.1 Minimising the total cost.

Assigning job j to facility f incurs cost F_{jf}. The total cost in the context of a cumulative scheduling problem is the cost of assigning all of the jobs j to facilities. The objective function that minimises the total cost is given by the two components

$$h(x) = \sum_{j \in \mathscr{J}} \sum_{f \in \mathscr{F}} F_{jf} x_{jf} \text{ and } v(y) = 0.$$

The objective function is present only in the master problem and the subproblem checks the feasibility of the assignments with respect to the facility capacity and time window constraints.

Hooker (2007) shows that it is important to include a relaxation of the subproblem (3.11) in the formulation of the master problem (3.10). In this case, the [Valid inequalities] in problem (3.10) contain a relaxation of constraints (3.6)–(3.7) as follows. Let $\mathscr{J}(t_1,t_2) = \{j \in \mathscr{J} | t_1 \leq r_j, d_j \leq t_2\}$ be a set of jobs with time windows that fit in a given time interval (t_1,t_2). Since the capacity of facility f per time unit is $\mathscr{C}_f$, the total amount of resource available at facility f in the interval (t_1,t_2) is $\mathscr{C}_f(t_2-t_1)$. Therefore, the total resource consumption $\sum_{j\in\mathscr{J}(t_1,t_2)} p_{jf}c_{jf}$ of jobs assigned to facility f cannot exceed $\mathscr{C}_f(t_2-t_1)$. This can be formulated as

$$\frac{1}{\mathscr{C}_f} \sum_{j\in\mathscr{J}(t_1,t_2)} p_{jf}c_{jf}x_{jf} \leq t_2 - t_1. \tag{3.12}$$

Only feasibility Benders' cuts are generated for this problem formulation, which are constructed as follows. Let x^k be the master problem solution in iteration k, if subproblem (3.11) is feasible, then (x^k, y^k) is the optimal solution to the original problem. Otherwise, let $\mathscr{J}_{kf} = \{j | x^k_{jf} = 1\}$ be the set of jobs assigned to facility f in iteration k and define a Benders' cut in the form of a nogood inequality (3.2) as

$$\sum_{j\in\mathscr{J}_{kf}} (1 - x_{jf}) \geq 1.$$

3.4.1.2 Minimising makespan.

In the cumulative scheduling problem the makespan is defined as the time the last job ends across all facilities. The objective function for minimising makespan is given by the components

$$h(x) = 0 \text{ and } v(y) = \max_{j\in\mathscr{J}} (y_{jf} + p_{jf}).$$

The objective function $v(y)$ is linearised by introducing auxilary variables M and M_f, that denote the makespan over all facilities and makespan of each facility f, respectively. The master problem minimises the makespan M. Each subproblem minimises M_f, and constraints

$$M_f \geq y_{jf} + p_{jf}, \quad j \in \mathscr{J}_f$$

are enforced. The variables M_f provide a bound on the variable M through inequalities $M \geq M_f, f \in \mathscr{F}$ added to the master problem.

The master problem is strengthened by [Valid inequalities] that provide a lower bound on the makespan M based on the total resource consumption of jobs assigned to facility f. These are given by

$$M \geq \frac{1}{\mathscr{C}_f} \sum_{j \in \mathscr{J}} c_{jf} p_{jf} x_{jf}, \quad f \in \mathscr{F}.$$

Since the objective function contains subproblem variables, the subproblem is an optimisation problem. As such, the subproblem generates two types of Benders' cuts. If the subproblem for facility f is infeasible, a feasibility cut in the form of nogood inequality (3.2) is generated. Otherwise, if the subproblem is feasible and has an optimal makespan M^*_{kf}, an optimality cut

$$M \geq M^*_{kf}\Big(1 - \sum_{j \in \mathscr{J}_{kf}} (1 - x_{jf})\Big) \tag{3.13}$$

is generated. The cut (3.13) provides the tightest bound on M when all of the jobs in $\mathscr{J}_{kf}$ are assigned to facility f.

3.4.1.3 Minimising total tardiness

Tardiness is defined as the time by which a job overruns its deadline. The objective function that defines the total tardiness of all jobs is given by the objective components

$$h(x) = 0 \text{ and } v(y) = \sum_{j \in \mathscr{J}} \max\{y_{jf} + p_{jf} - d_j, 0\}.$$

The objective function can be linearised by introducing auxiliary variables T and T_f that denote the total tardiness across all facilities and tardiness of each facility f, respectively. The total tardiness is minimised in the master problem with the following constraint enforced

$$T \geq \sum_{f \in \mathscr{F}} T_f.$$

The master problem can be strengthened by [Valid inequalities]. For each job $i \in \mathscr{J}$, let $\mathscr{J}(0, d_i)$ be the set of jobs that finish before its deadline d_i. Based on the resource consumption of the jobs, the total tardiness T_f of jobs $j \in \mathscr{J}(0, d_i)$ assigned to facility f is bounded below by

$$\max\left\{\frac{1}{\mathscr{C}_f} \sum_{j \in \mathscr{J}(0,d_i)} p_{jf} c_{jf} - d_i, 0\right\}, \quad i \in \mathscr{J}. \tag{3.14}$$

[Valid inequalities] can be derived from the bound (3.14). These are given by

$$\begin{aligned}
&T_f \geq \frac{1}{\mathscr{C}_f} \sum_{j \in \mathscr{J}(0,d_i)} p_{jf} c_{jf} x_{jf} - d_i, \quad f \in \mathscr{F}, i \in \mathscr{J}, \\
&T_f \geq 0, \quad f \in \mathscr{F},
\end{aligned}$$

where the variable T is the total tardiness over all facilities.

[Valid inequalities] can also include a second relaxation of the subproblem given by

$$\begin{aligned}
&T \geq \sum_{f \in \mathscr{F}} \sum_{i \in \mathscr{J}} T_{fi}, \\
&T_{fi} \geq \frac{1}{\mathscr{C}_f} \sum_{j \in \mathscr{J}} p_{\pi_f(j)f} c_{\pi_f(j)f} x_{jf} - d_i - (1 - x_{fi})\mathscr{U}_{fi}, \quad f \in \mathscr{F}, i \in \mathscr{J}, \\
&\mathscr{U}_{fi} = \frac{1}{\mathscr{C}_f} \sum_{j \in \mathscr{J}} p_{\pi_f(j)f} c_{\pi_f(j)f} - d_i,
\end{aligned}$$

where T_{fi} is a tardiness of job i on facility f and $\mathscr{U}_{fi}$ is the big-M term. These inequalities provide a valid bound only when the jobs are indexed in the order of increasing deadlines $d_1 \leq ... \leq d_n$, and π is the permutation of indices so that $p_{\pi(1)} \leq \cdots \leq p_{\pi(n)}$, see details in (Hooker, 2007).

Generation of optimality cuts is based on the same principle as the cuts given by inequality (3.13). Let T^*_{fk} be the minimum tardiness on facility f in iteration k when the jobs in $\mathscr{J}_{fk}$ are assigned to it. Then the Benders' cut in iteration k is given by

$$T_f \geq T^*_{kf}\left(1 - \sum_{j \in \mathscr{J}_{kf}} (1 - x_{jf})\right), \quad T_f \geq 0.$$

A Benders' cut can be generated for each facility $f \in \mathscr{F}$ and added to the master problem.

3.4.2 Single-facility scheduling with a segmented timeline

A single-facility scheduling problem is a problem of assigning start times y_j to a set of jobs $\mathscr{J}$ to run at a single facility. Each job $j \in \mathscr{J}$ has a processing time p_j and must be processed within its time window defined by a release time r_j and a deadline d_j. This problem does not decompose naturally. Therefore, the time horizon is divided into segments to decompose the problem into assignment and scheduling components. We study a variation of a single-facility scheduling problem with a segmented timeline that is presented in (Coban & Hooker, 2013). The main difference to the formulation in (Coban & Hooker, 2013) is that not all of the jobs have to be scheduled. First, the jobs are assigned to the segments, then scheduled within each segment. The time horizon is divided into m segments with the start and end times $[a_s, a_{s+1}]$ for $s \in \mathscr{S} = \{1, ..., m\}$. Let x_{js} take the value 1 if job j is assigned to segment s and 0 otherwise, and let y_j be the start time of job j. Note that there is a penalty for not processing a job. Since the problem is to minimise an objective function, if a job $j \in \mathscr{J}$ is not processed, a term $2p_j$ with the corresponding processing time is added to the objective. We introduce an auxiliary variable u_j, which takes value 1 if job is not processed and 0 otherwise.

The single-facility scheduling problem can be formulated in a form corresponding to (3.1) as

$$\min h(x)+v(y)+2\sum_{j\in\mathscr{J}} p_j u_j, \tag{3.15}$$

$$\text{s.t.} \sum_{s\in\mathscr{S}} x_{js}+u_j=1, \quad j\in\mathscr{J}, \tag{3.16}$$

$$\textsc{Disjunctive}((y_j|j\in\mathscr{J}),(p_j|j\in\mathscr{J})), \tag{3.17}$$

$$x_{js}\rightarrow r_j\le y_j\le d_j-p_j, \quad s\in\mathscr{S}, j\in\mathscr{J}, \tag{3.18}$$

$$x_{js}\rightarrow a_s\le y_j\le a_{s+1}-p_j, \quad s\in\mathscr{S}\setminus m, j\in\mathscr{J}, \tag{3.19}$$

$$x_{js}\in\{0,1\}, \quad j\in\mathscr{J}, s\in\mathscr{S}, \tag{3.20}$$

$$u_j\in\{0,1\}, \quad j\in\mathscr{J}, \tag{3.21}$$

$$y_j\in[r_j,d_j], \quad j\in\mathscr{J}, s\in\mathscr{S}. \tag{3.22}$$

Constraints (3.16) correspond to $A(x)$ and ensure that all jobs are assigned to no more than one segment. Constraint (3.17) corresponds to $H(y)$ and ensures that jobs do not overlap and run sequentially (see the definition in Section 2.1.2). Constraints (3.18)–(3.19) correspond to $C(x,y)$. Constraints (3.18) ensure the time windows of jobs are observed. Constraints (3.19) ensure jobs are processed within the time segments. The domains D_x and D_y are given by Constraints (3.20) and (3.22) respectively.

The master problem in iteration k is given by

$$\begin{aligned}
&\min z+2\sum_{j\in\mathscr{J}} p_j u_j,\\
&\text{s.t.} \sum_{s\in\mathscr{S}} x_{js}+u_j=1, \quad j\in\mathscr{J},\\
&\quad z\ge B_{x^i}(x), \quad i=1,\ldots,k-1,\\
&\quad \text{[Valid inequalities]},\\
&\quad x_{js}\in\{0,1\}, \quad j\in\mathscr{J}, s\in\mathscr{S},\\
&\quad u_j\in\{0,1\}, \quad j\in\mathscr{J}.
\end{aligned} \tag{3.23}$$

The master problem is augmented by [Valid inequalities] that contain the relaxation described in (Coban & Hooker, 2013). The relaxation ensures that jobs running in time interval $[t_1, t_2]$ have a total processing time of no more than $t_2 - t_1$. For each segment $s \in \mathscr{S}$, it is sufficient to enumerate all distinct intervals $[r_j, d_i]$ with $r_j < d_i$ for $i, j \in \mathscr{J}$. In order to obtain a tight inequality we consider effective bounds of the intervals within a time segment. If the given release time r_j is in the interval $[a_s, a_{s+1}]$, the effective release time is equal to the given release time. If $r_j < a_s$, the effective release time is a_s. Otherwise, if $r_j > a_{s+1}$, the effective release time is a_{s+1}. The effective deadline is defined by a similar logic. The effective bounds of interval $[r_j, d_i]$ on segment s are given by

$$\tilde{r}_{sj} = \max\{\min\{r_j, a_{s+1}\}, a_s\} \quad \text{and} \quad \tilde{d}_{si} = \min\{\max\{d_i, a_s\}, a_{s+1}\}.$$

The [Valid inequalities] are then defined as follows

$$\sum_{l \in \mathscr{J}(r_j, d_i)} p_l x_{ls} \leq \tilde{d}_{si} - \tilde{r}_{sj}, \quad s \in \mathscr{S}, \quad \forall \text{ distinct } [r_j, d_i], \tag{3.24}$$

where $\mathscr{J}(r_j, d_i)$ is the set of jobs whose time windows fall within time interval $[r_j, d_i]$.

The subproblem decomposes into a scheduling problem for each segment:

$$\begin{aligned} &\min\ v(y) \\ &\text{s.t. } \textsc{Disjunctive}((y_j | i \in \mathscr{J}), (p_j | i \in \mathscr{J})), \\ &\quad x_{js} \rightarrow r_j \leq y_j \leq d_j - p_j, \quad s \in \mathscr{S}, j \in \mathscr{J}, \\ &\quad x_{js} \rightarrow a_s \leq y_j \leq a_{s+1} - p_j, \quad s \in \mathscr{S} \setminus m, j \in \mathscr{J}. \end{aligned} \tag{3.25}$$

Depending on the objective function $h(x) + v(y)$, the subproblem is either solved as an optimisation problem or a feasibility problem. The study in this chapter includes finding a feasible schedule, minimising makespan, and minimising tardiness.

3.4.2.1 Finding a feasible schedule

The problem to find a feasible schedule results in master and subproblems that are both feasibility problems. Since unassigned jobs are penalised, the objective function $h(x)+v(y)$ is replaced by the penalty term $2\sum_{j\in\mathscr{J}} p_j u_j$. Similar to cumulative facility scheduling (3.4.1), the master problem is an assignment problem.

Let x^k be the master problem solution in iteration k. If the subproblem has a solution y^k, then the pair (x^k, y^k) is the solution to problem defined by equations (3.15)–(3.22). Otherwise, feasibility cuts are generated for each segment s. Let $\mathscr{J}_{ks} = \{j|x^k_{js} = 1\}$ be the set of jobs that are assigned to segment s in iteration k. The feasibility cuts are given by

$$\sum_{j\in\mathscr{J}_{ks}} (1-x_{js}) \geq 1, \quad s \in \mathscr{S}. \tag{3.26}$$

3.4.2.2 Minimising makespan

The objective function to minimise makespan, similar to the one presented for cumulative scheduling, is given by the components

$$h(x) = 0 \text{ and } v(y) = \max_{j\in\mathscr{J}}(y_j + p_j),$$

and the penalty term $2\sum_{j\in\mathscr{J}} p_j u_j$. The penalty term adds $2p_j$ to the objective value for each unassigned job j. To linearise the objective function, we introduce auxiliary variables M_s, which denote the makespan for each segment s, and auxiliary variable M, which denotes the makespan over all segments.

In addition to [Valid inequalities] (3.24), the master problem for the minimising makespan can be strengthened by the following bound for each distinct r_j and each segment s

$$M \geq \tilde{r}_{sj} + \sum_{l\in\mathscr{J}(r_j,\infty)} p_l x_{ls}, \quad s \in \mathscr{S}, j \in \mathscr{J}.$$

The subproblem can be separated into an optimisation problem for each segment s. If the subproblem is infeasible, feasibility cuts in the form of (3.26) are generated. If the subproblem in iteration k has a solution and M^*_{ks} is the minimum makespan for segment s, an optimality cut is given by

$$M \geq M^*_{ks}\Big(1 - \sum_{j \in \mathscr{J}_{ks}} (1 - x_{js})\Big).$$

The cut indicates that the makespan M cannot be lower than M^*_{ks} unless at least one job is removed from $\mathscr{J}_{ks}$.

3.4.2.3 Minimising total tardiness

The objective function to minimise tardiness is given by the components

$$h(x) = 0 \text{ and } v(y) = \sum_{j \in \mathscr{J}} \max\{y_j + p_j - d_j, 0\},$$

and the penalty term $2\sum_{j \in \mathscr{J}} p_j u_j$. We introduce auxiliary variables T_s to linearise the objective function. Variables T_s denote the total tardiness for each segment s, and the total tardiness over all segments $T = \sum_{s \in \mathscr{S}} T_s$ is minimised in the master problem.

Since the deadlines are now due dates, the time window constraints (3.18) are modified to

$$x_{js} \rightarrow r_j \leq y_j, \quad s \in \mathscr{S}, \quad j \in \mathscr{J}.$$

The [Valid inequalities] in the master problem (3.23) contain a modified version of the relaxation (3.24), where the effective deadline is replaced by the end of a segment. Let $\tilde{r}_{sj}$ be the effective release time of job j on segment s, and let $\mathscr{J}(r_j, \infty)$ be the set of jobs with time windows after release time r_j of job j. The [Valid inequalities] require the total processing time of jobs $j \in \mathscr{J}(r_j, \infty)$ that are assigned to segment s to not exceed $a_{s+1} - \tilde{r}_{sj}$. The inequalities are given by

$$\sum_{l \in \mathscr{J}(r_j,\infty)} p_l x_{sl} \leq a_{s+1} - \tilde{r}_{sj}, \quad s \in \mathscr{S}, \quad j \in \mathscr{J}.$$

The master problem can also be strengthened by the following bound for tardiness

$$T \geq \sum_{s \in \mathscr{S}} \big(a_s + p_j x_{js} - d_j - (1 - x_{js})(a_s - d_j)\big).$$

Since the subproblem is an optimisation problem, either a feasibility cut in the form of (3.26) or an optimality cut is generated. If the subproblem has an optimal solution in iteration k, let T^*_{ks} be the minimum tardiness on segment s. If $T^*_{ks} > 0$, we have the following optimality cut

$$T \geq T^*_{ks}\Big(1 - \sum_{j \in \mathscr{J}_{ks}} (1 - x_{js})\Big).$$

3.4.3 Vehicle routing problem with location congestion

The problem is to deliver goods from a central depot to various locations using a set of vehicles subject to vehicle capacity and location congestion constraints. The vehicle routing problem with location congestion was introduced in (Lam, Pardalos & Hentenryck, 2016) and a LBBD scheme for solving it was derived in (Lam et al., 2020).

Let R be the set of requests for goods and let $\mathscr{L}$ be the set of locations. Each request $i \in R$ is to be delivered to one location $l_i \in \mathscr{L}$ within a time window defined by a release time and a deadline, denoted by r_i and d_i, respectively. The set $R_l = \{i \in R | l_i = l\}$ is the set of all requests at location $l \in \mathscr{L}$. Each request $i \in R$ has weight q_i, and the maximum weight a vehicle can carry is Q. Each vehicle requires the use of one piece of equipment for processing time p_i to unload the goods. Each location has a fixed set of equipment, the total number is denoted by C_l. As such, there is a limited capacity at each of the locations.

This vehicle routing problem decomposes into routing and scheduling components. A graph $G = (\mathscr{N}, \mathscr{A})$ is defined to model the routing component of the problem. The set of nodes $\mathscr{N} = R \cup \{O^-, O^+\}$ includes the central depot and the set of requests with the location information, where O^- and O^+ respectively denote the artificial start and end nodes that correspond to the central depot. All vehicles must return to the central depot before time T. The set $\mathscr{A} = \{(i,j) \in \mathscr{N} \times \mathscr{N} | i \neq j\}$ denotes the arcs connecting the nodes. The master problem identifies a set of vehicle routes that satisfy all delivery requests. The variables x_{ij} equal 1 if a vehicle travels along arc (i,j), and 0 otherwise. Traversing arc (i,j) takes c_{ij} time units. There are two continuous subproblem variables at each node $i \in \mathscr{N}$. The variables y_i^{start} and y_i^{weight} are equal to the time a vehicle starts unloading goods and the total accumulated weight of delivered goods, respectively.

The vehicle routing problem with location congestion formulation is given by

$$\min h(x) + v(y), \tag{3.27}$$

$$\text{s.t.} \sum_{i:(i,j)\in\mathscr{A}} x_{ij} = 1, \quad j \in R, \tag{3.28}$$

$$\sum_{j:(i,j)\in\mathscr{A}} x_{ij} = 1, \quad i \in R, \tag{3.29}$$

$$\textsc{Cumulative}((y_i^{\text{start}}|i \in R_l), (p_i|i \in R_l), (1|i \in R_l), C_l), \quad l \in \mathscr{L}, \tag{3.30}$$

$$x_{ij} \rightarrow y_i^{\text{weight}} + q_j \leq y_j^{\text{weight}}, \quad (i,j) \in \mathscr{A}, \tag{3.31}$$

$$x_{ij} \rightarrow y_i^{\text{start}} + p_i + c_{ij} \leq y_j^{\text{start}}, \quad (i,j) \in \mathscr{A}, \tag{3.32}$$

$$x_{ij} \in \{0,1\}, \quad (i,j) \in \mathscr{A}, \tag{3.33}$$

$$y_i^{\text{start}} \in [r_i, d_i], \quad i \in \mathscr{N}, \tag{3.34}$$

$$y_i^{\text{weight}} \in [q_i, Q], \quad i \in \mathscr{N}. \tag{3.35}$$

Constraints (3.28)–(3.29), which correspond to $A(x)$ in problem (3.1), ensure that each request is assigned to exactly one vehicle. The Cumulative Constraints (3.30) correspond to $H(y)$ and enforce capacity limit at each location. Constraints (3.31)–(3.32) correspond to $C(x,y)$ in problem (3.1). The vehicle weight limits are enforced by constraints (3.31). Constraints (3.32) ensure that vehicles start unloading goods within arrival time windows corresponding to each request. Since all of the vehicles are identical and each node has exactly one incoming and outgoing arc, there is no need to represent the vehicles explicitly. The number of the arcs outgoing from (or incoming to) the central depot gives the number of vehicles used in a solution.

The master problem in iteration k is given by

$$\begin{aligned}
&\min z \\
&\text{s.t.} \sum_{i:(i,j)\in\mathscr{A}} x_{ij} = 1, \quad j \in R, \\
&\qquad \sum_{j:(i,j)\in\mathscr{A}} x_{ij} = 1, \quad i \in R, \\
&\qquad z \geq B_{x^i}(x), \quad i = 1, \ldots, k-1, \\
&\qquad [\text{Valid inequalities}].
\end{aligned}$$

The [Valid inequalities] comprise constraints (3.31)–(3.32). They state that for a vehicle that travels along arc (i,j) the accumulated weight of goods delivered after request j cannot be smaller than the accumulated weight after request i. Similarly, unloading at node j cannot start before unloading at node i. Note that [Valid inequalities] are added to the master problem regardless of the type of the objective function.

Let x^k be the master problem solution in iteration k, then the subproblem is given by

$$\begin{aligned}
&\min v(y) \\
&\text{s.t. } \textsc{Cumulative}((y_i^{\text{start}} | i \in R_l), (p_i | i \in R_l), (1 | i \in R_l), C_l), \quad l \in \mathscr{L}, \\
&\qquad x_{ij}^k = 1 \rightarrow y_i^{\text{weight}} + q_j \leq y_j^{\text{weight}}, \quad (i,j) \in \mathscr{A}, \\
&\qquad x_{ij}^k = 1 \rightarrow y_i^{\text{start}} + p_i + c_{ij} \leq y_j^{\text{start}}, \quad (i,j) \in \mathscr{A}.
\end{aligned}$$

3.4.3.1 Minimising total travel time

The goal is to minimise the total time all of the vehicles use to deliver goods and return to the central depot. The objective function is given by the components

$$h(x) = \sum_{(i,j)\in\mathscr{A}} c_{ij}x_{ij} \text{ and } v(y) = 0.$$

Since the objective function does not depend on the subproblem variables, the subproblem is solved as a feasibility problem. Therefore, only feasibility cuts are generated. Let $\mathscr{I}_k = \{(i,j) \in \mathscr{A} | x_{ij}^k = 1\}$ be the set of arcs that were selected in the master problem solution in iteration k. A feasibility cut is given by

$$\sum_{(i,j)\in\mathscr{I}_k} (1 - x_{ij}) \geq 1. \tag{3.36}$$

3.4.3.2 Minimising makespan

The goal is to minimise the time the last vehicle delivers goods and returns to the central depot. The objective function is given by

$$h(x) = 0 \text{ and } v(y) = \max_i (y_i + p_i).$$

The objective function can be linearised by introducing the auxiliary variable M that denotes makespan, which is minimised in the master problem.

Using the solution to the subproblem, either an optimality or feasibility cut is generated. If the subproblem is infeasible, a feasibility cut in the form of inequality (3.36) is generated. If the subproblem has an optimal solution with objective value M_k^* in iteration k, an optimality cut similar to inequality (3.13) is generated. This cut is given by

$$M \geq M_k^* \Big(1 - \sum_{(i,j)\in\mathscr{I}_k} (1 - x_{ij})\Big).$$

3.4.3.3 Minimising total tardiness

The tardiness of a request is defined as the time by which the delivery overruns its deadline, and the total tardiness is the total time by which all of the requests overrun their delivery deadlines. Since the requests are allowed to be delivered past their deadlines, the deadlines are replaced by due dates. The objective function to minimise the total tardiness is given by the components

$$h(x) = 0 \text{ and } v(y) = \sum_{i \in R} \max\{y_i + p_i - d_i, 0\}.$$

An auxiliary variable T, which denotes the total tardiness is introduced to linearise the objective function. The variable T is minimised in the master problem.

Using the solution to the subproblem either an optimality or feasibility cut is generated. If the subproblem is infeasible, the feasibility cut (3.36) is generated. Otherwise, if the subproblem has an optimal solution with objective value T_k^* in iteration k, an optimality cut similar to the one for minimising the makespan is generated. This optimality cut is given by

$$T \geq T_k^* \Big(1 - \sum_{(i,j) \in \mathscr{J}_k} (1 - x_{ij}) \Big), \quad T \geq 0.$$

The optimality cut provides a tight bound T_k^* on the total tardiness T when all of the arcs $(i,j) \in \mathscr{J}_k$ are added to the route.

3.5 Analytic Benders' cuts

It is possible to generate a type of Benders' cut that is based on the subproblem structure, termed analytic Benders' cuts. The aim of analytic Benders' cuts is to address the limitation of optimality cuts of form (3.3), where a tight bound is only given when each variable in $\mathscr{J}$ has the value 1. In contrast, analytic Benders' cuts improve the bound when some of the decision variables change their values from 1 to 0. The analytic cuts can be generated by analysing how changing the values of the decision variables affects the objective value. These cuts can be used along with the optimality cuts of type (3.3), or as an alternative. The analytic cuts can also be strengthened using the cut-strengthening techniques described in Section 3.3.2. Necessary assumptions to use analytic Bender's cuts are given in the following. All of the analytic cuts used in this study are based on the derivation given in (Hooker, 2007).

Cumulative scheduling

Let $\mathscr{J}_{fk}$ be the set of tasks assigned to facility f in iteration k, and M^*_{fk} be the corresponding minimum makespan. In the derivation of these cuts, it is assumed that all of the release times are equal to 0 and that the deadlines have different values. If one or more job assignments are removed from facility f, the resulting minimum makespan M_f is bounded by

$$M_f \geq M^*_{fk} - \sum_{j \in \mathscr{J}_{fk}} p_{jf}(1 - x_{jf}) - \max_{j \in \mathscr{J}_{fk}} \{d_j\} + \min_{j \in \mathscr{J}_{fk}} \{d_j\}. \tag{3.37}$$

The bound complies with the properties of the bounding function outlined in Section 3.3.1. When the supbroblem has an optimal solution with the minimum makespan M^*_{fk}, an analytic Benders' cut of form (3.37) can be generated instead of, or alongside an optimality cut of form (3.13).

Analytic Bender's cuts for the minimum tardiness problem are derived based on the same principle as the analytic cuts (3.37) for the minimising makespan problem. Let T^*_{fk} be the minimum tardiness corresponding to $\mathscr{J}_{fk}$. An analytic cut that bounds the total tardiness T over all facilities is given by

$$T \geq \sum_{f \in \mathscr{F}} \left(T^*_{fk} - \sum_{i \in \mathscr{J}_{fk}} \max\Big\{ \sum_{j \in \mathscr{J}_{fk}} p_{jf} - d_i, 0 \Big\} (1 - x_{if}) \right). \tag{3.38}$$

Disjunctive scheduling

Let $\mathcal{J}_{sk}$ be the set of jobs assigned to segment s in iteration k and let M^*_{sk} be the corresponding minimal makespan. Define $\tilde{\mathcal{J}}_{sk} = \{j \in \mathcal{J}_{sk} | r_j \leq a_s\}$ as the set of jobs in $\mathcal{J}_{sk}$ with release times before segment s. If one or more jobs from $\tilde{\mathcal{J}}_{sk}$ are no longer assigned to segment s in the subsequent iterations, a lower bound on the resulting makespan M_s is provided by an analytic cut

$$\begin{aligned} M_s \geq M^*_{sk} - \sum_{j \in \tilde{\mathcal{J}}_{sk}} p_{js}(1 - x_{js}) - \max_{j \in \mathcal{J}_{sk}} \{d_j\} \\ + \min_{j \in \tilde{\mathcal{J}}_{sk}} \{d_j\} - M^*_{sk} \sum_{j \in \mathcal{J}_{sk} \setminus \tilde{\mathcal{J}}_{sk}} (1 - x_{js}). \end{aligned} \tag{3.39}$$

The second sum in the left-hand side takes care of the case when jobs from $\mathcal{J}_{sk} \setminus \tilde{\mathcal{J}}_{sk}$ are removed from segment s, see details in (Coban & Hooker, 2013).

For the minimising tardiness problem. Let

$$r^{\max}_s = \max\Big\{ \max\{r_j | j \in \mathcal{J}_s\}, a_s \Big\}$$

be the last release time of jobs assigned to segment s, or the start time of the segment a_s, whichever is greater. Note that, if all the jobs assigned after the greatest release time are processed before the next segment, i.e., if

$$r^{\max}_s + \sum_{j \in \mathcal{J}_s} p_{js} \leq a_{s+1} \tag{3.40}$$

holds, the problem is feasible. Based on inequality (3.40), the analytic Benders' cut for minimising tardiness on segment s is

$$T_s \geq \begin{cases} \left(T^*_{sk} - \sum_{i \in \mathcal{J}_{sk}} \left(r^{\max}_s + \sum_{j \in \mathcal{J}_{sk}} p_{js} - d_i \right)^+ (1 - x_{js}) \right), \text{if (3.40) holds} \\ \left(T^*_{sk} - \left(1 - \sum_{i \in \mathcal{J}_{sk}} (1 - x_{js}) \right) \right), \quad \text{otherwise,} \end{cases}$$

where T^*_{sk} is the minimal tardiness on segment s in iteration k. The total tardiness over all segments is then bounded by

$$T \geq \sum_{s \in \mathscr{S}} T_s.$$

3.6 Computational evaluation

The effectiveness of the cut-strengthening techniques, described in Section 3.3.2, is evaluated in a series of computational experiments. The first experiment solves the cumulative scheduling problem for three objective functions given in Section 3.4.1. The experiment runs the LBBD solution scheme applying each cut-strengthening technique separately for all of the problems. The second and third experiments similarly solve the single-facility scheduling 3.4.2 and vehicle routing 3.4.3 problems, respectively.

Each experiment comprises one problem that only generates feasibility cuts and two problems that generate both feasibility and value cuts. We extend the computational experiments from the conference paper (Karlsson & Rönnberg, 2021) by adding an analysis of problems that generate value cuts. Moreover, two types of value cuts are analysed — strengthened optimality cuts and analytic Benders' cuts. Each corresponding problem is solved twice with respect to the type of generated value cuts. The cut-strengthening techniques are applied to both types of cuts. In the case when the subproblem can be separated into independent problems, we also make a comparison between solving these independent problems individually and solving them together as one large problem; referred to as solving the split or no-split subproblem, respectively.

The comparison of cut-strengthening techniques will be performed by evaluating the run time and the size of the strengthened cuts.

The LBBD scheme is implemented in Python 3.8, and the MIP and CP models are solved using Gurobi Optimizer version 9.1.2 and IBM ILOG CP Optimizer version 20.1, respectively. All tests have been carried out on a computer with two

Intel Xeon Gold 6130 processors (16 cores, 2.1 GHz each) and 96 GB RAM. Each instance was given a total time of 20 minutes and the MIP-gaps are set to 0 for the master problems.

3.6.1 Instances

The instances are either taken from the literature or generated in line with descriptions in the literature, but with new parameter settings. All instances can be accessed, either directly or via reference, from our repository[1]. For the cumulative facility scheduling problem with fixed costs, referred to as Problem 3.4.1, we use 336 instances from (Hooker, 2007). For the single machine scheduling problem with sequence-dependent setup times and multiple time windows, referred to as Problem 3.4.2, we use instances generated based on the description in (Coban & Hooker, 2013). For the vehicle routing problem with location congestion, referred to as Problem 3.4.3, we use 450 instances from (Lam et al., 2020).

3.6.2 Computational effectiveness

To evaluate the effectiveness of the cut-strengthening techniques we first look at their impact on the run time of the LBBD scheme. We then look at the average size of the cuts generated by the cut-strengthening techniques. Additional data is provided in Tables B.1–B.10 in Appendix B.

For the run time plots, the horizontal axis gives the time and the vertical axis gives the percentage of solved instances. A point (x, y) on the curve means that $y\%$ of instances can be solved in less than x seconds. The cut size figures are histograms, a point (x, y) means average cut size x has frequency y. Where the frequency y is the number of generated cuts. Since the type of a value cut does not influence the size of the cut, we only differentiate the histograms by the type of the subproblem. In the figures below, the additive/deletion filter and no cut strengthening are referred to as *adel* and *none*, respectively.

[1] https://gitlab.liu.se/eliro15/lbbd_instances

The experiments evaluating the effect of cut strengthening on analytic cuts show results very similar to results for strengthened optimality cuts. For the sake of brevity, the results and discussion related to analytic cuts have been moved to Appendix A.

3.6.2.1 Minimising makespan for cumulative facility scheduling problem

It can be seen in the Figure 3.2a for the split subproblem that the most instances are solved to optimality when applying the deletion filter and DFBS. Specifically, 65.48% and 64.29% of instances, respectively. This result is closely followed by additive/deletion filter and greedy method with very similar profiles in Figure 3.2a and the respective percentages of solved instances of 63.99% and 63.39%. An analysis of the experiment data shows that the additive/deletion filter and DFBS spend more time solving subproblems than deletion filter. The additive/deletion filter and DFBS respectively spend 107.05 seconds and 113.21 seconds of run time, while deletion filter spends 81.17 seconds. This might suggest that the random order of variables is better suited for the search strategy of deletion filter. Similar to deletion filter, the greedy method spends 84.19 seconds. However, Figure 3.3 shows that the greedy method generates cuts of higher density, which lead to a master problem that is harder to solve.

Interestingly, applying cut-strengthening techniques to the no-split subproblem has the same impact on the results as splitting the subproblem with no cut strengthening. The DFBS, deletion filter, additive/deletion filter, and additive method used for the no-split subproblem, with the respective percentages of solved instances of 58.93%, 54.17%, 56.25%, and 54.46%, achieve similar results to splitting the subproblem and applying no cut strengthening with 54.46% of solved instances. The weak effectiveness of the greedy algorithm for the no-split subproblem can be explained by the high density of the cuts, as can be seen in Figure 3.4. The cuts generated by the greedy algorithm are less sparse compared to other techniques. They substantially increase the number of iterations and the master problem run time. In other words, the time spent strengthening the cuts using the greedy algorithm does not pay off.

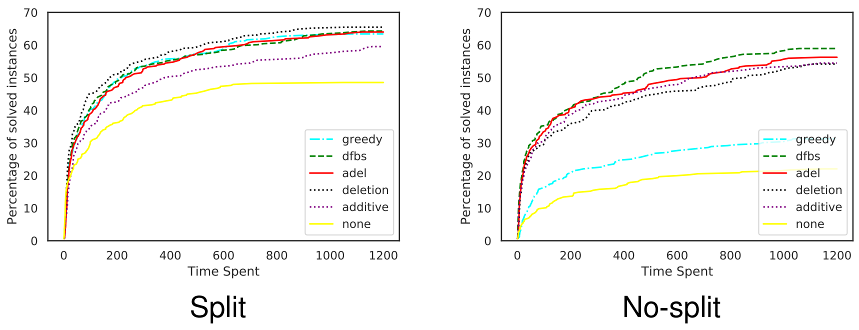


Figure 3.2: Cumulative scheduling: Percentage of solved instances for minimising makespan problem for with strengthened optimality cuts

3.6.2.2 Minimising tardiness for cumulative facility scheduling problem

The first thing to notice in the run time Figure 3.5 is that the split subproblem gives significantly higher results than the no-split subproblem. The best effectiveness for the no-split subproblem is shown in Figure 3.5b by the deletion filter with 7.44% of solved instances. However, this result is considerably different to effectiveness when splitting the subproblems, with the lowest result of 25.6% of solved instances that corresponds to no cut strengthening. The deletion filter shows the best result when splitting the subproblem with 37.2% of instances solved. This result is followed by additive/deletion, DFBS, and additive with 36.31%, 36.12%, and 35.71% of solved instances, respectively. The greedy algorithm with 30.36% of

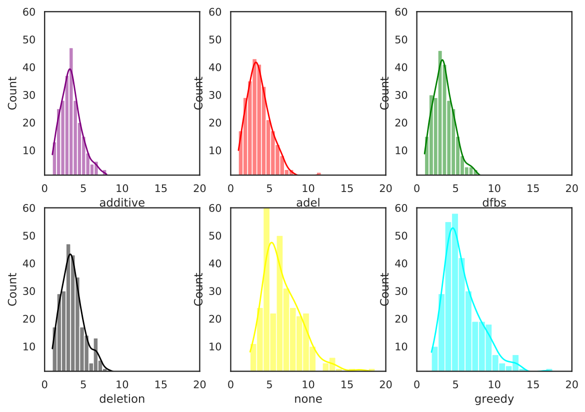


Figure 3.3: Cumulative scheduling: Average size of optimality cuts for the minimising makespan problem with split subproblem

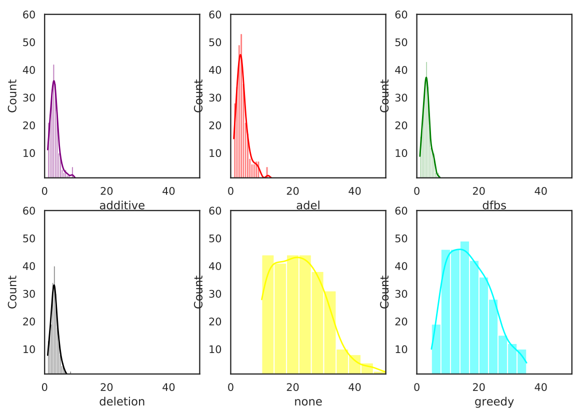


Figure 3.4: Cumulative scheduling: Average size of optimality cuts for the minimising makespan problem for with no-split subproblem

solved instances is notably behind other algorithms. The results in Figure 3.6 show that all of the algorithms except for the greedy method are similar in terms of the average size of the cuts. The median values of number of variables per cut for DFBS, additive/deletion filter, additive, and deletion filter are 3.30, 3.31, 3.32, and 3.37, respectively. Whereas, the greedy method and no cut strengthening have respective median values of 4.44 and 4.65 variables per cut. The nature of the objective function is likely to be the reason of the poor performance of the greedy algorithm. The greedy algorithm does not remove many assignments in the search process before the objective value of the subproblem decreases. This leads to the cut not having reduced its size. The generated dense cuts then lead to the increased number of Bender's iterations.

The relative effectiveness for each of the strengthening technqiues is similar for both the strengthened and analytic cuts. However, the results presented in Appendix A.2 demonstrate that analytic cuts for the split subproblem perform relatively poorly with less instances solved to optimality compared to strengthened optimality cuts, an observation that is also reported by Hooker (2007).

3.6.2.3 Minimising total cost for cumulative facility scheduling problem

Similar to the previous experiments, a considerable rise in computational effectiveness of the LBBD scheme comes from splitting the subproblem, as can be seen

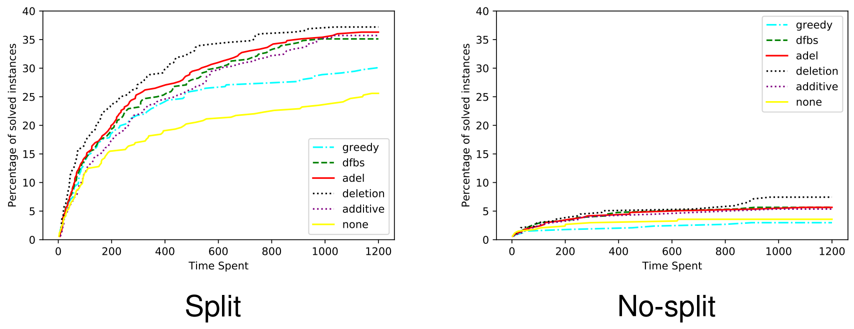


Figure 3.5: Cumulative scheduling: Percentage of solved instances for minimising tardiness problem with strengthened optimality cuts

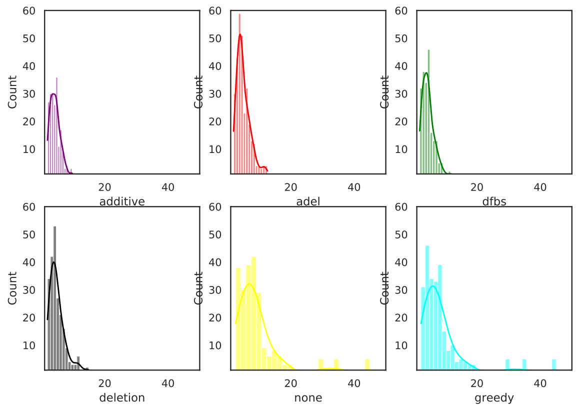


Figure 3.6: Cumulative scheduling: Average size of optimality cuts for the minimising tardiness problem with split subproblem

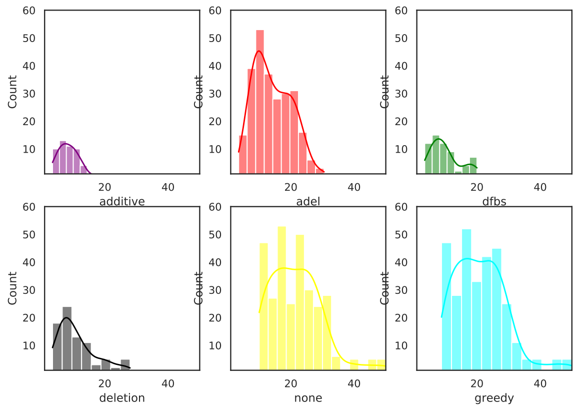
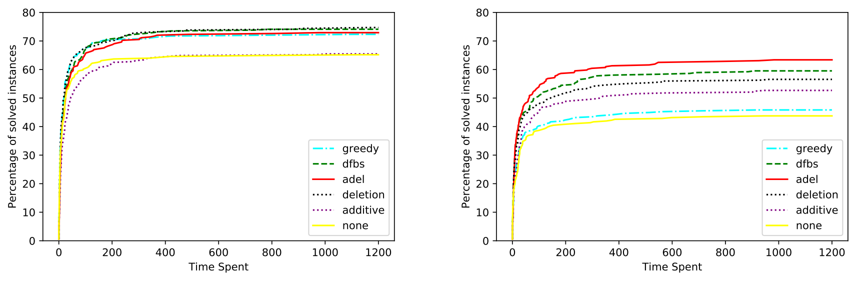


Figure 3.7: Cumulative scheduling: Average size of optimality cuts for the minimising tardiness problem with no-split subproblem

in run time Figure 3.8. At the same time, applying cut-strengthening to the split subproblem still gives a significant boost to the results: the deletion filter, DFBS, additive/deletion filter, and the greedy algorithm with the respective percentages of 74.70%, 74.11%, 72.91%, and 72.32% outperform no cut strengthening and additive method with 65.17% and 65.47% of solved instances, respectively. Interestingly, Figure 3.9 shows that DFBS, deletion filter, and additive/deletion filter have very similar histograms of average cut sizes with the same median value of 4. The median values for the additive method, greedy method, and no cut strengthening are 5.67, 6.38, and 7.52 variables per cut, respectively. Although the greedy algorithm generates denser cuts than the additive method, the number of subproblems it performs is lower. The greedy algorithm and the additive method solve 59.3 and 163.1 subproblems, respectively. This is likely to be the reason of the relatively good performance of the greedy algorithm. Overall, the results for the most of the cut-strengthening techniques for the split-subproblem do not differ much in terms of run time. In contrast to the split subproblem, the difference in effectiveness



Split

No-split

Figure 3.8: Cumulative scheduling: Percentage of solved instances for minimising total cost problem with strengthened feasibility cuts

is more pronounced for the no-split subproblem. The additive/deletion method for the no-split subproblem outperforms all other cut-strengthening techniques with 63.39% of instances solved to optimality. The additive/deletion filter, DFBS, and the deletion filter have similar sparsity of the feasibility cuts, as shown in Figure 3.10. However, additive/deletion filter has a lower number of subproblems solved per instance compared to the deletion filter and DFBS. This indicates that additive/deletion filter is successful in removing variables not contributing to infeasibility early on. The effectiveness of the additive/deletion filter also compares to the split subproblem with no cut strengthening.

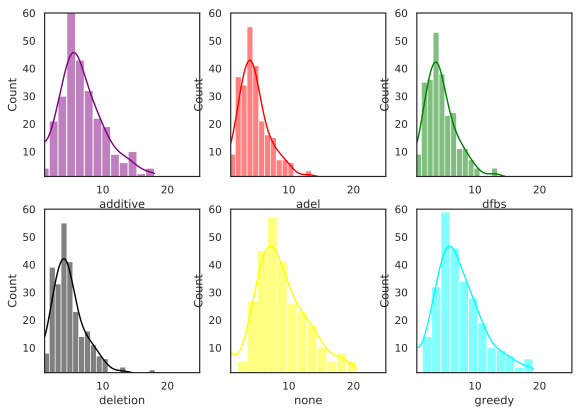


Figure 3.9: Cumulative scheduling problem: Average size of feasibility cuts for the minimising total cost with split subproblem

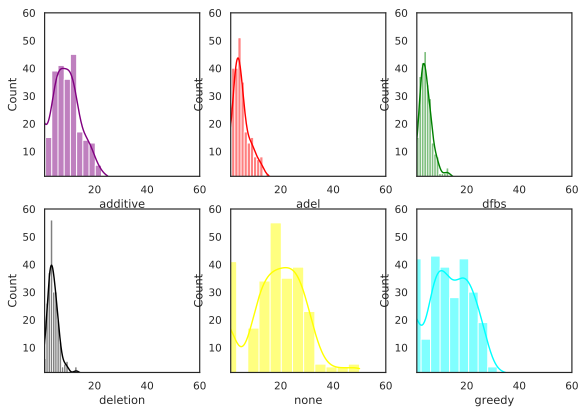
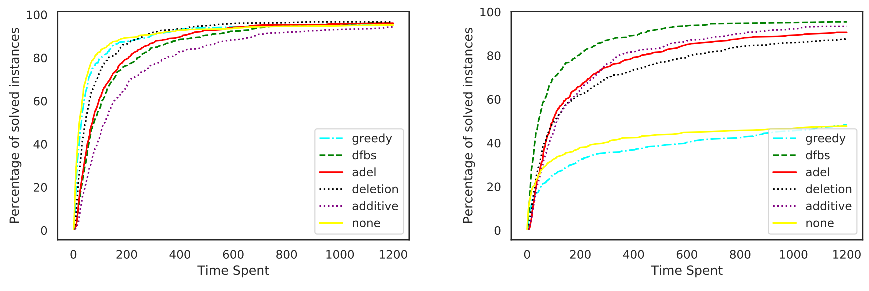


Figure 3.10: Cumulative scheduling: Average size of Feasibility cuts for the Minimising Total Cost problem with No-Split Subproblem

3.6.2.4 Minimising makespan for disjunctive scheduling problem

The run time results in Figure 3.11a show that using cut-strengthening techniques does not accelerate the LBBD scheme for the split subproblem. In fact, using the greedy method or no cut strengthening outperforms all of the other techniques. The results in Figure 3.12 show that on average all of the techniques fail to reduce the size of the cuts. This implies that master variables are already split into minimal subsets and cannot be reduced further. Therefore the time spent on cut strengthening is not compensated, as can be seen in the run time profiles in Figure 3.11a.



Split

No-split

Figure 3.11: Disjunctive scheduling: Percentage of solved instances for minimising makespan with strengthened optimality cuts

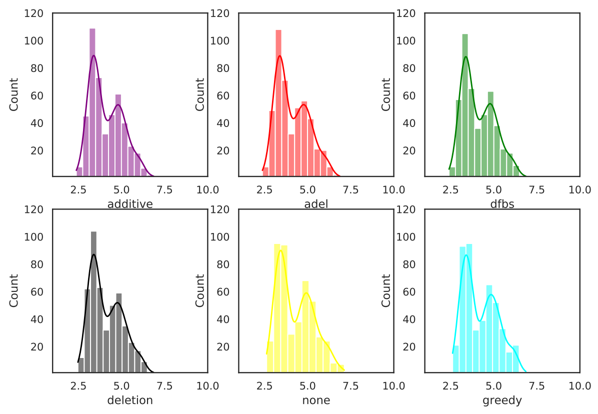


Figure 3.12: Disjunctive scheduling: Average size of optimality cuts for the minimising makespan problem with split subproblem

By contrast, Figure 3.13 shows that all of the cut-strengthening techniques, except for the greedy, give a significant rise in effectiveness for the no-split subproblem. This can be explained by significant difference in the sparsity of the

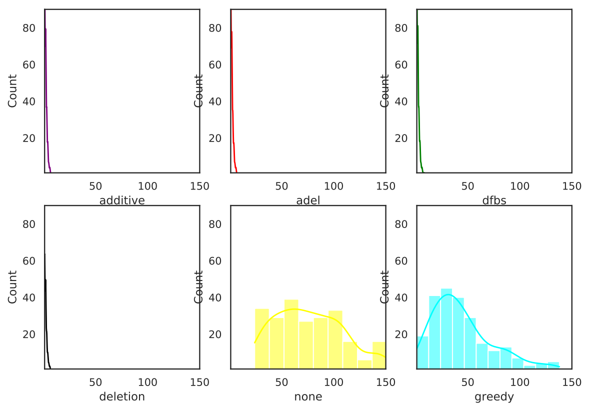


Figure 3.13: Disjunctive scheduling: Average size of optimality cuts for the minimising makespan problem with no-split subproblem

generated cuts. The DFBS, deletion filter, additive/deletion filter, and the additive method have the median of 2 for average cut size, compared to 38 and 75 for the greedy algorithm and no cut strengthening, respectively. The sparsity of the irreducible cuts can be explained by the fact that only one job at a time can be processed, and the suboptimality of the solution is likely to be caused by the few jobs with the greatest processing end times. The greedy algorithm generates cuts that are stronger than the original, however, they are not strong enough to compensate for the time spent on the search. On the other hand, the DFBS is by far the best-performing cut-strengthening technique. DFBS solves 95.41% of instances, which is higher than 95.20% using no cut strengthening for the split subproblem. This can be explained by the low number of iterations and subproblems solved: DFBS solves 28.28 subproblems in 4.91 iterations, compared to 49.38 subproblems in 5.14 iterations by additive/deletion filter and 90.94 subproblems in 5.02 iterations by deletion filter. This result suggests that the search strategy employed by the DFBS is more effective compared to the deletion filter and the additive/deletion filter. The comparison of the split and no-split results shows that DFBS can be used instead of splitting the subproblem, and vice versa.

3.6.2.5 Minimising tardiness for disjunctive scheduling problem

The most noticeable observation from the run time Figure 3.14 is that cut strengthening is not an effective way to accelerate the solution. Only splitting the subprob-

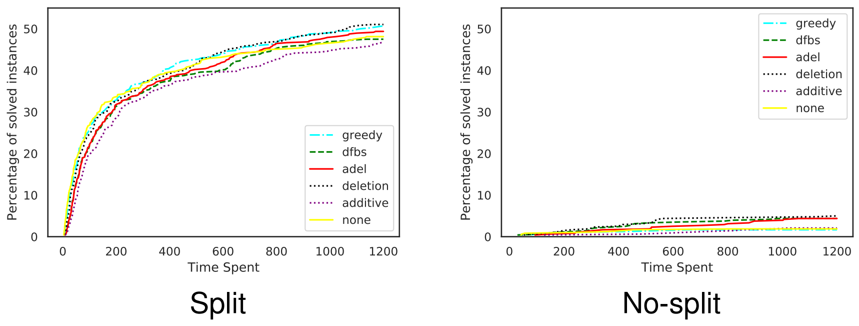


Figure 3.14: Disjunctive scheduling: Percentage of solved instances for minimising tardiness with strengthened optimality cuts

lem makes a meaningful impact on the ability to solve the instances. When using no cut strengthening, splitting the subproblems solves 48.13% of the instances to optimality. This is compared to 1.88% of instances solved to optimality when no splitting of the subproblems is performed. Similar to previous experiments, the greedy algorithm reduces the cut density the least, as can be seen in Figure 3.16. It can also be seen that although the additive method generates sparse cuts, it fails to solve many instances. This is due to the long time it takes to find the irreducible feasibility cuts. Another striking observation is that the results presented in Figure 3.15 for the split subproblem are similar for all of the cut-strengthening techniques including no cut strengthening. This suggests that, as in the case for the minimising makespan problem, the master variables are already split into minimal subsets.

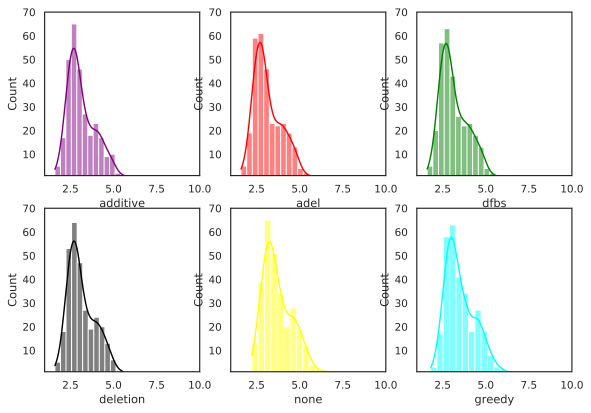


Figure 3.15: Disjunctive scheduling: Average size of optimality cuts for the minimising tardiness problem with split subproblem

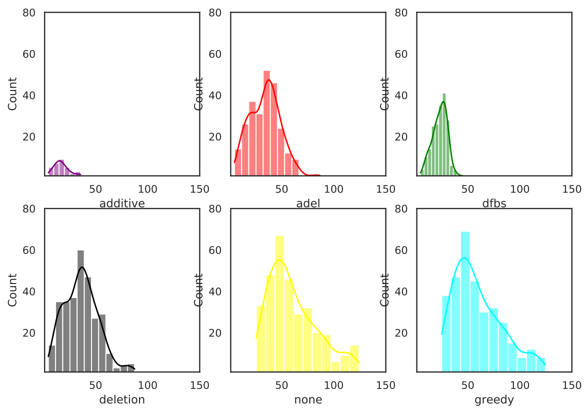


Figure 3.16: Disjunctive scheduling: Average size of optimality cuts for the minimising tardiness problem with no-split subproblem

3.6.2.6 Finding a feasible schedule for disjunctive scheduling problem

As can be seen from the comparison of the run time in Figure 3.17, splitting the subproblem is more effective in accelerating the solution process than cut strengthening. Within 9.98 seconds, all of the cut-strengthening techniques, including none, solve at least 87.7% of instances. Nevertheless, cut-strengthening techniques still have a substantial impact on the solution process when the subproblem is not split. Similar to the results for the minimising makespan problem, all of the cut-strengthening techniques except for the greedy method increase the percentage of solved instances compared to using no cut strengthening by at least 39.79%. The most effective algorithms for the no-split subproblem are DFBS and additive method with respective percentages of solved instances of 94.38% and 92.71%, compared to 47.92% with no cut strengthening. The DFBS tends to generate the most sparse cuts. The median value of the average cut size for DFBS is 3.33 variables per cut, compared to 3.4 for deletion filter and the additive method, and 3.44 for additive/deleton filter. The DFBS also spends much less time on average solving the subproblem — only 31.34 seconds compared to the next closest result of 62.22 seconds by the additive method. That implies that DFBS detects an irreducible set of variables much faster than other techniques. This result suggests that the search method used by DFBS is better suited for this problem compared to the other techniques under investigation.

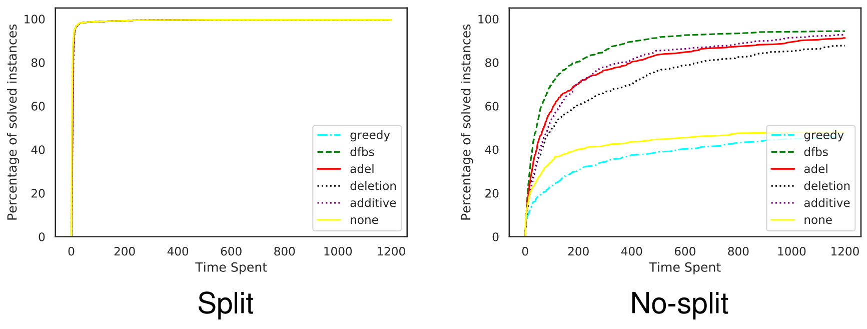


Figure 3.17: Disjunctive scheduling: Percentage of solved instances for finding a feasible schedule problem with strengthened feasibility cuts

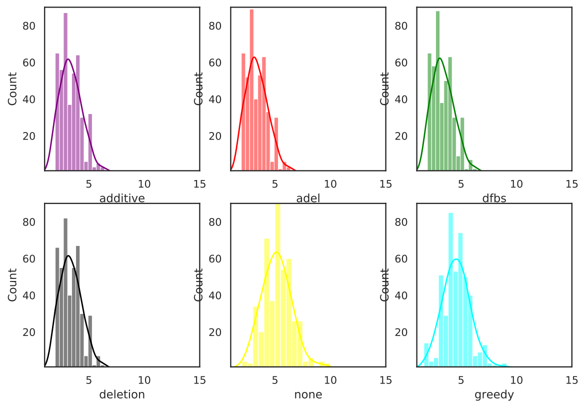


Figure 3.18: Disjunctive scheduling: Average size of feasibility cuts for the finding a feasible schedule problem with split subproblem

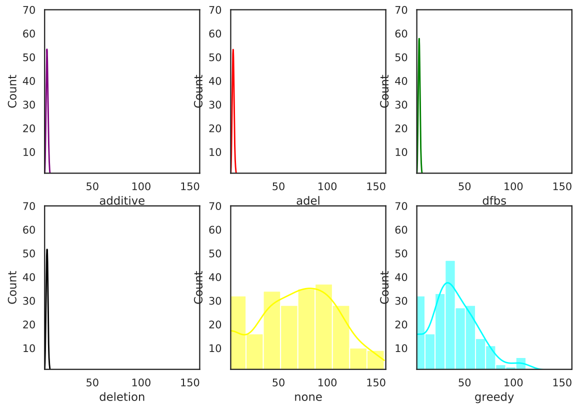
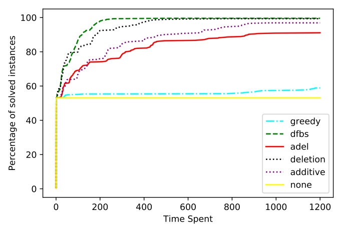


Figure 3.19: Disjunctive scheduling: Average size of feasibility cuts for the finding a feasible schedule problem with no-split subproblem

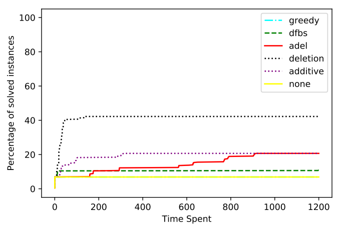
3.6.2.7 Minimising makespan for vehicle routing problem

The first thing to notice in Figure 3.20a is that 53.11% of instances terminate after a single iteration regardless of the cut-strengthening technique applied. The reason is that the instances either have an infeasible master problem or an infeasible subproblem. The infeasibility of the master problem detected before adding any feasibility cuts, and the infeasibility of the subproblem is detected without any restriction on the master problem variables. The subproblem infeasibility is checked before the initialisation of the LBBD scheme.

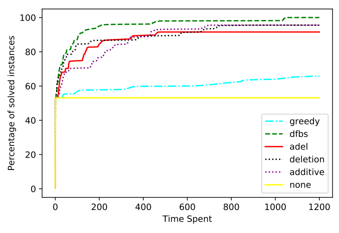
When using no cut strengthening, no instances are solved to optimality. Although the greedy algorithm improves that result by solving 59.33% of instances to optimality, it still performs poorly compared to other cut-strengthening techniques. Interestingly, cut-strengthening techniques with similar search processes show similar results. The additive/deletion filter and additive method solve 91.11% and 88.89% of instances to optimality, respectively. The deletion filter and DFBS are the most effective, solving 99.55% of instances to optimality. Figure 3.21 shows that the average size of the generated cuts is similar for the deletion filter, DFBS, additive/deletion filter, and the additive method. However, the deletion filter tends to have fewer Bender's iterations than other cut-strengthening techniques — 30.68 iterations compared to 62.64, 65.27, and 65.73 by DFBS, additive/deletion filter, and additive method, respectively. This suggests that the cuts generated by the deletion filter are more effective. At the same time, DFBS has a smaller number of subproblems solved — 697.14 compared to 1344.53, 1475.92, and 2676.89 when using additive/deletion filter, deletion filter, and additive method, respectively. That implies that DFBS generates less effective cuts, but in a shorter time. Therefore the time spent on a greater number of iterations is offset by smaller subproblem solution time per iteration.



Makespan



Tardiness



Total Travel Time

Figure 3.20: Vehicle routing problem: Percentage of solved instances for minimising makespan, minimising tardiness, and minimising total travel time problems

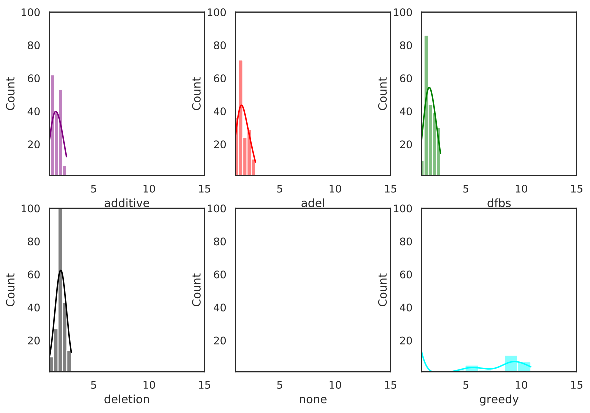


Figure 3.21: VRPLC: Average size of feasibility cuts for the minimising makespan problem

3.6.2.8 Minimising total tardiness for vehicle routing problem

As can be seen in Figure 3.20b the share of infeasible instances is smaller than for the minimising makespan problem. This is due to jobs being allowed to run past their deadlines. The job deadlines would otherwise cause infeasibility of valid inequalities in the master problem. All of the 6.89% of instances that terminate after a single iteration have an infeasible subproblem. When using no cut strengthening or the greedy method, no instances are solved to optimality. DFBS, surprisingly, has a small impact on the computational performance and solves 10.44% of instances. Somewhat better results are shown by the additive method and additive/deletion filter, with both solving 20.67% of instances. However, the additive method has a better run time profile than the additive/deletion filter. This is likely due to a greater number of iterations performed by additive/deletion filter — 197.71 compared to 61.71 by additive method. The deletion filter significantly boosts the performance with 41.33% of instances solved. Figure 3.22 shows that the DFBS, deletion filter, and the additive method have a similar average cut size. However, the deletion filter solves fewer subproblems — 820.79 compared to 3823.91 and 6036.35 by additive method and DFBS, respectively. Moreover, the deletion filter performs the smallest number of iterations — 27.06 compared to 61.71, 165.76, and 197.91 by additive method, DFBS, and additive/deletion filter, respectively. That implies that the deletion filter generates stronger cuts, and its search method is the best suited for the given instances.

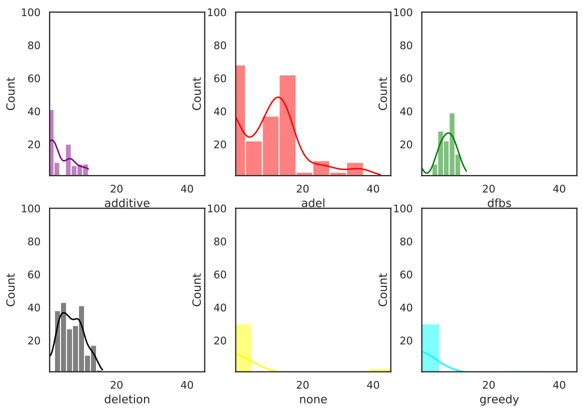


Figure 3.22: VRPLC: Average size of optimality cuts for the minimising tardiness problem

3.6.2.9 Minimising total travel time for vehicle routing problem

Similar to the minimising makespan problem, 53.11% of instances are detected to be infeasible after a single iteration. No other instances are solved to optimality when using no cut strengthening, as can be seen in Figure 3.20c. The greedy method improves this result by 12.66% with 65.77% of solved instances. Notably, other techniques under investigation show substantial improvement in computational effectiveness. Namely, using DFBS solves 100% of the given instances. This result is followed by deletion filter and additive method both solving 95.55% of instances. Although the additive/deletion filter solves 91.55% of instances, it has a better run time profile than the additive method. This is due to a lower number of iterations and subproblems solved — 20.99 iterations and 533.29 subproblems compared to 26.36 and 1510.15 by the additive method. All of these four cut-strengthening techniques have similar results in terms of the average cut size. However, DFBS on average solves 286.65 of subproblems — much lower number than the other three techniques. That implies that DFBS is more efficient in generating similar cuts. Overall, we note that DFBS, deletion filter, additive/deletion filter, and additive method generate similar cuts and the marginal difference in run time is caused by the different efficiency of the search strategies.

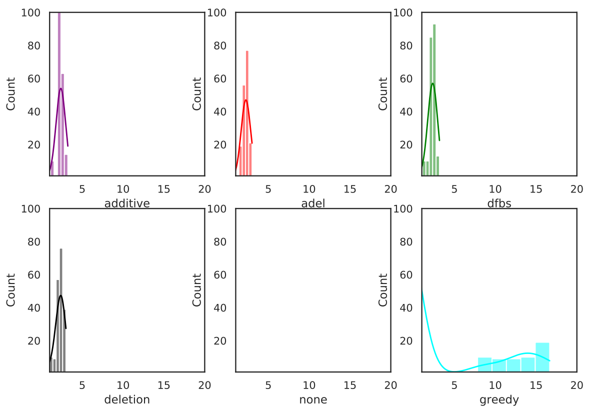


Figure 3.23: VRPLC: Average size of feasibility cuts for the minimising total travel time problem

3.7 Conclusions

This chapter investigates the impact of various cut-strengthening techniques on the computational effectiveness of the LBBD scheme. Namely, the greedy algorithm, deletion filter, additive method, additive/deletion filter, and the depth-first binary search were evaluated. The evaluation is based on computational experiments that solve the cumulative facility scheduling problem, the single-facility scheduling with a segmented timeline problem, and the vehicle routing with local congestion problem. The three types of problems with various objective functions have been solved applying the cut-strengthening techniques within the LBBD scheme. The previous work of Karlsson and Rönnberg (2021) is extended by including problem formulations that require generating both feasibility and optimality cuts. Additionally, the cut-strengthening techniques were applied to two types of value cuts—no-good optimality cuts and analytic Benders' cuts. For the problem formulations that allow subproblem separation, the comparison is made between applying the cut-strengthening techniques to the separated subproblem and applying them to one large subproblem.

When summarising the results, the following can be observed. The cut-strengthening techniques, such as DFBS, deletion filter, additive/deletion filter and additive, that generate irreducible cuts tend to outperform the greedy algorithm and no cut strengthening. This is mostly due to sparsity of the irreducible cuts. We also note that the effectiveness of the greedy algorithm depends on the type of the objective function. Overall, DFBS and deletion filter have the best computational effectiveness. Although DFBS and deletion filter generate cuts of the same size and strength as additive/deletion filter and the additive method, they solve lower number of subproblems. Therefore, DFBS and deletion filter have lower subproblem solution time. This implies that DFBS and deletion filter are more efficient in identifying irreducible cuts.

Another observation is that the separation of the subproblem does not always strongly dominate cut strengthening in terms of benefiting the solution process. In fact, there are cases when splitting the subproblem and applying cut strengthening are interchangeable. Therefore some of the techniques can benefit the solution process significantly even if the subproblem is not separable. For example, the vehicle routing problem does not naturally separate the subproblem, and the computational results show that it is imperative to use cut strengthening.

The results show that the difference in performance between the cut-strengthening techniques comes from the efficiency of the search strategy. It would be relevant to investigate the impact of the order of variables on the search strategy. Another observation is that the cut-strengthening techniques have different results depending on the type of the objective function. It may also be worth analysing how the objective function could be exploited to improve the effectiveness of the techniques.

Chapter 4

Increasing computational efficiency of depth-first binary search using variable sorting

4.1 Introduction

As discussed in the previous chapter, depth-first binary search (DFBS) is one of the most computationally effective cut-strengthening techniques for the LBBD scheme. However, the results showed that the effectiveness of a cut-strengthening technique does not only depend on the size of a generated cut, but it is strongly correlated with the number of subproblems solved to generate the cut. It was also observed that the random order of variables can negatively affect the efficiency of DFBS and lead to a higher number of subproblems solved. This chapter introduces new strategies to improve computational efficiency of the DFBS algorithm.

In this thesis, computational effectiveness of a cut-strengthening technique in LBBD is assessed based on the number of instances solved to optimality, the size of the generated cuts, and the number of Benders' iterations. Computational effectiveness is a measure of an impact that a cut-strengthening technique has on the overall LBBD scheme. Computational efficiency of a cut-strengthening technique, on the other hand, is assessed based on the subproblem solution time and the number of subproblems solved. Computational efficiency measures the time and the resources a cut-strengthening technique requires to generate a strengthened cut. Both measures are important in evaluating the overall quality of a technique, and there is a trade-off between effectiveness and efficiency.

In the previous chapter, the DFBS algorithm has been evaluated in its simplest implementation, where the order of variables was random, and the set of variables has always been divided into two equal groups. This chapter is therefore mainly concerned with evaluating the impact of the order of the variables and the size of variable groups on the efficiency of search strategy.

This chapter is structured as follows: An overview of the most relevant implementations and modifications of DFBS in the literature is presented in Section 4.2. Section 4.2 also presents a variable ordering heuristic that inspired one of the new strategies proposed in this chapter. Section 4.3 presents the preliminary investigation of the new variable sorting strategies. Three new heuristics based on these strategies are then formalised in Section 4.4. Section 4.5 evaluates the effectiveness and efficiency of the proposed methods based on computational experiments. The summary is provided in Section 4.6.

The contributions of this chapter are:

- the preliminary investigation of the variable-sorting strategies,
- the introduction of novel variable-sorting and variable-grouping heuristics,
- the computational evaluation of the new strategies.

4.2 Related literature

There are many depth-first search algorithms, and they are not exclusive to the optimisation community. This section, however, focuses on the literature that is relevant to the implementation of the DFBS algorithm, which is evaluated in Chapter 3.

4.2.1 Depth-first binary search as a filtering algorithm

Guieu and Chinneck (1999) presented several filtering algorithms such as the deletion filter, the additive method, and the additive/deletion filter, in their work on infeasibility of mixed-integer linear programs (MILP) and integer linear programs

(ILP). The algorithms are applied to identify an irreducible infeasible subset of constraints of an infeasible problem. They proposed several methods of improving the computational efficiency of the algorithms—by dynamically reordering the constraints, by grouping constraints, by replacing the original objective function with a function that decides feasibility status more rapidly, and other possibilities. The grouping of constraints with the fixed group size of four constraints is assessed for the deletion filter, the additive method, and the additive/deletion filter. The authors recommend this method when it is important to find an isolation quickly. DFBS can be viewed as a modification of the deletion filter that applies grouping of constraints.

Atlihan and Schrage (2008) generalise the idea of grouping constraints described in Guieu and Chinneck (1999) by using the binary search to isolate an IIS in infeasible LP, MILP, and general nonlinear programs (NLP). They propose a binary search algorithm that evaluates groups of constraints instead of one constraint at a time. In its simplest form the algorithm splits the set of constraints in half. Once a subset is known to be infeasible, the algorithm continues to 'pursue' that subset, while dropping the other group of constraints. The version of the DFBS algorithm proposed by Atlihan and Schrage (2008) is adapted as a cut-strengthening technique in the previous chapter, see pseudo-code in Algorithm 8. Based on their empirical tests, Atlihan and Schrage (2008) suggest that the algorithm can be improved by using alternative strategies for splitting the constraint sets into groups. One idea is to use a priority queue based on the estimates of a constraint being in an IIS. This idea is similar to the new variable sorting strategies proposed in this chapter.

Work on DFBS has also been done for constraint satisfaction programs and constraint programs. Junker (2004) proposes DIVIDE-AND-CONQUER, an algorithm similar to DFBS, as a part of the conflict detection method QUICKXPLAIN. The algorithm finds an irreducible subset of constraints causing infeasibility of a constraint programming problem. QUICKXPLAIN then produces a no-good cut containing variables present in the irreducible subset. Junker suggests that the algorithm could be improved by making more informed decisions on the grouping of constraints based on the connections between constraints.

4.2.2 DFBS in the LBBD scheme

Cambazard et al. (2004) apply QUICKXPLAIN to generate strong feasibility cuts in their LBBD scheme, which solves a hard real-time task allocation problem. The problem assigns real-time tasks with fixed priorities to distributed processors. It can be decomposed into allocation and scheduling problems. QUICKXPLAIN is used to determine the minimal set of tasks causing inconsistency to generate stronger no-good cuts in order to improve communication between the master problem and subproblem.

Karlsson and Rönnberg (2021) evaluate DFBS as one of the cut-strengthening techniques for feasibility cuts generated within an LBBD scheme. Their study considers the cumulative facility scheduling with fixed costs, single machine scheduling with sequence-dependent setup times and multiple time windows, and vehicle routing with local congestion problems. The authors conclude that DFBS is the most computationally effective technique and suggest investigating the impact of variable sorting on DFBS. In a different work, Karlsson and Rönnberg (2022) integrate the DFBS algorithm into their partial assignment acceleration technique within an LBBD scheme for the avionics scheduling problem. The implementation of DFBS in both papers is based on the algorithm proposed in Atlihan and Schrage (2008). To the best of my knowledge, these are the only works that presented results for DFBS in the LBBD related literature.c

4.2.3 Heuristic *dom/wdeg*

The variable sorting presented in this chapter is based on the conflict-directed variable ordering heuristic *dom/wdeg* proposed in Boussemart et al. (2004). The authors are interested in solving constraint satisfaction problems (CSPs). As mentioned in Chapter 2, solving a CSP involves either determining the infeasibility of the problem or finding one (or more) solution. The variable ordering heuristic is proposed to aid the search process towards hard or inconsistent parts of a CSP.

The *dom/wdeg* heuristic is a combination of two separate heuristics— *dom* and *wdeg*. The *dom* (Haralick & Elliott, 1980) heuristic is a well-known dynamic variable ordering heuristic. Dynamic in this context means that the order of variables changes along the search, and the current state of the search is taken into account. The heuristic orders the variables according to the current size of their domains. No single heuristic clearly outperforms other heuristics, but dynamic heuristics are usually considered to be the most efficient (Boussemart et al., 2004).

The limitations of traditional dynamic heuristics such as *dom* is that they only look at the current state of the search, without taking into account the previous states of the search. Boussemart et al. (2004) propose to capture such information by associating a *weight* counter with each constraint of the problem. The counters are updated whenever the search finds an inconsistency. The proposed variable ordering heuristic is denoted *wdeg*. The counter of a constraint is increased by 1 when it belongs to an inconsistent part of CSP. As the search progresses, the weights of hard constraints increase, the constraints with greatest weights then direct the search toward the inconsistent part of CSP. This is what makes the heuristic ”conflict-directed”. The variables in *wdeg* are ordered according to their *weighted degrees*. The degree of a variable is the number of constraints where it is involved. Similarly, the *weighted degree* of a variable is a sum of weights of the constraints containing the variable.

The authors suggest implementing the heuristic within a depth-first search algorithm applied to solve a CSP. At each step of the search process the algorithm performs a variable assignment followed by constraint propagation. Intuitively, by the so called fail-first principle, the variables with greatest weighted degrees lead the search to the hard parts of CSPs (Boussemart et al., 2004). Combining *dom* and *wdeg* results in a heuristic that prioritises the variables with the smallest ratio of current domain size to current weighted degree.

Hemery et al. (2006) use *dom/wdeg* to prioritise constraints in order to extract minimal unsatisfiable cores (MUCs) from constraint networks. As mentioned in Chapter 2, extracting the MUCs from CSPs is similar to isolating IIS of MIPs. Therefore, *dom/wdeg* could potentially be applied to MIPs.

The results presented both in Boussemart et al. (2004) and Hemery et al. (2006) confirm that the number of times each constraint is violated during the search is important information that can be used to locate the inconsistent part of a CSP. Similar to filtering algorithms and other ideas that are applicable to both CSP and MIP, this result can be transferred to MIPs.

4.3 Preliminary investigation

The previous implementations of DFBS within the LBBD scheme in the literature, and in this thesis, used a random order of variables. The natural question is what impact variable sorting would have on the efficiency of the algorithm. The goal of this section is to introduce two types of variable sorting strategies that will be proposed as heuristics in the following sections. The first strategy is based on an idea similar to the *dom/wdeg* heuristic. The idea behind the second strategy is to use the problem structure and data. The argument for the use of the new strategies is constructed based on the evidence provided by computational experiments. The computational experiments run the default implementation of the DFBS algorithm.

The computational experiments solve the cumulative scheduling problem to minimise tardiness. The experiments in Chapter 3 showed that the tardiness instances tend to be the most difficult ones to solve. Therefore they seemed to be the most interesting choice. All of the instances are taken from Hooker (2007) with the same settings as in the previous chapter (the instances can be accessed via the reference). The number of jobs to be scheduled varies between 10, 12, 14, 16, and 18, and the number of machines goes from 2 to 4 machines.

The LBBD scheme is implemented in Python 3.8, and the MIP and CP models are solved using Gurobi Optimizer version 9.1.2 and IBM ILOG CP Optimizer version 20.1, respectively. All tests have been carried out on a computer with two Intel Xeon Gold 6130 processors (16 cores, 2.1 GHz each) and 96 GB RAM. Each instance was given a total time of 20 minutes and the MIP-gaps are set to 0 for the master problems.

4.3.1 Variable sorting

As described in Section 3.3.2.5, in each iteration of the DFBS algorithm, the goal is to find a new variable that belongs to an IIS. Since this requires repeatedly solving the subproblems, the speed of the algorithm depends on the number of subproblems solved. In each minor iteration of DFBS, shown in Algorithm 9, the best outcome is to remove as many variables as possible that do not belong to the IIS. One can note, intuitively, that the more of the IIS variables get selected into set T_1 as early in the search as possible, the better, see Example 2.

Algorithm 9 ”Minor iteration” of the DFBS cut-strengthening algorithm

```
1: Split $T$ into $T_1$ and $T_2$
2: $\bar{x}_j \leftarrow 1, \quad j \in S \cup T_1 \cup I$
3: let $v'_k$ be the new optimal value of $[SP(\bar{x})]$
4: if $v'_k = v_k$ then
5:     $T \leftarrow T_1$
6: else
7:     $S \leftarrow S + T_1$; $T \leftarrow T_2$
8: end if
```

Example 2 *The example presented in Figures 4.1– 4.2 illustrates DFBS applied to set $\mathscr{J} = \{1,2,3,4,5,6,7,8\}$. Let v be the objective value of [$SP(\mathscr{J})$]. DFBS is applied to identify an irreducible subset $I \subseteq \mathscr{J}$, such that the objective value of [$SP(I)$] is equal to v. In each iteration, the goal of DFBS is to identify a variable that belongs to an IIS, if the set of variables to consider is greater than one, then the minor iteration is called. Set T stores current candidates to be included in set I. Set S stores variables that are not current candidates, but can still be considered in the following iterations. Set T is initially equal to $\mathscr{J}$, and sets S and I are empty.*

We start by looking at the DFBS implementation that uses an order of variables *with no particular sorting in Figure 4.1.*

- *Iteration 1: Set T contains more than one variable, it is therefore split into $T_1 = \{1,2,4,6\}$ and $T_2 = \{3,7,5,8\}$, see Algorithm 9. First, the objective value v' of $[SP(T_1 \cup I \cup S)]$ is evaluated. Since v' is not equal to the original objective v, set T_1 is stored in set S, and set T_2 becomes the new set T.*

- *Iteration 2: Set T is then split into $T_1 = \{3,7\}$ and $T_2 = \{5,8\}$. The objective value v' of $[SP(T_1 \cup I \cup S)]$ is evaluated, since v' is not equal to v, $T_1 = \{3,7\}$ is added to set S, and T_2 becomes the new set T.*

- *Iteration 3: Set T is split into $T_1 = \{5\}$ and $T_2 = \{8\}$. The objective value v' of $[SP(T_1 \cup I \cup S)]$ is evaluated, v' is equal to v, therefore $T_1 = \{5\}$ is stored as new set T, and $T_2 = \{8\}$ is not considered in the following iterations.*

- *Iteration 4: Set $T = \{5\}$ only contains a single index, therefore it is added to I. Next, the objective value v' of $[SP(I)]$ is evaluated, v' is not equal to v, indices of set $S = \{1,2,4,6,3,7\}$ are therefore stored in set T, set S is then emptied.*

- *Iteration 5: Since set T contains more than one index, it is split into $T_1 = \{1,2,4\}$ and $T_2 = \{6,3,7\}$. The objective value v' of $[SP(T_1 \cup I \cup S)]$ is evaluated, v' is not equal to v, therefore set T_1 is added to set S, and set T_2 becomes the new set T.*

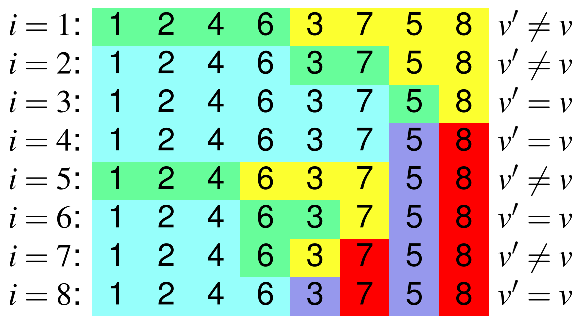
- *Iteration 6: Set T is split into sets $T_1 = \{6,3\}$ and $T_2 = \{7\}$. The objective value v' of $[SP(T_1 \cup I \cup S)]$ is evaluated, v' is equal to v, therefore set $T_1 = \{6,3\}$ becomes the new set T, and set $T_2 = \{7\}$ is no longer considered in the following iterations.*

- *Iteration 7: Set T is split into sets $T_1 = \{6\}$ and $T_2 = \{3\}$. The objective value v' of $[SP(T_1 \cup I \cup S)]$ is evaluated, v' is not equal to v, $T_1 = \{6\}$ is therefore added to set S, and $T_2 = \{3\}$ becomes the new set T.*

- *Iteration 8: Since set $T = \{3\}$ only contains one index, it is added to set*

$I = \{5\}$. The objective value v' of $[SP(I)]$ is evaluated, v' is equal to v, the search is complete with the irreducible set $I = \{3,5\}$.

It can be seen in Figure 4.1, the random order of variables leads to set S containing a big number of indices. For most of the search, only one variable is identified as an IIS member, and only one variable is removed from the search. This leads to a greater number of iterations and subproblems solved.



$i=1$:	🟩 1	🟩 2	🟩 4	🟩 6	🟨 3	🟨 7	🟨 5	🟨 8	$v' \neq v$
$i=2$:	🟦 1	🟦 2	🟦 4	🟦 6	🟩 3	🟩 7	🟨 5	🟨 8	$v' \neq v$
$i=3$:	🟦 1	🟦 2	🟦 4	🟦 6	🟦 3	🟦 7	🟩 5	🟨 8	$v' = v$
$i=4$:	🟦 1	🟦 2	🟦 4	🟦 6	🟦 3	🟦 7	🟪 5	🟥 8	$v' = v$
$i=5$:	🟩 1	🟩 2	🟩 4	🟨 6	🟨 3	🟨 7	🟪 5	🟥 8	$v' \neq v$
$i=6$:	🟦 1	🟦 2	🟦 4	🟩 6	🟩 3	🟨 7	🟪 5	🟥 8	$v' = v$
$i=7$:	🟦 1	🟦 2	🟦 4	🟩 6	🟨 3	🟥 7	🟪 5	🟥 8	$v' \neq v$
$i=8$:	🟦 1	🟦 2	🟦 4	🟦 6	🟪 3	🟥 7	🟪 5	🟥 8	$v' = v$

Figure 4.1: DFBS example with random sorting: T_1= 🟩, T_2=🟨, S=🟦, I=🟪

We now look at the DFBS implementation with a favourable order of variables.

- *Iteration 1: Assume set T is ordered by applying some variable sorting. Set T is then split into sets $T_1 = \{3,5,4,6\}$ and $T_2 = \{1,2,7,8\}$. The objective value v' of $[SP(T_1 \cup I \cup S)]$ is evaluated, v' is equal to the original objective v, set T_1 therefore becomes the new set T and, set T_2 is no longer considered in the following iterations.*

- *Iteration 2: Set T is split into sets $T_1 = \{3,5\}$ and $T_2 = \{4,6\}$. The objective value v' of $[SP(T_1 \cup I \cup S)]$ is evaluated, v' is equal to v, set $T_1 = \{3,5\}$ becomes the new set T, and set $T_2 = \{4,6\}$ is no longer considered in the following iterations.*

- *Iteration 3: Set T is split into sets $T_1 = \{3\}$ and $T_2 = \{5\}$. The objective value v' of $[SP(T_1 \cup I \cup S)]$ is evaluated, v' is not equal to v, set $T_1 = \{3\}$ is therefore added to set S, and set $T_2 = \{5\}$ becomes the new set T.*

- *Iteration 4: Since set $T = \{5\}$ only contains a single index, it is added to set I. The objective value v' of $[SP(I)]$ is then evaluated, v' is not equal to v, set*

$S = \{3\}$ become the new set T.

- *Iteration 5: Set $T = \{3\}$ only contains a single index, it is therefore added to set $I = \{5\}$. The objective value v' of $[SP(I)]$ is evaluated, v' is equal to v. The search is complete with the irreducible set $I = \{3,5\}$.*

It can be seen in Figure 4.2, that having elements of an IIS in set T_1 early on in the search significantly increases efficiency of the algorithm. Half of the variables are not considered in the search after the first iteration. In each iteration, at least one variable is either removed from the search or identified to be in the IIS. Set S only stored one index for a single iteration. The number of iterations is therefore lower than the number of iterations for the search with the random order of variables.

$i = 1$:	🟩 3	🟩 5	🟩 4	🟩 6	🟨 1	🟨 2	🟨 7	🟨 8	$v' = v$
$i = 2$:	🟩 3	🟩 5	🟨 4	🟨 6	🟥 1	🟥 2	🟥 7	🟥 8	$v' = v$
$i = 3$:	🟩 3	🟨 5	🟥 4	🟥 6	🟥 1	🟥 2	🟥 7	🟥 8	$v' \neq v$
$i = 4$:	🟦 3	🟪 5	🟥 4	🟥 6	🟥 1	🟥 2	🟥 7	🟥 8	$v' = v$
$i = 5$:	🟪 3	🟪 5	🟥 4	🟥 6	🟥 1	🟥 2	🟥 7	🟥 8	$v' = v$

Figure 4.2: DFBS example with favoured sorting: T_1= 🟩, T_2=🟨, S=🟦, I=🟪

The example highlights the importance of variable sorting. Using the right sorting strategy, e.g. sorting strategy that increases the chances of set T_1 containing all of the elements of an IIS, is expected to increase the efficiency of the algorithm.

4.3.2 Variable weight counter

This section introduces the idea of using a variable weight counter in variable sorting for DFBS. The variable weight counter is the key component of the heuristic *weights* proposed in Section 4.4.

A somewhat simplified version of the *dom/wdeg* heuristic can be used to sort variables in DFBS. The main idea of the *dom/wdeg* heuristic is to use the information about the number of times a constraint was violated to direct the search to 'difficult' parts of CSP. Similarly, the number of times a variable was present in an irreducible cut can be used to direct DFBS to the subset of variables likely to contain the irreducible cut elements.

Since all of the variables in the original cut have value 1, domains of the variables do not differ throughout the search, therefore the *dom* heuristic is not applicable to variable sorting. The *wdeg* part of the *dom/wdeg* heuristic can be applied as a simple variable weight counter. The variable weight counter is defined as follows. Each variable has its own weight counter. At the start of the LBBD solution process, all of the weight counters are initialised to 0. Then, in each iteration of the LBBD, the weight counter of a variable is increased by 1 when the variable belongs to the irreducible cut. The variables with higher weights are then prioritised by the DFBS algorithm.

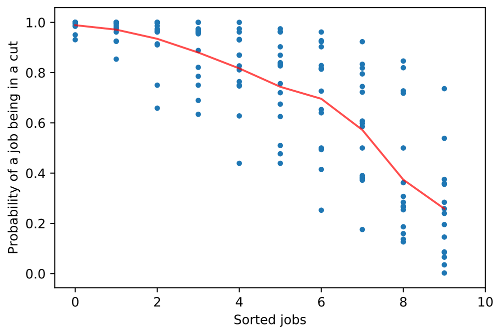
To validate the use of the weight counter for variable sorting, the following computational experiment is carried out. The default implementation of the DFBS algorithm is applied to strengthen cuts in the LBBD scheme for minimising total tardiness for cumulative scheduling problem (see Section 3.4.1). The weight counter is set up for each variable. In each Benders' iteration, the counter of a variable is increased by 1 when it is present in the irreducible cut. Note in the experiment that the weight counter is only used to collect information and does not influence the DFBS algorithm. The experiment solves 75 instances with 10, 12, 14, 16, and 18 jobs that must be scheduled on 2, 3, or 4 machines.

The results of the experiment are presented in Figure 4.3. There is a separate plot for each instance set corresponding to a particular number of jobs. A point on a plot corresponds to the probability of a variable (job) being in an irreducible cut. The probability is calculated as the ratio of the number of Benders' iterations where the variable was present in an irreducible cut (the weight of the variable) to the total number of Benders' iterations. Note that there can be multiple irreducible cuts, only one irreducible cut per iteration identified in one execution of the algorithm is used. In the plots, the variables are sorted in the descending order by the probability of appearing in the irreducible cuts. Note that the variables are sorted for each instance separately, and the sorting is different for each instance. For one instance, job number 2 can have the highest probability, while for another instance, it can be job number 10. The red line in each plot shows the average probability of jobs across instances.

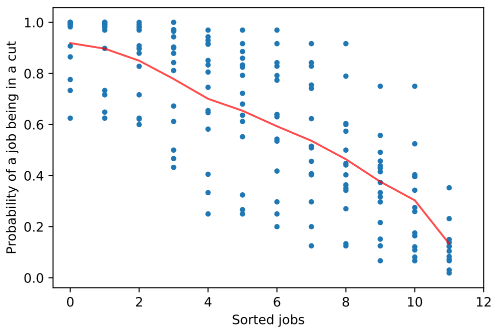
The first thing to notice in Figure 4.3 is that, some variables are more likely to be present in an irreducible cut than others. All of the plots show a clear downward trend — with some jobs having the probability as great as 1, and some jobs having probability of 0. For the instance set with 14 jobs, there is an instance where 7 jobs were present in all of the cuts. This indicates that as the solution scheme progresses, these jobs can clearly be prioritised over the other half of the jobs. In all of the plots, one can also observe in that even for the instances with all variables having low probabilities, some variables can clearly be prioritised over others. These instances are represented by the lowest series of points in each plot.

Another interesting observation is that, for the instance set with 18 jobs, the plot shows that all of the jobs were present in the irreducible cuts for one of the instances. This is due to solving the problem in one Benders' iteration and generating only one cut. Overall, in each plot, the red line indicates that for all of the instances, on average, some of the variables are more likely to be in an irreducible cut than others. Moreover, for each of the instances sets, there are jobs that were never present in the cuts.

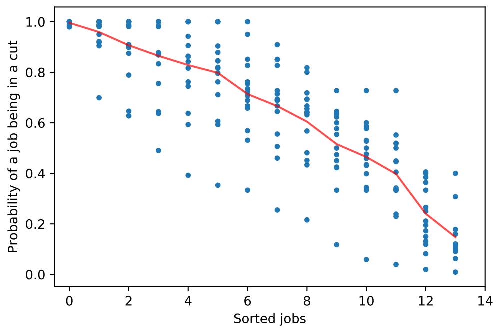
The experimental results show that, although different cuts are generated in each Benders' iteration, there are variables that appear in the cuts almost constantly. In other words, sorting by the probability of the variable being in a cut clearly favours some variables over others. By capturing the information about the previous cuts, one can direct the search towards the variables with higher weights, which are more likely to be in an irreducible cut.



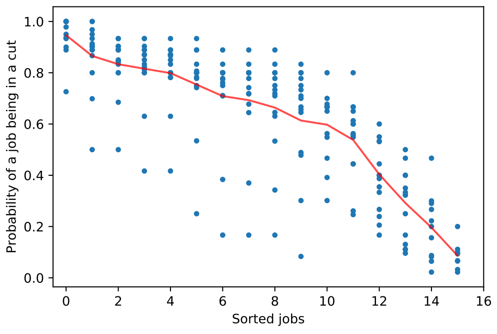
Instance set with 10 jobs



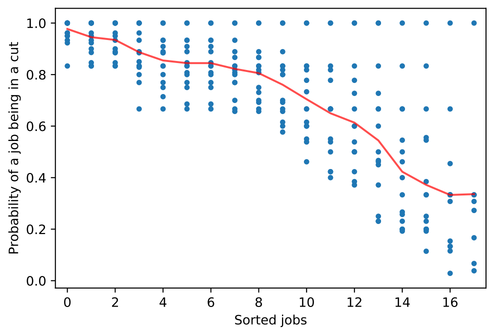
Instance set with 12 jobs



Instance set with 14 jobs



Instance set with 16 jobs



Instance set with 18 jobs

Figure 4.3: Results for the default DFBS implementation

4.3.3 Valid inequality

The second approach to variable sorting is to use the problem structure and data. It is important to determine what information about the problem structure can be used in variable sorting. One can observe from the Example 2 that favoured sorting should prioritise variables that are likely to appear in an irreducible cut.

The likelihood of variables appearing in a cut depends on the objective function and the type of cut generated. In the case of the minimising tardiness problem, the LBBD scheme mostly generates optimality cuts. Therefore, it is important to prioritise jobs that contribute to total tardiness.

In the solution process, once the jobs are assigned to facilities by the master problem, the subproblem schedules them according to the time windows of the jobs and the capacity of a facility. Note that the facility becomes known only after the master problem solution. One way to use this data and estimate tardiness of each job is to use subproblem relaxation in a form of valid inequality.

Recall that $\mathscr{J}(t_1,t_2)$ is the set of jobs that can be scheduled within time interval $[t_1,t_2]$. The number of jobs that can run simultaneously is limited due to the resource capacity of a facility. Therefore, based on the resource consumption of each job, the last job in the set $\mathscr{J}(0,d_k)$ scheduled on facility i has an end time of no earlier than

$$\bar{T} = \frac{1}{C_i} \sum_{j \in \mathscr{J}(0,d_k)} p_{jf} c_{jf}. \tag{4.1}$$

Since the last job has a due date no later than d_k, its tardiness is no less than $\max(\bar{T} - d_k, 0)$ (Hooker, 2007). This lower bound on tardiness of a job can be used as a score to prioritise the variables. The following score can be calculated for each variable k:

$$score = \frac{1}{C_i} \sum_{j \in \mathscr{J}(0,d_k)} p_{jf} c_{jf} - d_k \tag{4.2}$$

Note that unlike tardiness, that cannot have negative values, the score values can be negative. The variables with higher scores are more likely to contribute to total tardiness.

Since the score of a variable depends on the facility it is assigned to, the master problem solution is necessary to identify all of the jobs assigned to the same machine. The score is then recalculated in each Benders' iteration based on the master problem solution.

Similar to the weight counter, a computational experiment is set up to validate the use of the proposed strategy. The default implementation of the DFBS algorithm is applied to strengthen cuts in the LBBD scheme for the minimising total tardiness for cumulative scheduling problem. The experiment solves 75 instances from Hooker (2007) with 10, 12, 14, 16, and 18 jobs that must be scheduled on 2, 3, or 4 machines. The score values are calculated for all variables in each Benders' iteration. The calculation is only used for data collection and does not influence the DFBS algorithm.

The results of the computational experiment are presented in Figure 4.4. Since the scores are recalculated in each iteration, and the score of a variable varies significantly, the scores are not aggregated for variables. Instead, we look at all scores separately. The histograms on the left-hand side of the figure present score distributions for each instance set. Overall distribution of scores is given in blue colour, and the scores are divided into 30 bins for each instance set. Some of these scores belong to variables that were present in irreducible cuts. The distribution of such scores is given in orange colour, the bins for these scores are equal to the bins for all scores. The bar graphs on the right-hand side of the figure 4.4 represent the ratio between the number of scores in irreducible cuts and the overall number of scores. For each score value on the horizontal axis, the bar can be seen as the probability of a variable with such score being in an irreducible cut.

The most important thing to notice in Figure 4.4 is the difference between two distributions. For each instance set, the plot on the left-hand side shows that as the value of score increases, the difference between the orange bars and blue bars decreases. This can also be seen in the ratio bars on the right-hand side. This indicates that with the increase of the score value, it is more likely that the variable with such score will be present in an irreducible cut.

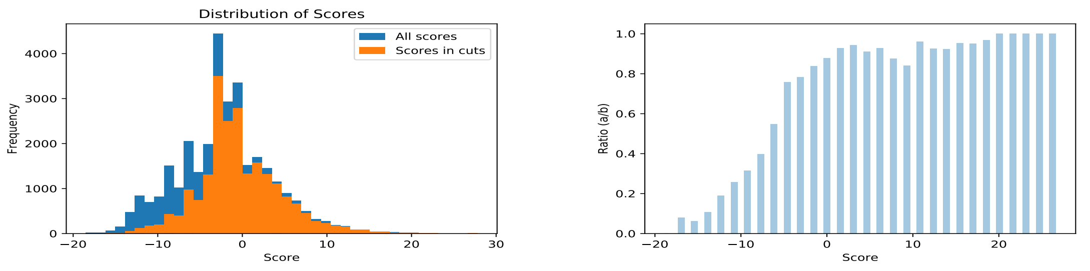
Another interesting observation is that most of the scores for all instance sets are in the range between -10 and 0. The distribution of scores in cuts usually has more significant increase in frequency than the overall distribution of scores. This again highlights that the probability of scores being in a cut increases with the value of the score. Much smaller number of scores have values greater than 0 or smaller than -10. The plots that correspond to instance sets with 10 and 14 jobs show that, when a greater number of scores have values outside of -10 and 0, the scores follow the overall trend of increasing probability. In other plots, sudden drops in probability can be seen, when the number of scores is significantly low.

Overall, the plots confirm that the variables with higher scores are more likely to be in an irreducible cut. The increase in the score value is generally matched by the increase in the probability of a variable being in the cut. This can been easily observed by upward trends in the bar graphs.

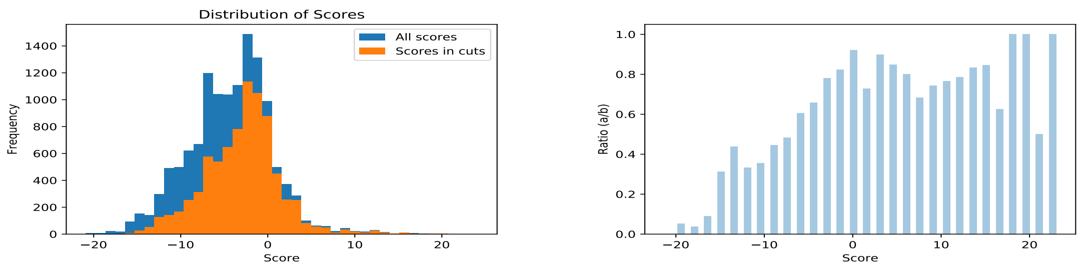
These results suggest that the proposed scores can be used in variable sorting. The variables with high scores should be prioritised over variables with lower scores.

4.4 Proposed heuristics

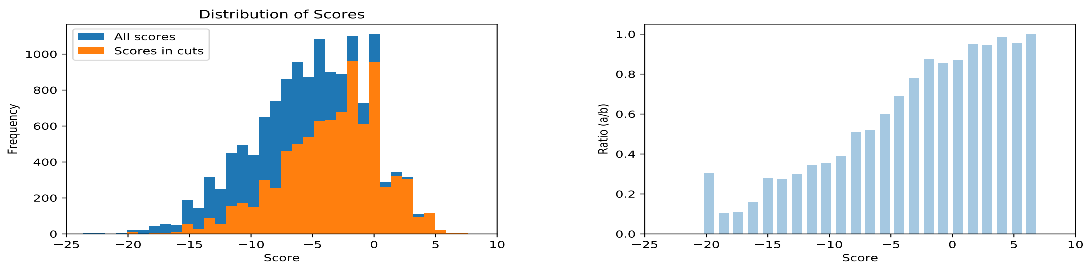
The preliminary investigation of the two strategies of variable sorting in Section 4.3 showed that both information about the cuts and the problem structure can be used to sort variables. This section proposes and formalises three new heuristics that can be used to sort variables within the DFBS algorithm.



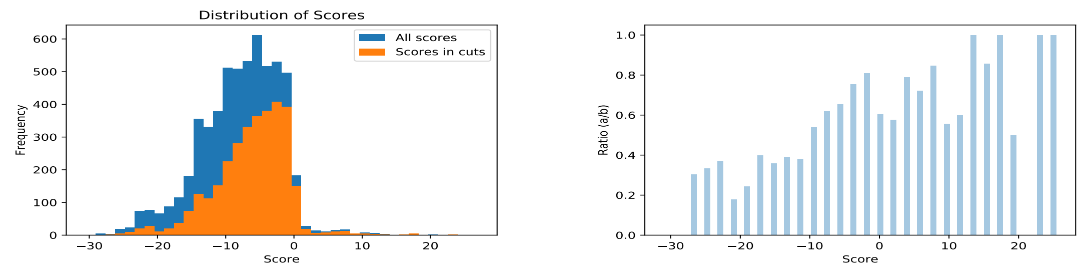
Instance set with 10 jobs



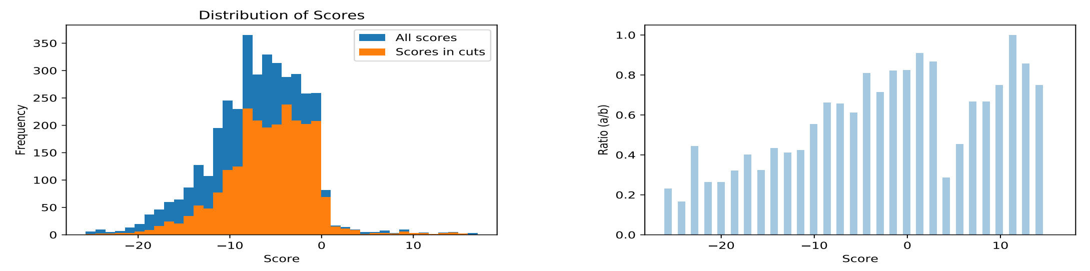
Instance set with 12 jobs



Instance set with 14 jobs



Instance set with 16 jobs



Instance set with 18 jobs

Figure 4.4: Distribution of scores

4.4.1 Proposed heuristic *weights*

The main idea of the *weights* heuristics is to use the variable weight counter to direct the search in DFBS. Initially, all of the variable weight counters are set to 0. In each iteration of the LBBD, the weight counter of a variable is then increased by 1 when the variable belongs to the irreducible cut. As the LBBD solution scheme progresses, the variables contributing to the objective value (infeasibility) the most, have higher weights. See Algorithm 10.

Algorithm 10 Weight counter

```
Input: An irreducible subset set $\mathscr{J}(\hat{x}), weight(x)$
Output: An array weight of variables' weights
1: for $j \in \mathscr{J}(\hat{x})$ do
2:     $weight[x_j] \leftarrow weight[x_j] + 1$
3: end for
4: return $weight(x)$
```

Algorithm 10 updates the *weights* array in each Benders' iteration. The array is then used in the DFBS algorithm (see Algorithm 8) to prioritise variables with the greatest weights when sorting variables in set T.

The main advantage of the proposed method is the simplicity. Updating the weights of variables does not require extra computation. Another important advantage is that the weights do not depend on the problem structure.

4.4.2 Valid-inequality based priority

Unlike the *weights* heuristics, the valid-inequality based priority utilises problem structure, and therefore has to be tailored for each problem. However, using valid inequalities can be considered as the general guide.

The valid inequality based priority for the minimising total tardiness for cumulative scheduling problem is implemented as follows. The heuristic recalculates *scores* in (4.2) for variables based on the lower bound of tardiness of a job in (4.1). The *scores* are recalculated in each Benders' iteration based on the master problem solution. The variable in set T are then sorted in descending order of their scores.

Since the scores are recalculated in each Benders' iteration, the heuristics only takes into account the current state of the LBBD search. In addition to using the problem structure, this is another main difference with the *weights* heuristics. The advantage of it is that, the heuristic is efficient from the first iteration.

4.4.3 Weight+Dynamic size

This heuristic combines the ideas of variable sorting and dynamic grouping of constraints. One can note that, if T_1 either contains too few or too many variables and, subsequently, the objective value of the relaxation is not equal to the original objective value, set T_2 will not be discarded. DFBS will then pursue T_2, while T_1 is appended to S and "set aside". Set S becoming large means the number of variables to evaluate is still high, and it has adverse effects on computational efficiency.

Ideally, set T_1 should contain the number of variables equal to or close to the size of the cut that will eventually be generated. One way to get an estimate of the size is to look at the average size of the cuts generated so far. The ratio of the average reduced cut size to the original cut size can then be used to split set T. The ratio is calculated as follows,

$$ratio = \frac{N_{av}}{N}, \tag{4.3}$$

where N_{av} is the average number of jobs in the irreducible cuts generated so far, and N is the number of jobs in the original cut.

This heuristic should be used in combination with a heuristic that sorts variables. We suggest using the *weights* heuristic, since it is also does not depend on the problem structure and requires no extra computation. Set T is first sorted using *weights*, and then split in the following way

$$T_1 := \{x_i \in T | i = 1, ..., ratio * n\}, \quad T_2 = \{x_i \in T | i = ratio * n, ..., n\},$$

where n is the total number of variables. The main idea of this heuristic is to facilitate all of the IIS variables falling to T_1, ideally the first time T is split.

4.5 Computational evaluation

I will refer to the modified versions of the DFBS algorithm as different cut-strengthening techniques. The effectiveness and efficiency of the cut-strengthening techniques are evaluated in a similar way as in Chapter 3. The computational experiment solves the cumulative scheduling problem to minimise tardiness. The LBBD scheme is set up the same way as for the preliminary investigation. The problem is solved using the LBBD solution scheme applying each cut-strengthening technique separately. Minimising tardiness problem generates only value cuts. As in the previous chapter, the comparison is made between solving the split and no-split subproblem.

The comparison of the computational effectiveness will be made by evaluating the run time and the size of the strengthened cuts. Since I am interested in the efficiency of each strategy, particular attention will be paid to number of subproblems solved and to the number of Benders' iterations.

4.5.1 Instances

All of the 336 instances are taken from Hooker (2007) with the same settings as in the previous chapter. The number of tasks to be scheduled varies between 10, 12, 14, up to 38, and the number of machines goes from 2 to 10 machines.

4.5.2 Computational effectiveness and efficiency

The effectiveness of the strategies is evaluated by looking at the impact on the run time of the LBBD scheme. The main metric for the efficiency of each search strategy is the number of subproblems solved.

For the run time plots, the horizontal axis gives the time and the vertical axis gives the percentage of solved instances. A point (x, y) on the curve means that $y\%$ of instances can be solved in less than x seconds. The cut size figures are histograms, a point (x, y) means average cut size x has frequency y. Where the frequency y is the number of generated cuts.

All of the statistics in the tables are evaluated for instances that were solved by all of the techniques. The tables below present the following data.

$\mathrm{N_{iter}}$ — average number of Benders' iterations needed to solve an instance.

$\mathrm{N_{sub}}$ — average number of subproblems solved per instance.

$\mathrm{T_{sub}}$ — average subproblem solution time per instance.

$\mathrm{T_{mas}}$ — average master problem solution time per instance

$\mathrm{N_{inst}}$ — number of instances solved by all of the techniques

$\mathrm{P_{sol}}$ — percentage of instances solved by one technique

$\mathrm{N_{sol}}$ — number of instances solved by one technique

$\mathrm{T_{inst}}$ — average time spent on problems solved by all of the techniques.

In the figures and tables below, the modifications of DFBS using weighted variable prioritising, valid inequality prioritising, and dynamic size grouping are referred to as *weight*, *valin*, and *dyno*, respectively. The *dfbs* method refers to the DFBS using the random order of variables. Separate results are given for the split and no-split subproblems.

4.5.2.1 No-split subproblem

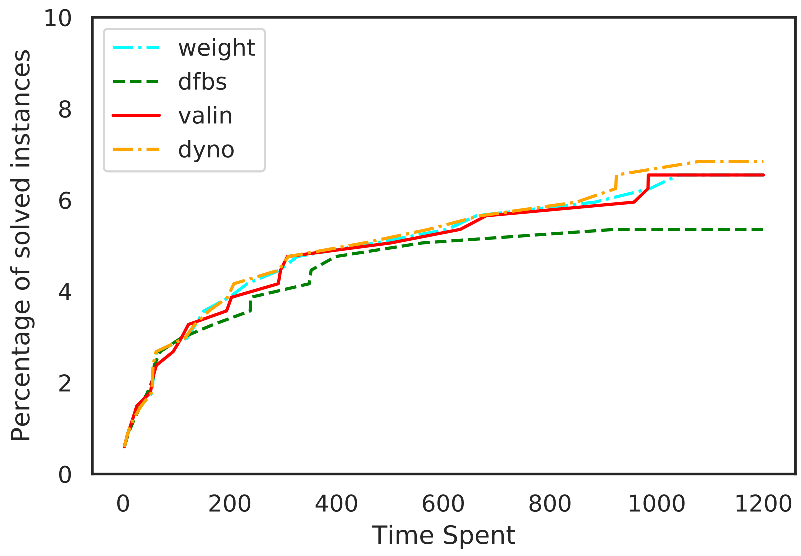


Figure 4.5: Percentage of solved instances for minimising tardiness problem with strengthened optimality cuts and no-split subproblem

Cut-strengthening technique	P_{sol}	T_{spent}	N_{sol}	N_{inst}
DFBS	5.35	213.77	18	17
DYNO	6.85	166.33	23	17
WEIGHT	6.55	170.04	22	17
VALIN	6.55	177.73	22	17

Table 4.1: Results for the minimising tardiness problem with no-split subproblem

It can be seen in Figure 4.5 and Table 4.1 that all of the modifications of the DFBS algorithm outperform the default implementation, both in terms of the profile and in terms of the number of instances solved. Figure 4.6 shows that all of the techniques generate cuts of similar size, the number of Benders' iterations is also similar for all of the techniques (see Table 4.2). The difference in performance comes from the lower number of subproblems solved by the modified techniques. Valid inequality technique solves 1192 subproblems on average, the weight technique solves 1156, and the dynamic size techniques solves 1104 subproblems, which is 6.5%, 9.3%, and 13.4% less than 1275 subproblems by the default implementation,respectively. The lower number of subproblems then results in lower subproblem solution time, as can be seen in Table 4.2. The overall most efficient technique is the dynamic size technique. It solves more instances than other techniques — 23, compared to 22 by the weight and valid inequality, and 18 by the default implementation; while solving the smallest number of subproblems. This leads to the dynamic size technique spending 27.17% less time on solving subproblems than the default implementation.

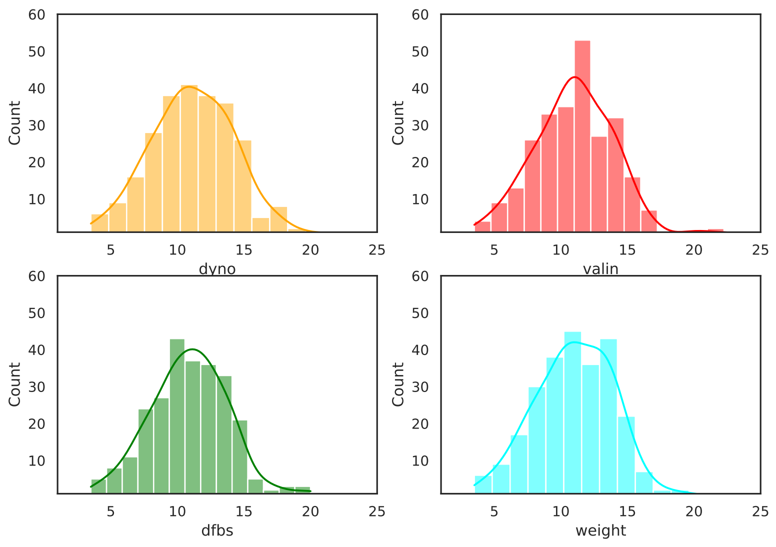


Figure 4.6: Cumulative scheduling: : Average size of optimality cuts for the minimising tardiness problem with no-split subproblem

Cut-strengthening technique	N_{iter}	N_{sub}	T_{sub}	T_{mas}	N_{inst}
DFBS	63	1275	211.13	1.27	17
DYNO	64	1104	153.75	1.28	17
WEIGHT	64	1156	167.45	1.26	17
VALIN	65	1192	174.96	1.27	17

Table 4.2: Results for the minimising tardiness problem with no-split subproblem

4.5.2.2 Split subproblem

Similar to the results for the no-split problem, it can be seen in Figure 4.7 and Table 4.3 that all of the modified techniques outperform the default implementation of the DFBS algorithm. The results in Figure 4.8 illustrate that all of the techniques generate cuts of similar size. While the number of Benders' iterations is similar too (see Table 4.4), interestingly, all of the modified techniques also spend less time solving the master problem. Note that there can be multiple irreducible cuts for each master problem solution. The cuts generated by different techniques are not necessarily the same. The master problem solution time might indicate that the cuts generated by the modified techniques are less complicated for the

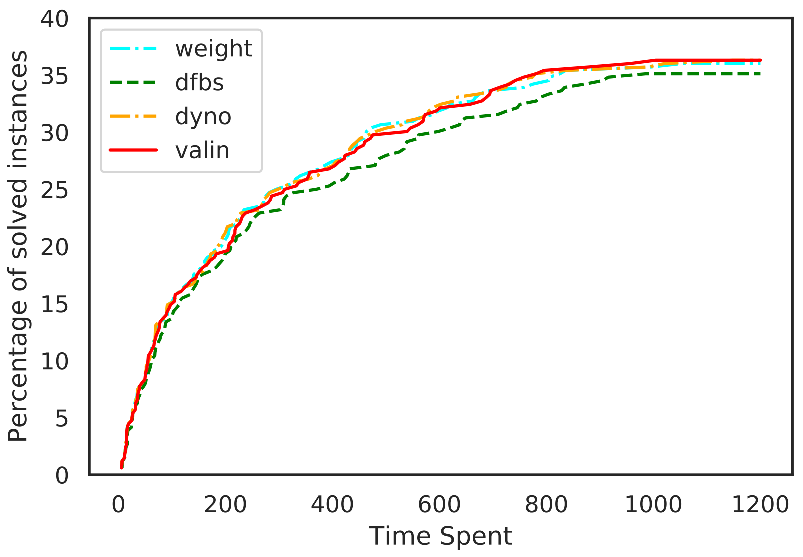


Figure 4.7: Percentage of solved instances for minimising tardiness problem with strengthened optimality cuts and split subproblem

Cut-strengthening technique	P_{sol}	T_{spent}	N_{sol}	N_{inst}
DFBS	35.12	256.20	118	113
DYNO	36.31	208.44	122	113
WEIGHT	36.01	213.68	121	113
VALIN	36.31	226.77	122	113

Table 4.3: Results for the minimising tardiness problem with no-split subproblem

master problem. Similar to the no-split subproblem, the dynamic size technique solves the smallest number of subproblems — 1823, which is 9.2% less than 2009 by the default implementation. The lower number of subproblems solved is also reflected in the lower subproblem solution time — 201.33 on average per instance, which is 9.2% lower than 222.05 by the default implementation. This result is closely followed by the weight heuristic with 1928 subproblems solved and 206.67 seconds of subproblem solution time.

Cut-strengthening technique	N_iter	N_sub	T_sub	T_mas	N_inst
DFBS	72	2009	222.05	5.56	113
DYNO	71	1823	201.33	4.98	113
WEIGHT	71	1928	206.67	4.89	113
VALIN	72	1993	219.76	4.82	113

Table 4.4: Results for the minimising tardiness problem with split subproblem

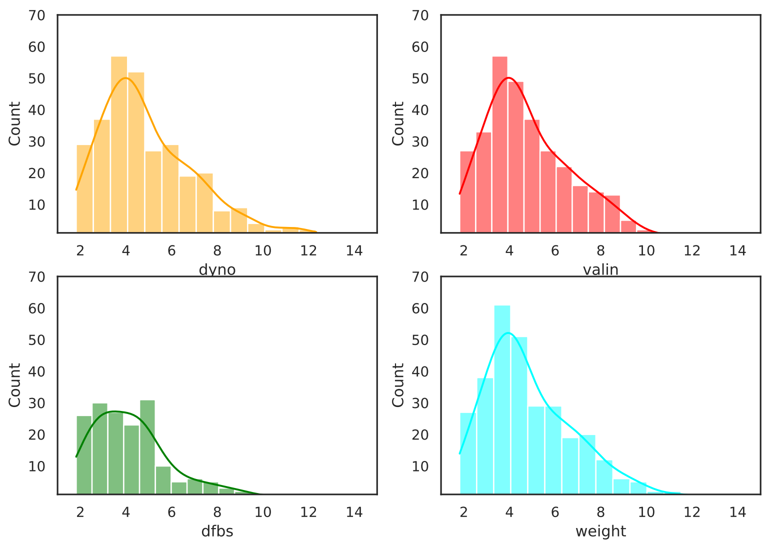


Figure 4.8: Cumulative scheduling: : Average size of feasibility cuts for the finding a feasible schedule problem with split subproblem

4.6 Conclusions

This chapter investigates new methods for improving the efficiency of the DFBS algorithm. Namely, the impact of variable sorting and dynamic grouping of variables is investigated. Three methods were proposed and evaluated — weighted variable sorting, valid inequality sorting, and combination of weighted variable sorting with dynamic size grouping of variables. The evaluation is based on the computational experiment that solves the minimising tardiness for cumulative scheduling problem. The results are reported for the split and no-split subproblems.

The computational results show that all of the modified techniques have better efficiency than the default DFBS cut-strengthening technique. The dynamic group size technique combined with the weighted variables showed the best results. The proposed technique solved the lowest number of subproblems, and subsequently spent the lowest time solving the subproblems. An interesting observation is how the lower number of subproblems solved translates into difference in the subproblem solution time — for the no-split subproblem the percentage difference in the number of solved subproblems gives roughly twice as much difference in subproblem solution time. This indicates that the modified techniques not only solve fewer subproblems, but also lead to less difficult subproblems, i.e. subproblems with fewer variables. For the split subproblem the percentage difference in the number of subproblem solved and the difference in subproblem solution time are almost equal. This indicates that the difference in the size of subproblems solved is not as significant as for the no-split subproblem. Another interesting observation is that the modification also impacts the master problem solution time for the split subproblem.

The proposed techniques save the cost of solving more subproblems, which can have big impact for large industrial instances (such as the instances presented in (Karlsson & Rönnberg, 2022)). Importantly, the proposed techniques are easy to implement and require no additional computation. Therefore, it is worth investing time in implementing the modifications.

Chapter 5

Subproblem separation in logic-based Benders' decomposition for the vehicle routing problem with local congestion

5.1 Introduction

Logic-based Benders' decomposition (LBBD) is an extension of classical Benders' decomposition. LBBD extends the classical Benders' approach by allowing the subproblem to be an optimisation problem of any form. LBBD was first introduced in (Hooker, 2000) and then formalised in (Hooker & Ottosson, 2003). It generates Benders' cuts by using logical deductions from subproblem solutions, therefore it can handle any type of subproblem. This makes LBBD an effective method of combining different approaches such as mixed-integer programming (MIP) and constraint programming (CP).

Successful applications of LBBD include resource allocation and scheduling problems (Coban & Hooker, 2013; Emde, Polten & Gendreau, 2020; Hooker, 2007; Karlsson & Rönnberg, 2022; Lombardi & Milano, 2012; Sun, Tang & Baldacci, 2019), vehicle routing problems (Lam et al., 2020; Raidl, Baumhauer & Hu, 2014; 2015; Riazi et al., 2013), and other types of large-scale optmisation problems (Hooker, 2019). Various acceleration techniques have been used in these applications in order to improve the effectiveness of the LBBD scheme. The common acceleration techniques are subproblem relaxation, cut-strengthening techniques, and subproblem separation. The computational evaluation in Chapter 3 shows that for problems, which have inherently separable subproblems, benefits of applying subproblem separation dominate the benefits of applying cut-strengthening techniques.

Subproblem separation can be applied to problems with a bordered block-diagonal structure. This structure allows splitting the subproblem when master problem variables are fixed. We propose a new method for subproblem separation for problems that do not have such structure. On the example of the VRPLC, we demonstrate why problems with no block-diagonal structure can not be separated at the problem formulation stage. We then introduce a new method that applies the connected components algorithm to identify separable blocks of the subproblem based on the master problem solution in each iteration of the Benders' algorithm. The computational evaluation of the new method confirms that subproblem separation we propose has a significant impact on the LBBD scheme.

The main contributions of this chapter are:

- A new method of subproblem separation in the LBBD scheme using the connected components in a graph.
- A derivation and analytical validation of various Benders'cuts.
- Detailed computational experiments evaluating four different methods applying subproblem separation and the default method with no subproblem separation.
- The code related to the LBBD scheme and the subproblem separation is freely available at https://git.exeter.ac.uk/as1392/subproblem-separation-in-lbbd.

The rest of this chapter is structured as follows. Section 5.2 describes the VRPLC. Section 5.3 presents the problem formulation. A brief description of the LBBD scheme for the problem is given in Section 5.4. The main contributions of this paper are Sections 5.5–5.7. Section 5.5 introduces the new method for subproblem separation. The derivation of new Benders' cuts is given in Section 5.6.

The computational results in Section 5.7 demonstrate the effectiveness of the subproblem separation by evaluating the solution run times for the various new methods. Finally, concluding comments are given in Section 5.8.

5.2 Vehicle routing problem with local congestion

The vehicle routing problem (VRP) (Clarke & Wright, 1964) is a problem that finds a set of suitable routes for a fleet of vehicles that deliver/collect goods from the central depot to a set of customers. The obtained routes must satisfy all of the customer requests while minimising the total travel distance and/or other costs. Some examples of practical applications are grocery delivery, parcel delivery (Berbeglia et al., 2007), farm produce collection, waste collection (Benjamin & Beasley, 2010) etc. The VRP is one of the most extensively studied problems in optimisation due to both practical and theoretical interest.

There is no standard VRP formulation. However, one of the best-studied formulations of the vehicle routing problem is the capacitated VRP (CVRP)(Arnold, Gendreau and Sörensen, 2019; Pecin et al., 2017; Raidl, Baumhauer and Hu, 2014; 2015; Ralphs et al., 2003; Riazi et al., 2013; Uchoa et al., 2017, and references therein). A lot of VRP problem formulations are based on CVRP. A feasible solution of CVRP is a set of routes starting and ending at the central depot. Every customer is visited only once on a specific route, and the cumulative demand (weight) of all requests the vehicle delivers must not exceed its capacity.

The VRPLC is the CVRP enriched with time window and local congestion constraints. This variant of the CVRP was introduced in (Lam, Pardalos & Hentenryck, 2016). The customer requests are grouped by locations. The congestion constraint at each location is a cumulative resource constraint that limits the number of vehicles present and/or in service at any given time. If all resources at a location are engaged, incoming vehicles must wait until the resources become available. This leads to time dependencies, and, subsequently, a scheduling substructure that is not present in conventional CVRPs (Lam, Pardalos & Hentenryck, 2016).

An example of such time dependency is that a delay on one route entails delays on other routes visiting the same location. These delays can cause infeasibility of a solution because of the time window constraints.

Developing a hybrid MIP and CP solver ”Nutmeg”, Lam et al. (2020) introduce a logic-based Benders' decomposition (LBBD) scheme for the VRPLC. As VRPLC can be decomposed into a routing and a scheduling problem, it is naturally suited to LBBD. The authors use the branch-and-cut style of LBBD known as branch-and-check. Two objective types are considered— minimising total travel distance and minimising makespan.

5.3 Problem formulation

The problem formulation studied in this chapter is based on the VRPLC formulation in (Lam et al., 2020). However, we only consider the minimising total tardiness objective, that was shown to be the most difficult in Chapter 3. The requests are allowed to be delivered past their time window, the tardiness of each request is the amount of time that has passed since the end of the window until the delivery time. The total tardiness is the sum of the tardiness of all requests.

The problem is to create a set of routes for vehicles to deliver goods from a central depot to various locations. The vehicles and the locations are subject to vehicle capacity and congestion constraints, respectively.

Table 5.1 lists the data and decision variables for the problem. The requests for goods are grouped by locations. Each request $i \in R$ must be delivered to location $l_i \in \mathscr{L}$ within a time window $[r_i, d_i]$, and all vehicles must return to the central depot before time T. Each vehicle requires the use of one piece of equipment for processing time p_i to unload the goods for request $i \in R$. Each location only has the total fixed set of equipment C_l, the limited capacity of equipment then leads to location congestion.

The problem can be modelled using a graph $(\mathscr{N}, \mathscr{A})$. The central depot and

Name	Description
Sets	
$l \in \{1,\dots,\mathscr{L}\}$	Set of locations
$\mathscr{N} = R \cup \{O^-, O^+\}$	Set of nodes
$R_l = \{i \in R \vert l_i = l\}$	Set of requests at location $l \in \mathscr{L}$
$\mathscr{A} = \{(i,j) \in \mathscr{N} \times \mathscr{N} \vert i \neq j\}$	Set of arcs connecting the nodes
$R = \{1,\dots,n\}$	Set of requests
Parameters	
$T \in \{1,\dots,\infty\}$	Time horizon
$C_l \in \{1,\dots,\infty\}$	Resource capacity of location $l \in \mathscr{L}$
n	Number of requests
$l_i \in \mathscr{L}$	Location of request $i \in R$
$r_i \in [0,T]$	Release time of request $i \in R$
$d_i \in [0,T]$	Deadline of request $i \in R$
$p_i \in [0,T]$	Processing time of request $i \in R$
$q_i \in [1,Q]$	Weight of request $i \in R$
$Q \in \{0,\dots,\infty\}$	Maximum weight a vehicle can carry
O^+	Artificial start that corresponds to the central depot
O^-	Artificial end that corresponds to the central depot
c_{ij}	Travel time along arc (i,j)
Decision variables	
$x_{ij} \in \{0,1\}$	Indicates if a vehicle travels along arc (i,j)
$y_i^{\text{start}} \in [r_i, d_i]$	Time a vehicle starts unloading goods
$y_i^{\text{weight}} \in [q_i, Q]$	Total accumulated weight of delivered goods

Table 5.1: Data and decision variables for the model

each request $i \in R$ with the corresponding location information are represented through the set of nodes $\mathscr{N}$. The set $\mathscr{A}$ denotes the arcs connecting the nodes. The variables x_{ij} equal 1 if a vehicle travels along arc $(i,j) \in \mathscr{A}$, and 0 otherwise. Moving along arc (i,j) takes c_{ij} time units. There are two continuous subproblem variables at each node $i \in \mathscr{N}$. The continuous variables y_i^{start} and y_i^{weight} are equal to the time a vehicle starts unloading goods and the total accumulated weight of delivered goods, respectively.

The model for the vehicle routing problem with location congestion is given by

$$\min \sum_{i \in R} \max\{y_i^{\text{start}} + p_i - d_i, 0\} \tag{5.1}$$

$$\text{s.t.} \sum_{i:(i,j)\in\mathscr{A}} x_{ij} = 1, \quad j \in R, \tag{5.2}$$

$$\sum_{j:(i,j)\in\mathscr{A}} x_{ij} = 1, \quad i \in R, \tag{5.3}$$

$$\textsc{Cumulative}((y_i^{\text{start}} | i \in R_l), (p_i | i \in R_l), (1 | i \in R_l), C_l), \quad l \in \mathscr{L}, \tag{5.4}$$

$$x_{ij} = 1 \rightarrow y_i^{\text{weight}} + q_j \le y_j^{\text{weight}}, \quad (i,j) \in \mathscr{A}, \tag{5.5}$$

$$x_{ij} = 1 \rightarrow y_i^{\text{start}} + p_i + c_{ij} \le y_j^{\text{start}}, \quad (i,j) \in \mathscr{A}, \tag{5.6}$$

$$x_{ij} \in \{0,1\}, \quad (i,j) \in \mathscr{A}, \tag{5.7}$$

$$y_i^{\text{start}} \in [r_i, d_i], \quad i \in \mathscr{N}, \tag{5.8}$$

$$y_i^{\text{weight}} \in [q_i, Q], \quad i \in \mathscr{N}. \tag{5.9}$$

The objective function (5.1) minimises total tardiness of all requests. Constraints (5.2)–(5.3) ensure each node has exactly one incoming and outgoing arc. This ensures each request is assigned to exactly one vehicle. The CUMULATIVE constraints (5.4) enforce processing capacity limit at each location. Vector $((1|i \in R_l))$ represents resource requirement for each request $i \in R_l$. The CUMULATIVE constraints require the following: $\sum_{i \in R_{lt}} 1 \le C_l$ for all times t, where $R_{lt} = \{i | y_i^{\text{start}} \le t < y_i^{\text{start}} + p_i\}$ is the set of requests being processed at time t. Constraints (5.5)–(5.6) are only enforced when the corresponding values of x_{ij} are equal to 1. Constraints (5.5) ensure that the total accumulated weight of delivered goods by a vehicle does not decrease after each delivered request. Constraints (5.6) ensure request processing time and minimum travel times are respected. Constraints (5.6) are sufficient to avoid cycles. All of the vehicles are assumed to be identical, and each node has one incoming and outgoing arc, therefore the vehicles are not presented explicitly. The number of arcs outgoing from (or incoming to) the central depot gives the number of vehicles used in a solution.

5.4 Logic-based Benders' decomposition for VRPLC

The VRPLC decomposes into routing and scheduling components. The variables x_{ij} are viewed as the complicating variables. Fixing variables x_{ij} to trial values leads to a scheduling subproblem. The scheduling subproblem can be solved as a CP. The trial values of x_{ij} are found by solving the routing master problem as a MIP. The master problem identifies a set of vehicle routes that satisfy all delivery requests.

Let T denote the total tardiness. The master problem in iteration k is given by

$$\min T \tag{5.10}$$

$$\text{s.t.} \sum_{i:(i,j)\in\mathscr{A}} x_{ij} = 1, \quad j \in R, \tag{5.11}$$

$$\sum_{j:(i,j)\in\mathscr{A}} x_{ij} = 1, \quad i \in R, \tag{5.12}$$

$$T \geq B_{x^i}(x), \quad i = 1,...,k-1, \tag{5.13}$$

$$[\text{Valid inequalities}]. \tag{5.14}$$

The [Valid inequalities] contain constraints (5.5)–(5.6), they are added to the master problem to retain some information about the subproblem. They state that for a vehicle that travels along arc (i,j) the accumulated weight of goods delivered after request j cannot be smaller than the accumulated weight after request i. Similarly, unloading of goods at node j cannot start before unloading at node i.

Inequalities (5.13) are the Benders' cuts obtained by solving the subproblem for master problem solutions x^i in iterations $i = 1, \dots, k-1$. The Benders' cuts ensure feasibility and optimality of the problem solution.

Let x^k be the master problem solution in iteration k, then the subproblem is given by

$$\min \sum_{i \in R} \max\{y_i^{\text{start}} + p_i - d_i, 0\} \tag{5.15}$$

$$\text{s.t. } \textsc{Cumulative}((y_i^{\text{start}} | i \in R_l), (p_i | i \in R_l), (1 | i \in R_l), C_l), \quad l \in \mathscr{L}, \tag{5.16}$$

$$x_{ij}^k = 1 \rightarrow y_i^{\text{weight}} + q_j \leq y_j^{\text{weight}}, \quad (i,j) \in \mathscr{A}, \tag{5.17}$$

$$x_{ij}^k = 1 \rightarrow y_i^{\text{start}} + p_i + c_{ij} \leq y_j^{\text{start}}, \quad (i,j) \in \mathscr{A}. \tag{5.18}$$

The subproblem is solved to schedule deliveries on the routes identified by the master problem.

The solution procedure is an iterative process that iterates between solving the master problem and the subproblem. Let T^* and T_k^* denote the optimal objective value of the master problem and the subproblem, respectively. In each iteration, the optimal value T^* provides a lower bound on the optimal value of (5.1)–(5.9), and T_k^* provides an upper bound. The optimal value T^* increases monotonically, the subproblem value T_k^* can increase or decrease. The procedure terminates when $T^* = \min\{T_1^*, \dots, T_k^*\}$.

The main idea of LBBD is to use T_k^* and the reasoning behind this solution to obtain a bounding function $B_{x^k}(x)$ that gives a valid lower bound on the optimal value of (5.1). The bounding function $B_{x^k}(x)$ should have following two properties.

Property 3 $B_{x^k}(x)$ *provides a valid lower bound on* (5.1) *for any given* $x \in D_x$, *where* D_x *is the domain of* x. *That is,* $T \geq B_{x^k}(x)$ *for any feasible* (x,y) *in problem .*

Property 4 $B_{x^k}(x^k) = T_k^*$.

It is convenient to regard T_k^* as having an infinite value if the subproblem is infeasible. By this assumption, a strong duality holds for the dual of the subproblem: the optimal value of the subproblem is always equal to the optimal value of its dual (Hooker & Ottosson, 2003).

Theorem 3 *(Hooker, 2007) If the bounding function $B_{x^k}(x)$ satisfies properties 3 and 4 in each iteration of the Benders algorithm, and the domain D_y of y is finite, the Benders algorithm converges to the optimal value of (problem) after finitely many steps.*

Let $\mathscr{J}_k = \{(i,j) \in \mathscr{A} | x_{ij}^k = 1\}$ be the set of arcs that were selected in the master problem solution in iteration k. If the subproblem is infeasible, a feasibility cut given by

$$\sum_{(i,j)\in\mathscr{J}_k} (1 - x_{ij}) \geq 1 \tag{5.19}$$

is generated. If the subproblem has an optimal solution with value T_k^*, an optimality cut $T \geq B_{x^k}(x)$ is generated. The cut is given by

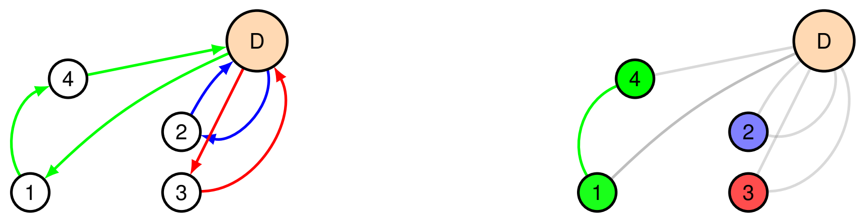
$$T \geq T_k^* \big(1 - \sum_{(i,j)\in\mathscr{J}_k} (1 - x_{ij})\big). \tag{5.20}$$

The cut indicates that the total tardiness T will have a value of at least T_k^*, unless one of the arcs $(i,j) \in \mathscr{J}_k$ is removed from the route.

Both feasibility cuts (5.19) and optimality cuts (5.20) can be routinely strengthened by replacing $\mathscr{J}_k$ with a smaller subset $\mathscr{J}_k^{'} \subseteq \mathscr{J}_k$, if the subproblem corresponding to $\mathscr{J}_k^{'}$ gives a solution with the same objective value as the solution for $\mathscr{J}_k$.

5.5 Subproblem separation

Subproblem separation is a common strategy used to accelerate the LBBD scheme. It is especially useful when the subproblem is a scheduling problem. Given that scheduling problems are difficult to scale up, splitting one big subproblem into many small independent subproblems usually benefits the solution procedure.



Routes in the MP solution   Connected components

Figure 5.1: Example of a graph representing the central depot (D) and four locations.

The subproblem can be separated when the problem has a bordered block diagonal structure, where the master variables define the border. Therefore, fixing the master variables to trial values makes the blocks separable. Meaning that the subproblem can be decoupled into a separate subproblem for each such block, and the solution of any decoupled subproblem does not depend on solutions of other subproblems.

The VRPLC formulation does not exhibit a block diagonal structure. Fixing master variables for the vehicle routing problem, does not naturally decouple the subproblem. One might consider decoupling the subproblem by each location. However, since a vehicle can travel through more than one location, constraints (5.5)–(5.6) create a border that connects the locations. For example, if a vehicle first travels through location l_1 and then location l_2, the subproblem for location l_2 would require the subproblem solution for l_1. This issue could be resolved by solving the subproblem for l_1 before solving subproblem for l_2. However, since a location may host several requests, it is possible that another vehicle travels through l_1 and l_2 in the opposite order, thus making this method of subproblem separation inapplicable. Therefore, the subproblem cannot be decoupled by locations at the problem formulation stage.

We propose to separate the subproblem during the solution process. One can note that some master subproblem solutions give routes that only connect some of the locations. For example, see Figure 5.1a, locations 1 and 4 are connected to each other and the central depot, while locations 2 and 3 are only connected to the central depot. One subproblem can be solved for locations 1 and 4 together, and one subproblem for each of locations 3 and 4.

We propose to identify separable blocks of locations in each iteration of the Benders' algorithm. The master problem solution x^k_{ij} can be represented as a graph $\mathscr{J}_k$. The edges in graph $\mathscr{J}_k$ are the connections between requests. In order to identify separable locations, a new graph is formed where the nodes are given by locations. The edges between requests are mapped onto the edges between locations. Connected locations can then be found by using an algorithm to identify connected components in the location graph. Our implementation uses a depth-first search (DFS) to identify connected components. Note that the edges that connect the central depot to the locations are ignored, otherwise, all of the locations will belong to a single connected component through the central depot. The number of independent calls of the DFS function is equal to the number of connected components. In the example above, (see Figure 5.1b) the connected components are $\{[1,4],[2],[3]\}$. A separate subproblem is then solved for each connected component. Observe that solving the subproblem when a connected component is a path is trivial.

It is important to note that the proposed algorithm can identify connected components for more general cases than the example above. Since a vehicle can travel to multiple locations, the algorithm ensures that all of the locations visited by one vehicle belong to a single component. Moreover, if several vehicles deliver requests to the same location, all of the locations traversed by the vehicles will be encompassed in a single component. This is possible due to the step of mapping the edges between requests to the edges between locations — several request nodes become one location node.

5.6 New Benders' cuts

There is an inherent computational benefit to splitting the subproblem into smaller independent subproblems. Nevertheless, in order to fully exploit the new subproblem structure it is important to generate strong Bender's cuts. In this section, we analytically derive different sets of valid Benders' cuts.

Let x^k be the master problem solution in iteration k, and let set $\mathscr{J}_k$ be given by $\mathscr{J}_k = \{(i,j) | x^k_{ij} = 1\}$. As mentioned in Section 5.5, since $\mathscr{J}_k \subseteq \mathscr{A}$ is a graph, it can be separated into connected components. Let set C_k denote the set of connected components in iteration k. The connected components partition the set $\mathscr{J}_k$, i.e.,

$$\bigcup_{c \in C_k} \mathscr{J}_k^c = \mathscr{J}_k \quad \text{and} \quad \mathscr{J}_k^c \bigcap \mathscr{J}_k^{c'} = \varnothing \quad c, c' \in C_k, \; c \neq c',$$

where $\mathscr{J}_k^c \subseteq \mathscr{J}_k$ is the set of all edges from $\mathscr{J}_k$ in the connected component c.

Sets $\mathscr{J}_k^c$ partition $\mathscr{J}_k$, cut (5.20) therefore can be rewritten as

$$T \geq T_k^* \big(1 - \sum_{c \in C_k} \sum_{(i,j) \in \mathscr{J}_k^c} (1 - x_{ij})\big). \tag{5.21}$$

Each connected component $c \in C_k$ describes a separate subproblem. The optimal objective for subproblem c in iteration k is given by T_{ck}^*. Further, cut (5.21) can then be rewritten as the first cut we are proposing to generate

$$T \geq \sum_{c \in C_k} T_{ck}^* \big(1 - \sum_{(i,j) \in \mathscr{J}_k^c} (1 - x_{ij})\big). \tag{5.22}$$

The cut (5.22) can be seen as a summation of cuts of type (5.20) for each component $c \in C_k$. Note that cut strengthening can be applied to all cuts presented in this section unless otherwise stated.

The second valid set of cuts we propose to generate in iteration k is given by

$$T_{ck} \geq T_{ck}^* \big(1 - \sum_{(i,j) \in \mathscr{J}_k^c} (1 - x_{ij})\big), \quad c \in C_k, \tag{5.23}$$

$$T \geq \sum_{c \in C_k} T_{ck}. \tag{5.24}$$

The auxiliary variables T_{ck}, denoting total tardiness for each connected component, are added to the master problem.

We now looking at splitting the cuts further. The main idea is to look at the edges $(i,j) \in \mathscr{A}$ in the routes identified by the master problem. When the subproblem is decoupled, the corresponding edges are also decoupled. Tardiness incurred by a vehicle traveling along (i,j) is denoted by T_{ij}. Consider cut (5.20), replacing T with $T = \sum_{(i,j)\in\mathscr{A}} T_{ij}$ results in

$$\sum_{(i,j)\in\mathscr{A}} T_{ij} \geq T_k^*\big(1 - \sum_{(i,j)\in\mathscr{J}_k} (1 - x_{ij})\big). \tag{5.25}$$

Since $(i,j) \in \mathscr{A} \setminus \mathscr{J}_k$ have no influence on the bound given by cut (5.25), we can replace $\mathscr{A}$ by $\mathscr{J}_k$ and rewrite the cut as

$$\sum_{(i,j)\in\mathscr{J}_k} T_{ij} \geq T_k^*\big(1 - \sum_{(i,j)\in\mathscr{J}_k} (1 - x_{ij})\big), \tag{5.26}$$

Theorem 4 *Cuts* (5.26) *will provide a valid set of cuts to solve the problem to optimality.*

Proof.

Cuts (5.26) can be presented in the form of Benders' cuts $T \geq B_{x^k}(x)$. Where in iteration k

$$T = \sum_{(i,j)\in\mathscr{J}_k} T_{ij}, \quad \text{and} \quad B_{x^k}(x) = T_k^*\big(1 - \sum_{(i,j)\in\mathscr{J}_k} (1 - x_{ij})\big)$$

According to Theorem 1 and Properties 1 and 2, if for any feasible solution (x,y) the total tardiness is bounded such that $T \geq B_{x^k}(x)$, and $B_{x^k}(x^k) = T_k^*$, the Benders' algorithm converges to the optimal value.

We start from proving that $T \geq B_{x^k}(x)$ for any feasible (x,y). Note that, trivially $T \geq \sum_{(i,j)\in\mathscr{J}_k} T_{ij}$.

Let m be an iteration of the Benders' algorithm, such that $m \neq k$. Let set $\mathscr{J}_m$ be defined as $\mathscr{J}_m = \{(i,j)|x_{ij}^m = 1\}$. There can be three cases: $\mathscr{J}_m$ is a subset of $\mathscr{I}_k$, $\mathscr{I}_m$ is a superset of $\mathscr{I}_k$, and the symmetrical set difference $\mathscr{I}_m \triangle \mathscr{I}_k$ is non-empty. In the third case, the indices in $\mathscr{I}_m \setminus \mathscr{I}_k$ do not influence the bound by the definition of the cut, the indices in set $\mathscr{I}_k \setminus \mathscr{I}_m$ correspond to variables $x_{ij}^m = 0$ in the cut and do not influence the bound. Therefore, it is sufficient to consider the former two cases:

- $\mathscr{I}_m \subseteq \mathscr{I}_k$. This gives $\big(1 - \sum_{(i,j)\in\mathscr{I}_k}(1 - x_{ij}^m)\big) \leq 0$, since $\exists (i,j) \in \mathscr{I}_k$, such that $x_{ij}^m = 0$. Therefore $T \geq T_k^*\big(1 - \sum_{(i,j)\in\mathscr{I}_k}(1 - x_{ij})\big)$.
- $\mathscr{I}_k \subseteq \mathscr{I}_m$. This gives $\big(1 - \sum_{(i,j)\in\mathscr{I}_k}(1 - x_{ij}^m)\big) = 1$, since $\forall (i,j) \in \mathscr{I}_k$, $x_{ij}^m = 1$. Therefore $T \geq T_k^*\big(1 - \sum_{(i,j)\in\mathscr{I}_k}(1 - x_{ij})\big)$, because $T \geq T_k^*$.

We now prove that $B_{x^k}(x^k) = T_k^*$ in any iteration k:

$$B_{x^k}(x^k) = T_k^*\big(1 - \sum_{(i,j)\in\mathscr{I}_k}(1 - x_{ij})\big) = T_k^*, \quad \text{since } \forall (i,j) \in \mathscr{I}_k, \quad x_{ij}^k = 1.$$

□

The third set of cuts we propose to generate can be derived by splitting the edges in $\mathscr{I}_k$ by the connected components. Since sets $\mathscr{I}_k^c$ partition $\mathscr{I}_k$, cut (5.26) can be rewritten as the set of cuts

$$\sum_{(i,j)\in\mathscr{I}_k^c} T_{ij} \geq T_{ck}^*\big(1 - \sum_{(i,j)\in\mathscr{I}_k^c}(1 - x_{ij})\big), \quad c \in C_k \tag{5.27}$$

$$T \geq \sum_{c\in C_k} \sum_{(i,j)\in\mathscr{I}_k^c} T_{ij}. \tag{5.28}$$

The main difference from previously introduced cuts is that new auxiliary variables T_{ij} are added. Compared to $T_c k$ variables, new variables further partition the total tardiness.

Theorem 5 *Cuts* (5.27)–(5.28) *will provide a valid set of cuts to solve the problem* *to optimality*

Proof. A logic similar to the proof of Theorem 4 can be applied here. Let T_{ck} be defined as $T_{ck} = \sum_{(i,j)\in\mathscr{J}_k^c} T_{ij}$. We first prove that $T_{ck} \geq T_{ck}^*\big(1-\sum_{(i,j)\in\mathscr{J}_k^c}(1-x_{ij})\big)$ for any feasible solution (x,y).

Let $\mathscr{J}_m$ be the master problem solution in iteration m. The set J_m can be split by the connected components obtained in iteration k such that $\bigcup_{c\in C_k}\mathscr{J}_m^c = \mathscr{J}_m$. Similar to the proof of Theorem 4, it is sufficient to consider the following two cases:

- $\mathscr{J}_m^c \subseteq \mathscr{J}_k^c$. This gives $\big(1-\sum_{(i,j)\in\mathscr{J}_k^c}(1-x_{ij}^m)\big) \leq 0$, since $\exists(i,j)\in\mathscr{J}_k^c$, such that $x_{ij}^m = 0$. Therefore $T_{ck} \geq T_{ck}^*\big(1-\sum_{(i,j)\in\mathscr{J}_k^c}(1-x_{ij})\big)$.
- $\mathscr{J}_k^c \subseteq \mathscr{J}_m$. This gives $\big(1-\sum_{(i,j)\in\mathscr{J}_k^c}(1-x_{ij}^m)\big) = 1$, since $\forall(i,j)\in\mathscr{J}_k^c,\ x_{ij}^m = 1$. Therefore $T_{ck} \geq T_{ck}^*\big(1-\sum_{(i,j)\in\mathscr{J}_k^c}(1-x_{ij})\big)$

□

5.7 Computational experiments

The computational effectiveness of subproblem separation and various Benders' cuts, described in Section 5.6, is evaluated in a series of computational experiments. The evaluated cuts are cuts (5.22), cuts (5.23)–(5.24), cuts (5.27), and the combination of cuts (5.23)–(5.24) and cuts (5.27). Using the combination of cuts (5.23)–(5.24) and cuts (5.27) means generating both sets of cuts in each iteration. Each experiment solves the VRPLC with the minimising total tardiness objective. The experiments run the LBBD scheme with the separated subproblem generating each type of cuts separately. Another experiment runs the default implementation of the LBBD scheme with no subproblem separation. The different implementations are referred to as ”methods” for the sake of brevity. Since it is important to apply cut strengthening to accelerate the solution process, the deletion filter cut-strengthening technique with the default variable sorting has been applied in all computational runs.

The main metric of computational effectiveness is the impact of different methods on the run time of the LBBD scheme. For the run time plots, the horizontal axis gives the time and the vertical axis gives the percentage of solved instances. A point (x,y) on the curve means that y% of instances can be solved in less than x seconds.

5.7.1 Experiment setting

The LBBD scheme is implemented in Python 3.8, and the MIP and CP models are solved using Gurobi Optimizer version 9.1.2 and IBM ILOG CP Optimizer version 20.1, respectively. All tests have been carried out on a computer with two Intel Xeon Gold 6130 processors (16 cores, 2.1 GHz each) and 96 GB RAM. Each instance was given a total time of 20 minutes and the MIP-gaps are set to 0 for the master problems.

5.7.2 Instances

We use 450 instances from Lam et al. Lam et al., 2020, the instances are available at https://github.com/ed-lam/nutmeg/tree/master/examples/vrplc/Instances. The instances are generated for 5, 8, or 10 locations. For each number of locations, there are instances with 20, 30, and 40 requests. Location resource capacities vary between one and eight for all instances. Since the instances were originally created for feasibility and makespan objective functions, the instances are modified for the minimising total tardiness objective with deadlines decreased by 10%.

5.7.3 Percentage of solved instances

The main observation from Figure 5.2 is that all of the methods applying subproblem separation outperform the default implementation. The default implementation with 12% of solved instances is notably behind the methods applying subproblem separation. Cuts (5.23)–(5.24) have the highest effectiveness with 81.5% of solved instances. Cuts (5.22) and the combination of cuts (5.23)–(5.24) and (5.27) have marginally lower percentages of solved instances—77.8% and 80%, respectively.

An interesting result is that although cuts (5.27) outperform the default method, they show significantly lower results than other methods applying subproblem separation—28.2% of solved instances. The main reason is that the cuts lead to repeated master problem solutions with the same connected components. This implies that this type of cut introduces symmetry that is difficult to handle for the master problem solver. The results for the combination of cuts (5.27) and cuts (5.23)–(5.24) being slightly lower than the results for cuts (5.23)–(5.24) also imply that cuts (5.27) adversely impact the run time. As it was expected, Figure 5.3 shows that optimality cuts (5.23)–(5.24), cuts (5.22), and the combination of cuts (5.23)–(5.24) have a similar average cut size, with most of the cuts being sparse. It is also noticeable that cuts (5.27) did not generate cuts of full size.

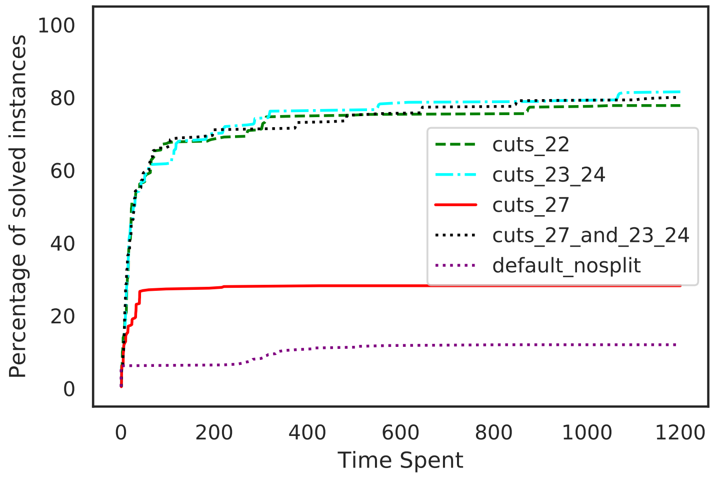


Figure 5.2: Percentage of instances solved to optimality for the minimising tardiness for cumulative scheduling problem.

The results in Table 5.2 highlight the effect of subproblem separation on the LBBD scheme. The reported values are calculated for different sets for each method. For each method, the set comprises retrieved instances that were either solved to optimality or timed out. The N_{inst} column indicates the number of instances in each set, this includes solved instances and the instances that

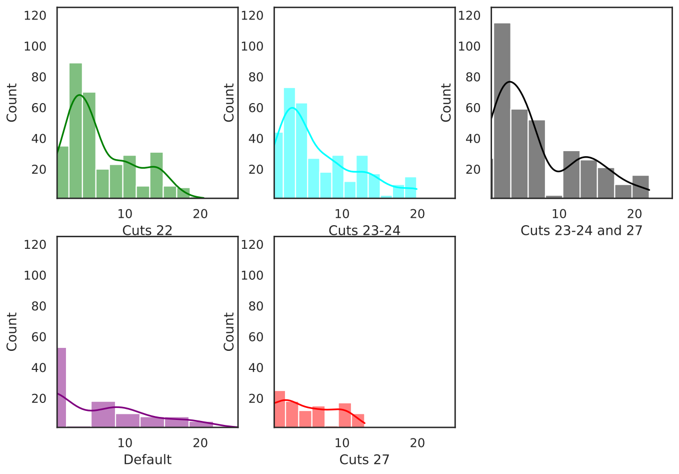


Figure 5.3: Average size of optimality cuts for the minimising tardiness for cumulative scheduling problem.

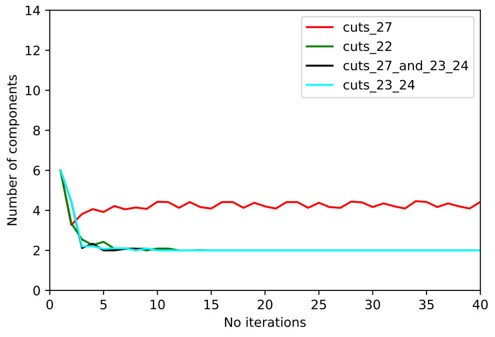
timed out. Note that if an instance times out while solving the subproblem, the results cannot be retrieved. The number of instances solved to optimality by each method is given in the last column. As can be seen in Table 5.2, the default implementation spends much more time solving subproblem per instance—3.75 seconds compared to 0.141 seconds and below by other methods. This can be explained by the greater average subproblem solution time per Benders' iteration—0.67 seconds compared to 0.02 seconds, 0.016 seconds, 0.03 seconds, and 0.015 seconds by cuts (5.22), cuts (5.23)–(5.24), cuts (5.27), and the combination of cuts (5.23)–(5.24) and cuts (5.27), respectively. This result shows that it takes less time to solve multiple smaller subproblems than to solve one subproblem. Another interesting observation is that the default implementation leads to a higher number of Benders' iterations and subproblems solved, this suggests that the cuts generated by the default implementation are less effective than the cuts generated by the other methods.

5.7.4 Connected components

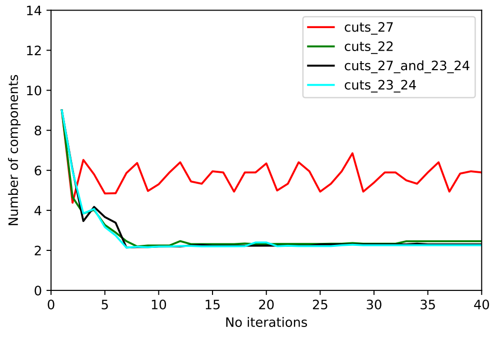
Another interesting question is what problem structures various types of cuts lead to. Different cuts lead to different master problem solutions, which in turn lead to different blocks in subproblem separation. Figure 5.4 gives information about

Method	T_{MP}	T_{sub}	T_{sub} *(per iter)*	N_{sub}	N_{iter}	N_{inst}	N_{inst} *(solved)*
Default	1.36	3.75	0.67	1453	40.3	105	53
Cuts 22	0.92	0.138	0.02	430	11.25	352	349
Cuts 23-24	1.09	0.141	0.016	578	17.6	367	366
Cuts 27	0.26	0.065	0.03	108	3.3	126	126
Combination	1.19	0.068	0.015	502	12.11	361	359

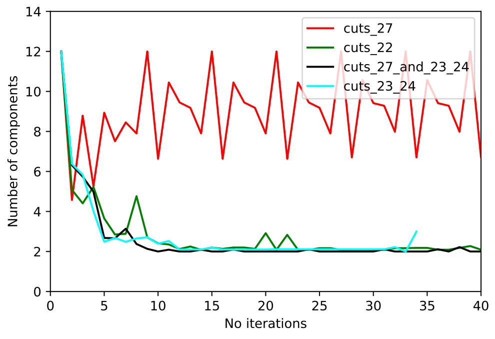
Table 5.2: The table presents the average master problem solution time, average subproblem solution time, and number of subproblems solved per instance. The instances for which the results were not retrieved within 20 minutes are omitted



Instances with 5 locations



Instances with 8 locations



Instances with 11 locations

Figure 5.4: Results for the default DFBS implementation

the subproblem structure by showing how the average number of components changes in each Benders' iteration for different instance sets. For each instance set, the maximum number of components is the number of locations plus one central depot. The changes in the number of components are shown for 40 Benders' iterations.

Interestingly, it can be seen in Figure 5.4 that the first master solution always leads to subproblem structure with all of the locations separated. In the first 2 or

3 iterations the number of components decreases for all types of cuts, meaning that some of the locations are connected. This decreasing tend to continues until the number of components converges at 2 (with some small fluctuations) for all the cuts except for cuts (5.27). Cuts (5.27) repeats a certain pattern in each plot, this repetition again implies that these cuts lead to symmetry in the solutions. This means that the algorithm keeps trying symmetric solution that do not leat to convergence of the algorithm.

5.8 Conclusions

This chapter proposes a new implementation of the LBBD scheme for the vehicle routing problem with local congestion. We propose using the connected components algorithm to identify separable blocks of the subproblem. The new implementation reformulates the separated subproblem in each Benders' algorithm iteration. This method of separating the subproblem can be applied to other vehicle routing problems with vehicle capacity and congestion constraints. Since the new reformulation requires new Bender's cuts, we derive various types of cuts. We then evaluate subproblem separation and new Benders' cuts in computational experiments.

The main conclusion is that subproblem separation is an effective technique for accelerating the LBBD scheme for the vehicle routing problem with local congestion. However, in order to fully exploit the new subproblem structure, it is important to generate strong cuts. Splitting the cuts by the connected components showed the best computational results. Whereas, splitting the cuts by edges was not effective. An area of future work is to investigate methods to handle the difficulty introduces by these cuts.

Chapter 6

Summary and Conclusion

Logic-based Benders' decomposition (LBBD) is a solution approach introduced by Hooker and Ottosson (2003) as an extension to classical Benders' decomposition. LBBD decomposes a given optimisation problem into a master problem, which is often solved as a mixed-integer program (MIP), and a subproblem, which is often solved as a constraint program (CP). The LBBD solution scheme is a finite iterative algorithm, which guarantees optimality of the solution. The convergence of the algorithm depends on the strength of the generated Benders' cuts. This thesis proposes various improvements to cut generation algorithms in order to accelerate LBBD. The key features of this thesis are i) computational evaluation of cut-strengthening techniques, ii) variable sorting heuristics, and iii) a subproblem separation method.

The cut-strengthening techniques investigated in this thesis are the greedy algorithm, deletion filter, additive method, additive/deletion filter, and depth-first binary search. The computational evaluation of cut-strengthening techniques is presented in Chapter 3, which gives an in-depth discussion of each of the cut-strengthening techniques. The computational experiments in Chapter 3 cover cumulative facility scheduling with fixed costs, single-facility scheduling with a segmented timeline, and vehicle routing with local congestion problems. The key contribution of Chapter 3 is the first systemic evaluation of cut-strengthening techniques for both feasibility and optimality Benders' cuts. The main metrics of effectiveness of cut-strengthening techniques are the percentage of instances solved, number of Benders' iterations, subproblem solution time, and the overall solution time. Computational results in Chapter 3 show that cut-strengthening techniques that generate irreducible cuts outperform the greedy algorithm and no cut strengthening. The results also demonstrate that due to the lowest subproblem solution times, deletion filter and DFBS have the highest computational effectiveness overall. Chapter 3 showed that although subproblem separation often dominated the effectiveness of cut-strengthening techniques, cut-strengthening techniques and subproblem separation can be used interchangeably for the acceleration of LBBD.

The main contribution of Chapter 4 are the novel heuristics for variable sorting. A preliminary investigation of variable sorting strategies is presented in Chapter 4. Based on this investigation, three new heuristics for variable sorting are proposed. The new heuristics have been proposed for DFBS, which Chapter 3 determined as one of the most computationally effective techniques. However, the heuristics can be applied to other cut-strengthening techniques. Moreover, *weights* and *dynamic size* heuristics do not depend on the problem structure, and can therefore be applied to solving any problem. The computational results in Chapter 4 show that applying variable sorting increases the efficiency of DFBS and leads to the lower overall solution time.

Chapter 5 proposes a method of subproblem separation using the example of the vehicle routing problem with local congestion, for which the subproblem does not separate naturally. The main contribution of Chapter 5 is a subproblem separation technique based on a connected components algorithm. The proposed technique allows separating the subproblem by locations that have no connecting arcs between them. Another important contribution of Chapter 5 are the new types of Benders' cuts. The results in Chapter 5 show that subproblem separation applied together with certain types of Benders' cuts achieves a significant acceleration of LBBD. This result highlights that the type of Benders' cut to generate is an important decision for cut generation algorithms, which can negatively affect the overall acceleration of LBBD.

Outlook

This thesis demonstrates how improvements of cut generation algorithms accelerate the LBBD solution scheme. The results from each chapter show that effective convergence of LBBD requires various enhancements like cut-strengthening techniques, subproblem separation, valid inequalities etc. Further research into these techniques can help realise the full potential of LBBD.

One of the limitations of the studies in thesis is that the investigated problems assume a specific problem structure presented in Section 3.3.1.2. This problem structure facilitates the use of cut-strengthening techniques. One of the potential avenues of future research is to look into other theoretical assumptions that would make using cut-strengthening techniques valid.

Chapter 3 showed that cut-strengthening techniques perform differently depending on the objective function. More theoretical research could be done to investigate the impact of the type of the objective functions on the computational effectiveness of the techniques. A special interest are the objective functions leading to optimisation subproblems.

Chapter 4 shows that there is limited previous research on various heuristics for cut strengthening in LBBD. More computational and theoretical results could be established by studying the problem structure and valid inequalities in LBBD. An interesting practical question from Chapter 5 is handling the difficulty introduce by the cuts of type (5.27).

Computational results in this thesis showed that although various cut-strengthening techniques generate cuts of similar size, they often lead to different master problem solutions and numbers of Benders' iterations. It would be worthwhile to investigate the difference between the generated cuts. The deeper look into what cuts are being generated might yield interesting results.

From the practical point of view, it would be interesting to apply cut generation algorithms studied in this thesis to a large-scale industrial application. The improvements to cut generation algorithms were studied independently , so it would be interesting to see the results of applying them simultaneously. New formulations, new valid inequalities and other important decisions in applying LBBD might also be possible when looking at new practical applications.

Bibliography

Arnold, F., Gendreau, M., & Sörensen, K. (2019). Efficiently solving very large-scale routing problems. *Computers & Operations Research*, *107*, 32–42 (Cited on page 121).

Atlihan, M. K., & Schrage, L. (2008). Generalized filtering algorithms for infeasibility analysis. *Computers & Operations Research*, *35*, 1446–1464. https://doi.org/10.1016/j.cor.2006.08.005 (Cited on pages 56, 95 and 96)

Benders, J. (1962). Partitioning procedures for solving mixed-variable programming problems. *Numerische Mathematik*, *4* (Cited on pages 14 and 30).

Benini, L., Lombardi, M., Mantovani, M., Milano, M., & Ruggiero, M. (2008). Multi-stage Benders decomposition for optimizing multicore architectures. In L. Perron & M. A. Trick (Eds.), *CPAIOR 2008: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (pp. 36–50). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-68155-7_6. (Cited on pages 41 and 42)

Benjamin, A. M., & Beasley, J. E. (2010). Metaheuristics for the waste collection vehicle routing problem with time windows, driver rest period and multiple disposal facilities. *Computers & Operations Research*, *37*(12), 2270–2280 (Cited on page 121).

Berbeglia, G., Cordeau, J.-F., Gribkovskaia, I., & Laporte, G. (2007). Static pickup and delivery problems: A classification scheme and survey. *Top*, *15*, 1–31 (Cited on page 121).

Bockmayr, A., & Hooker, J. N. (2005). Constraint programming. *Handbooks in Operations Research and Management Science*, *12*, 559–600 (Cited on page 25).

Boussemart, F., Hemery, F., Lecoutre, C., & Sais, L. (2004). Boosting systematic search by weighting constraints. *ECAI*, *16*, 146 (Cited on pages 96, 97 and 98).

Cambazard, H., Hladik, P.-E., Déplance, A.-M., Jussien, N., & Trinquet, Y. (2004). Decomposition and Learning for a Hard Real Time Task Allocation. In M. Wallace (Ed.), *Principles and Practice of Constraint Programming – cp 2004* (pp. 153–167). Springer. https://doi.org/10.1007/978-3-540-30201-8_14. (Cited on pages 42, 44, 56 and 96)

Carlier, J. (1982). The one-machine sequencing problem. *European Journal of Operational Research*, *11*(1), 42–47. https://doi.org/https://doi.org/10.1016/S0377-2217(82)80007-6 (Cited on page 44)

Chinneck, J. W. (1997). Feasibility and viability. In T. Gal & H. J. Greenberg (Eds.), *Advances in sensitivity analysis and parametic programming* (pp. 491–531). Springer US. https://doi.org/10.1007/978-1-4615-6103-3_14. (Cited on page 53)

Chinneck, J. W., & Dravnieks, E. W. (1991). Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, *3*(2), 157–168. https://doi.org/10.1287/ijoc.3.2.157 (Cited on pages 29, 52 and 53)

Ciré, A. A., Çoban, E., & Hooker, J. N. (2016). Logic-based Benders decomposition for planning and scheduling: a computational analysis. *The Knowledge Engineering Review*, *31*(5), 440–451. https://doi.org/10.1017/S0269888916000254 (Cited on pages 40 and 50)

Clarke, G., & Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, *12*(4), 568–581 (Cited on page 121).

Coban, E., & Hooker, J. N. (2013). Single-facility scheduling by logic-based Benders decomposition. *Annals of Operations research*, *210*, 245–272. https://doi.org/10.1007/s10479-011-1031-z (Cited on pages 39, 42, 43, 63, 65, 74, 76 and 119)

Colmerauer, A. (1990). An introduction to Prolog III. *Communications of the ACM*, *33*(7), 69–90 (Cited on page 24).

Colmerauer, A., Kanoui, H., Pasero, R., & Roussel, P. (1972). Un système de communication en français. *Groupe D'intelligence Artificielle, Faculté Des Sciences De Luminy, Marseille* (Cited on page 23).

Dantzig, G. B. (1957). Discrete-variable extremum problems. *Operations research*, *5*(2), 266–288 (Cited on page 20).

Dincbas, M., Simonis, H., & Van Hentenryck, P. (1988). Solving the car-sequencing problem in constraint logic programming. *ECAI*, *88*, 290–295 (Cited on page 24).

Emde, S., Polten, L., & Gendreau, M. (2020). Logic-based Benders decomposition for scheduling a batching machine. *Computers & Operations Research*, *113*, 104777 (Cited on page 119).

Gleeson, J., & Ryan, J. (1990). Identifying minimally infeasible subsystems of inequalities. *ORSA Journal on Computing*, *2*(1), 61–63 (Cited on page 29).

Greenberg, H. J., & Murphy, F. H. (1991). Approaches to diagnosing infeasible linear programs. *ORSA Journal on Computing*, *3*(3), 253–261 (Cited on page 29).

Guieu, O., & Chinneck, J. W. (1999). Analyzing infeasible mixed-integer and integer linear programs. *INFORMS Journal on Computing*, *11*(1), 63–77 (Cited on pages 28, 29, 94 and 95).

Haralick, R. M., & Elliott, G. L. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, *14*(3), 263–313 (Cited on page 97).

Hemery, F., Lecoutre, C., Sais, L., Boussemart, F., et al. (2006). Extracting MUCs from constraint networks. *ECAI*, *6*, 113–117 (Cited on page 98).

Hooker, J. N. (2007). Planning and scheduling by logic-based Benders decomposition. *Operations research*, *55*, 588–602. https://doi.org/10.1287/opre.1060.0371 (Cited on pages 39, 41, 42, 43, 46, 47, 49, 58, 60, 63, 73, 76, 79, 98, 106, 107, 112, 119 and 127)

Hooker, J. N. (2000). *Logic-based methods for optimization: Combining optimization and constraint satisfaction*. John Wiley; Sons. (Cited on pages 14, 27, 34 and 119).

Hooker, J. N. (2019). Logic-based Benders decomposition for large-scale optimization. In *Large scale optimization in supply chains and smart manufacturing: Theory and applications* (pp. 1–26). Springer International Publishing. https://doi.org/10.1007/978-3-030-22788-3_1. (Cited on pages 41 and 119)

Hooker, J. N., & Ottosson, G. (2003). Logic-based Benders decomposition. *Mathematical Programming*, *96*(1), 33–60 (Cited on pages 14, 15, 34, 39, 47, 49, 119, 127 and 139).

Hooker, J. N., & Yan, H. (1995). Logic circuit verification by Benders decomposition. *Principles and practice of constraint programming: the newport papers*, 267–288 (Cited on page 34).

Jaffar, J., & Lassez, J.-L. (1987). Constraint logic programming. *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, 111–119 (Cited on page 24).

Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., & Wolsey, L. A. (2009). *50 years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media. (Cited on page 14).

Junker, U. (2004). QuickXPlain: Preferred explanations and relaxations for over-constrained problems. *In Proceedings of AAAI-2004*, 167–172 (Cited on pages 56 and 95).

Junker, U. (2001). Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, *4* (Cited on page 44).

Karlsson, E., & Rönnberg, E. (2021). Strengthening of feasibility cuts in logic-based benders decomposition. In P. J. Stuckey (Ed.), *Integration of constraint programming, artificial intelligence, and operations research* (pp. 45–61).

Springer International Publishing. https://doi.org/10.1007/978-3-030-78230-6_3. (Cited on pages 16, 40, 41, 42, 44, 45, 50, 51, 75, 91 and 96)

Karlsson, E., & Rönnberg, E. (2022). Logic-based Benders decomposition with a partial assignment acceleration technique for avionics scheduling. *Computers & Operations Research*, *146*, 105916. https://doi.org/https://doi.org/10.1016/j.cor.2022.105916 (Cited on pages 41, 42, 43, 96, 118 and 119)

Kowalski, R. (1974). Predicate logic as programming language. *IFIP congress*, *74*, 569–544 (Cited on page 23).

Lam, E., Gange, G., Stuckey, P. J., Van Hentenryck, P., & Dekker, J. J. (2020). Nutmeg: A MIP and CP hybrid solver using branch-and-check. *SN Operations Research Forum*, *1*, 22:1–22:27. https://doi.org/10.1007/s43069-020-00023-2 (Cited on pages 39, 41, 42, 43, 68, 76, 119, 122 and 134)

Lam, E., Pardalos, P. M., & Hentenryck, P. V. (2016). A branch-and-price-and-check model for the vehicle routing problem with location congestion. *Constraints*, *21*, 394–412 (Cited on pages 68 and 121).

Lindh, E., Olsson, K., & Rönnberg, E. (2022). Scheduling of an underground mine by combining logic-based Benders decomposition and a priority-based heuristic. *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling—PATAT, Leuven, Belgium*, 2–30 (Cited on pages 41, 42 and 43).

Lombardi, M., & Milano, M. (2012). Optimal methods for resource allocation and scheduling: A cross-disciplinary survey. *Constraints*, *17*, 51–85 (Cited on page 119).

Markowitz, H. M., & Manne, A. S. (1957). On the solution of discrete programming problems. *Econometrica: journal of the Econometric Society*, 84–110 (Cited on page 20).

Marriott, K., & Stuckey, P. J. (1998). *Programming with constraints: An introduction*. MIT press. (Cited on page 24).

Pecin, D., Pessoa, A., Poggi, M., & Uchoa, E. (2017). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, *9*, 61–100 (Cited on page 121).

Rahmaniani, R., Crainic, T. G., Gendreau, M., & Rei, W. (2017). The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, *259*, 801–817. https://doi.org/10.1016/j.ejor.2016.12.005 (Cited on page 41)

Raidl, G. R., Baumhauer, T., & Hu, B. (2014). Speeding up Logic-Based Benders' Decomposition by a Metaheuristic for a Bi-Level Capacitated Vehicle Routing Problem. *International Workshop on Hybrid Metaheuristics*, 183–197 (Cited on pages 119 and 121).

Raidl, G. R., Baumhauer, T., & Hu, B. (2015). Boosting an Exact Logic-Based Benders Decomposition Approach by Variable Neighborhood Search. *Electronic Notes in Discrete Mathematics*, *47*, 149–156 (Cited on pages 119 and 121).

Ralphs, T. K., Kopman, L., Pulleyblank, W. R., & Trotter, L. E. (2003). On the capacitated vehicle routing problem. *Mathematical Programming*, *94*, 343–359 (Cited on page 121).

Riazi, S., Seatzu, C., Wigström, O., & Lennartson, B. (2013). Benders/gossip methods for heterogeneous multi-vehicle routing problems. *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, 1–6 (Cited on pages 119 and 121).

Riedler, M., & Raidl, G. (2018). Solving a selective dial-a-ride problem with logic-based Benders decomposition. *Computers & Operations Research*, *96*, 30–54. https://doi.org/https://doi.org/10.1016/j.cor.2018.03.008 (Cited on pages 41, 42 and 43)

Sadykov, R. (2008). A branch-and-check algorithm for minimizing the weighted number of late jobs on a single machine with release dates. *European Journal of Operational Research*, *189*(3), 1284–1304. https://doi.org/https://doi.org/10.1016/j.ejor.2006.06.078 (Cited on pages 41, 42 and 44)

Sun, D., Tang, L., & Baldacci, R. (2019). A Benders decomposition-based framework for solving quay crane scheduling problems. *European Journal of Operational Research*, *273*(2), 504–515 (Cited on page 119).

Tamiz, M., Mardle, S., & Jones, D. (1996). Detecting IIS in infeasible linear programmes using techniques from goal programming. *Computers & Operations Research*, *23*(2), 113–119. https://doi.org/https://doi.org/10.1016/0305-0548(95)00018-H (Cited on page 53)

Thorsteinsson, E. S. (2001). Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. *International conference on principles and practice of constraint programming*, 16–30 (Cited on page 44).

Tsang, E. (1993). Foundations of constraing satisfaction. *Academic Press* (Cited on page 24).

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Vidal, T., & Subramanian, A. (2017). New benchmark instances for the Capacitated Vehicle Routing Problem. *European Journal of Operational Research*, *257*(3), 845–858 (Cited on page 121).

Van Hentenryck, P. (1989). *Constraint satisfaction in logic programming*. MIT press. (Cited on page 24).

Van Loon, J. (1981). Irreducibly inconsistent systems of linear inequalities. *European Journal of Operational Research*, *8*(3), 283–288. https://doi.org/https://doi.org/10.1016/0377-2217(81)90177-6 (Cited on page 28)

Winston, W. L. (2004). *Operations research: Applications and algorithm*. Thomson Learning, Inc. (Cited on page 14).

Wolsey, L. A., & Nemhauser, G. L. (1999). *Integer and combinatorial optimization* (Vol. 55). John Wiley & Sons. (Cited on pages 14 and 21).

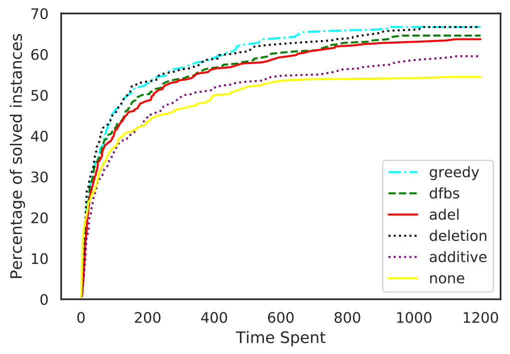
Appendices

Appendix A

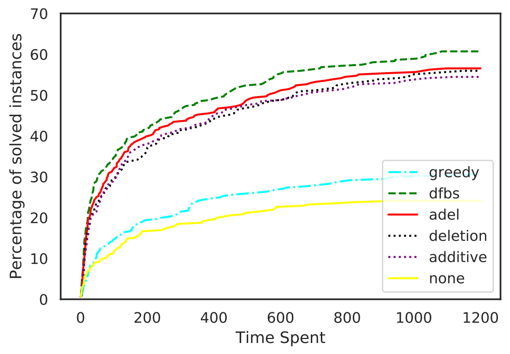
Analytic cuts

A.1 Minimising makespan for cumulative facility scheduling problem

As can be seen in Figure A.1a The deletion filter and the greedy algorithm give the best performance for the split subproblem with 66.7% of the instances solved to optimality for both methods. Similarly, the DFBS and the additive/deletion with respective percentages of solved instances of 64.58% and 63.69% show better results than the additive method and no cut strengthening. Given that splitting the subproblem gives the performance rise from 24.11% to 54.46% for no cut strengthening, we can tell that substantial time improvement comes from the subproblem separation. Figure A.1b shows that all of the cut-strengthening techniques except for the greedy algorithm used for the no-split subproblem either outperform or match the performance of the split subproblem with no cut strengthening.



Split

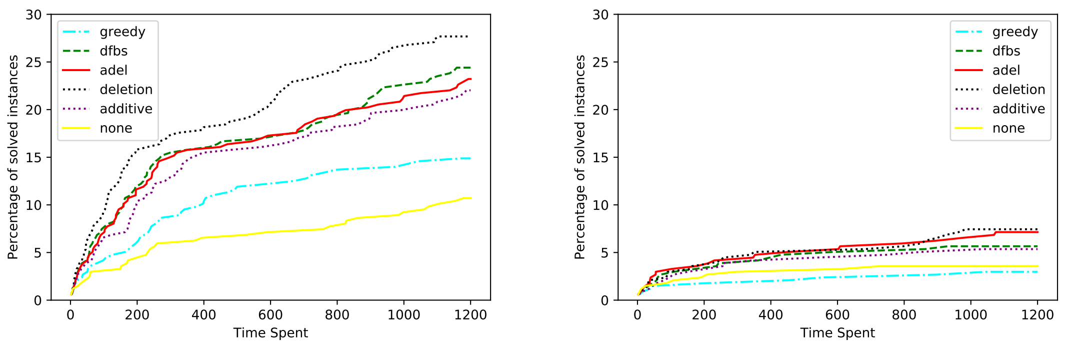


No-split

Figure A.1: Cumulative scheduling:Percentage of solved instances for minimising makespan problem with analytic optimality cuts

A.2 Minimising tardiness for cumulative facility scheduling problem

It can be seen in Figure A.2b that the deletion filter shows the best performance for the no-split subproblem with 7.4% of instances solved to optimality. This results is notably lower than 10.71% of solved instances that correspond to no cut strengthening for the split subproblem. The deletion filter shows the best result for the split subproblem with 27.67% of solved instances.



Split

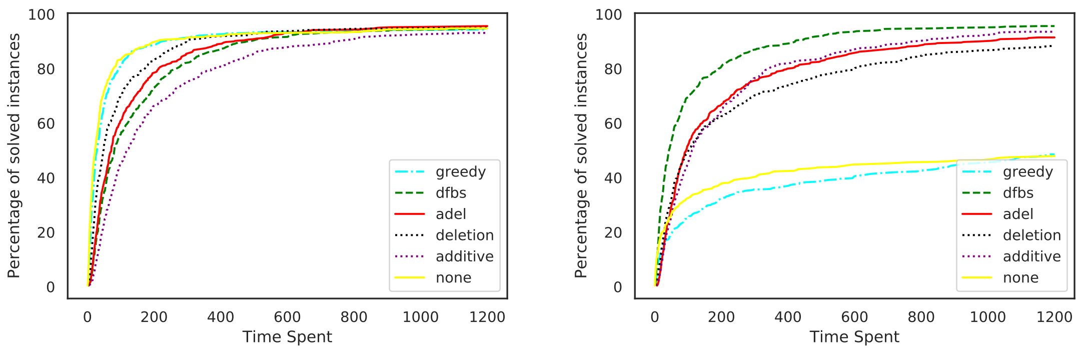
No-split

Figure A.2: Cumulative scheduling: Percentage of solved instances for minimising tardiness problem with analytic optimality cuts

A.3 Minimising makespan for disjunctive scheduling problem

Using no cut strengthening for the split subproblem solves 94.79% for analytic cuts. The additive/deletion filter, the best-performing algorithm for analytic cuts, improves the results by 0.83% with 95.62% of solved instances. Another interesting observation is that using the additive method for the split subproblem solves 93.13% of instances, which is worse than no cut strengthening. The cut size Figure 3.12 shows that the additive method generates cuts of similar size to other cut-strengthening techniques. However, it performs many more subproblems. As a results, it spends much more time solving the subproblems.

All of the cut-strengthening techniques except for the greedy have a considerable impact on the computational performance. Using the DFBS for the no-split subproblem solves 95.62% of instances. This results matches the best performance for the split subproblem.

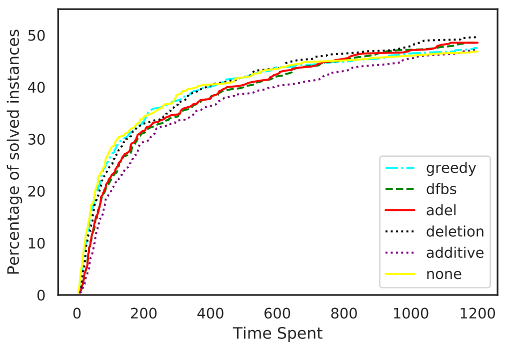


Split No-split

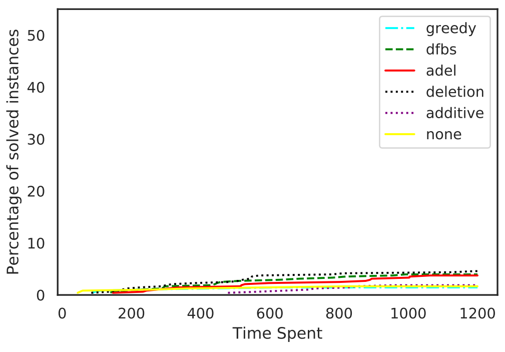
Figure A.3: Disjunctive scheduling: Percentage of solved instances for minimising makespan with analytic cuts

A.4 Minimising tardiness for disjunctive scheduling problem

Using analytic cuts for the split subproblem solves 46.88% of instances compared to 1.67% for the no-split subproblem. It appears that the impact of cut-strengthening techniques does not differ much when comparing split and no-split results. For example, the deletion filter gives a performance rise of 2.91% of instances solved for the split subproblem, and 2.71% of instances solved for the no-split subproblem.



Split



No-split

Figure A.4: Disjunctive scheduling: Percentage of solved instances for minimising tardiness with analytic cuts

Appendix B

Tables with additional data

problem	cut_str	N_{iter}	N_{sub}	T_{sub}	T_{mas}	N_{inst}
no_str	Greedy	64.58	366.13	179.93	1.93	62
	DFBS	11.0	131.45	63.01	0.12	62
	AdDel	11.0	75.45	52.68	0.13	62
	Deletion	10.9	146.66	94.05	0.13	62
	Additive	11.1	222.03	66.56	0.13	62
	None	409.76	409.82	105.73	71.62	62
no_an	Greedy	57.87	324.74	163.58	1.95	69
	DFBS	10.33	118.93	54.78	0.12	69
	AdDel	10.51	71.3	53.98	0.13	69
	Deletion	10.33	139.52	81.34	0.13	69
	Additive	10.55	206.46	65.85	0.13	69
	None	337.23	337.3	93.08	56.84	69
sp_str	Greedy	38.09	387.93	84.19	5.58	153
	DFBS	17.38	679.6	107.05	1.48	153
	AdDel	17.4	379.91	113.21	1.24	153
	Deletion	17.32	417.82	81.17	1.36	153
	Additive	17.38	1053.56	188.55	1.26	153
	None	93.08	346.67	88.76	25.55	153
sp_an	Greedy	32.38	364.7	70.92	10.1	160
	DFBS	17.18	762.74	110.18	2.9	160
	AdDel	17.2	431.67	116.72	2.94	160
	Deletion	16.71	454.84	78.14	2.98	160
	Additive	17.34	1204.62	187.64	3.15	160
	None	67.84	267.49	57.31	29.43	160

The results are only compared for instances that were solved by all of the methods

Table B.1: Cumulative scheduling: Results for the variants of the Minimising Makespan problem

problem	cut_str	N_{iter}	N_{sub}	T_{sub}	T_{mas}	N_{inst}
no_str	Greedy	125.71	330.57	175.1	4.23	7
	DFBS	27.0	558.29	138.39	0.35	7
	AdDel	27.0	265.0	140.39	0.36	7
	Deletion	27.0	312.43	118.49	0.33	7
	Additive	27.0	1047.0	121.04	0.35	7
	None	122.14	122.43	63.84	7.11	7
no_an	Greedy	125.71	330.57	175.26	4.44	7
	DFBS	27.0	558.29	138.17	0.35	7
	AdDel	15.86	156.57	99.52	0.2	7
	Deletion	27.0	312.43	118.32	0.34	7
	Additive	27.0	1047.0	120.76	0.36	7
	None	122.14	122.43	63.69	7.2	7
sp_str	Greedy	126.17	849.74	77.61	14.79	80
	DFBS	54.2	1359.97	134.31	4.04	80
	AdDel	52.33	720.12	131.07	3.93	80
	Deletion	53.09	780.8	77.75	3.86	80
	Additive	53.54	2032.7	164.37	3.84	80
	None	268.38	864.85	74.12	52.96	80
sp_an	Greedy	216.69	1289.53	124.14	27.27	32
	DFBS	74.03	1612.75	167.22	5.78	32
	AdDel	74.19	899.16	174.34	5.7	32
	Deletion	74.09	947.22	102.24	5.67	32
	Additive	74.38	2442.06	198.59	5.87	32
	None	441.28	1239.97	105.48	105.21	32

The results are only compared for instances that were solved by all of the methods

Table B.2: Cumulative scheduling: Results for the variants of the Minimising Tardiness problem

problem	cut_str	N_{iter}	N_{sub}	T_{sub}	T_{mas}	N_{inst}
No-split	Greedy	10.2	65.43	22.5	0.27	125
	DFBS	3.98	45.0	22.31	0.04	125
	AdDel	3.95	27.41	22.48	0.04	125
	Deletion	4.21	50.93	22.62	0.04	125
	Additive	6.08	88.03	22.55	0.09	125
	None	36.02	37.02	22.82	2.53	125
Split	Greedy	8.94	59.3	0.04	0.36	202
	DFBS	5.49	104.21	0.04	0.14	202
	AdDel	5.54	64.33	0.04	0.15	202
	Deletion	5.49	70.66	0.04	0.14	202
	Additive	7.74	163.1	0.04	0.24	202
	None	17.03	57.79	0.04	0.81	202

The results are only compared for instances that were solved by all of the methods

Table B.3: Cumulative scheduling: Results for the variants of the Minimising Total Cost problem

problem	cut_str	N_{iter}	N_{sub}	T_{sub}	T_{mas}	N_{inst}
no_str	Greedy	17.39	406.78	100.8	48.02	179
	DFBS	4.91	106.79	28.28	25.64	179
	AdDel	5.14	198.61	49.38	24.03	179
	Deletion	5.02	375.66	90.94	19.93	179
	Additive	4.99	541.09	79.74	20.56	179
	None	30.91	31.91	5.92	73.96	179
no_an	Greedy	17.39	406.78	100.03	48.05	179
	DFBS	4.91	106.79	28.14	25.6	179
	AdDel	5.14	198.61	88.95	23.93	179
	Deletion	5.02	375.66	109.92	19.92	179
	Additive	4.99	541.09	79.26	20.55	179
	None	30.91	31.91	5.91	73.96	179
sp_str	Greedy	7.14	242.47	21.63	30.21	428
	DFBS	5.78	1046.75	92.84	26.23	428
	AdDel	5.59	491.89	78.84	22.97	428
	Deletion	5.69	506.39	45.41	25.83	428
	Additive	5.68	1695.06	150.89	24.53	428
	None	7.84	120.3	11.0	32.34	428
sp_an	Greedy	6.83	234.42	20.91	24.24	425
	DFBS	6.25	1144.71	101.71	26.39	425
	AdDel	5.83	521.09	84.07	26.54	425
	Deletion	6.09	555.2	49.93	24.91	425
	Additive	5.78	1694.09	150.78	24.72	425
	None	8.11	127.13	11.66	26.85	425

The results are only compared for instances that were solved by all of the methods

Table B.4: Disjunctive scheduling: Results for the variants of the Minimising Makespan problem

problem	cut_str	N_{iter}	N_{sub}	T_{sub}	T_{mas}	N_{inst}
no_str	Greedy	111.25	500.5	142.42	80.86	4
	DFBS	10.75	507.5	149.69	3.25	4
	AdDel	10.75	347.5	182.08	3.23	4
	Deletion	10.75	349.5	117.4	3.18	4
	Additive	10.75	2075.5	591.15	3.23	4
	None	128.75	129.75	44.37	96.23	4
no_an	Greedy	104.0	372.0	118.44	49.56	3
	DFBS	12.67	631.0	191.93	3.03	3
	AdDel	12.67	381.0	214.07	3.07	3
	Deletion	12.67	382.0	140.71	3.01	3
	Additive	12.67	2522.67	748.05	3.09	3
	None	119.67	120.67	45.54	62.23	3
sp_str	Greedy	20.79	434.3	39.6	88.27	198
	DFBS	18.14	939.87	83.73	71.51	198
	AdDel	18.22	599.21	82.92	70.58	198
	Deletion	18.38	626.41	56.48	72.53	198
	Additive	18.02	1337.05	119.55	69.93	198
	None	22.71	327.65	30.85	98.79	198
sp_an	Greedy	19.13	386.86	35.05	71.2	195
	DFBS	17.76	921.14	82.24	67.0	195
	AdDel	17.87	581.02	80.99	67.33	195
	Deletion	17.99	605.18	54.7	69.61	195
	Additive	17.88	1337.76	119.81	67.14	195
	None	20.82	282.62	26.17	78.06	195

The results are only compared for instances that were solved by all of the methods

Table B.5: Disjunctive scheduling: Results for the variants of the Minimising Tardiness problem

problem	cut_str	N_{iter}	N_{sub}	T_{sub}	T_{mas}	N_{inst}
No-Split	Greedy	20.02	444.46	117.18	38.65	178
	DFBS	5.79	111.52	31.34	29.65	178
	AdDel	5.74	149.14	71.39	20.14	178
	Deletion	6.01	392.45	127.21	26.95	178
	Additive	5.48	472.69	62.22	23.12	178
	None	42.41	43.41	7.86	74.47	178
Split	Greedy	1.0	21.54	1.49	4.79	477
	DFBS	1.0	39.74	2.79	4.81	477
	AdDel	1.0	29.16	2.85	4.79	477
	Deletion	1.0	30.57	2.1	4.8	477
	Additive	1.0	52.97	3.66	4.76	477
	None	1.0	17.66	1.23	4.78	477

The results are only compared for instances that were solved by all of the methods

Table B.6: Disjunctive scheduling: Results for the variants of the Finding a feasible schedule problem

cut_str	N_{iter}	N_{sub}	T_{sub}	T_{mas}	N_{inst}
DFBS	62.64	697.14	0.01	2.46	158
AdDel	65.27	1344.53	0.01	2.55	158
Deletion	30.68	1475.92	0.01	1.29	158
Additive	65.73	2676.89	0.01	2.67	158

The results are only compared for instances that were solved by all of the methods

Table B.7: Vehicle routing problem: Results for the Minimising Makespan problem

cut_str	N_{iter}	N_{sub}	T_{sub}	T_{mas}	N_{inst}
DFBS	3.0	117.5	0.06	0.04	16
AdDel	128.5	1266.0	0.07	1.88	16
Deletion	3.0	85.0	0.06	0.04	16
Additive	3.0	300.0	0.06	0.04	16

The results are only compared for instances that were solved by all of the methods

Table B.8: Vehicle routing problem: Results for the Minimising Tardiness problem

cut_str	N_{iter}	N_{sub}	T_{sub}	T_{mas}	N_{inst}
DFBS	165.76	6036.35	0.07	5.03	34
AdDel	197.71	2283.91	0.07	3.61	34
Deletion	27.06	820.79	0.07	0.67	34
Additive	61.71	3823.91	0.07	1.47	34

The results are only compared for instances that were solved or timed out by all of the methods

Table B.9: Vehicle routing problem: Results for the Minimising Tardiness problem

cut_str	N_{iter}	N_{sub}	T_{sub}	T_{mas}	N_{inst}
DFBS	22.36	286.65	0.02	12.19	161
AdDel	20.99	533.29	0.02	9.04	161
Deletion	25.96	887.76	0.02	22.44	161
Additive	26.36	1510.15	0.02	12.93	161

The results are only compared for instances that were solved by all of the methods

Table B.10: Vehicle routing problem: Results for the Minimising Total Travel time problem