Computational evaluation of cut-strengthening techniques in logic-based Benders' decomposition

Aigerim Saken[1], Emil Karlsson[2,3], Stephen J. Maher[1,4], and Elina Rönnberg[2]

[1] Department of Mathematics, University of Exeter, United Kingdom
as1392@exeter.ac.uk
[2] Department of Mathematics, Linköping University, SE-581 83 Linköping, Sweden
elina.ronnberg@liu.se
[3] Saab AB, SE-581 88 Linköping, Sweden
[4] Quantagonia GmbH, Bad Homburg, Germany

Abstract. Cut-strengthening techniques have a significant impact on the computational effectiveness of the Logic-based Benders' decomposition (LBBD) scheme. While there have been numerous cut-strengthening techniques proposed, very little is understood about which techniques achieve the best computational performance for the LBBD scheme. This is typically due to implementations of LBBD being problem specific and thus, no systematic study of cut-strengthening techniques for both feasibility and optimality cuts has been performed. This paper aims to provide guidance for future researchers with the presentation of an extensive computational study of five cut-strengthening techniques that are applied to three different problem types. The computational study involving 3000 problem instances shows that cut-strengthening techniques that generate irreducible cuts outperform the greedy algorithm and the use of no cut strengthening. It is shown that cut strengthening is a necessary part of the LBBD scheme; and depth-first binary search and deletion filter are the most effective cut-strengthening techniques

Keywords: Logic-based Benders' decomposition · Cut strengthening · Feasibility cuts · Optimality cuts · Irreducible cuts · Benders'cuts

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 [11], 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. However, no systematic investigation into the effectiveness of cut-strengthening techniques of 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 [7]. 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 [14]—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 [14] 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 paper aims to act as a guide for future researchers applying LBBD and cut-strengthening techniques. To this end, the main contributions are:

- An indepth 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.
- 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 paper 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 2. Section 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 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 5. The main contributions of this paper are the results from computational experiments. Section 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 7.

2 Literature background

Cut strengthening is one of the most common acceleration techniques for Benders-like algorithms [19]. The benefit of applying cut strengthening in an LBBD scheme has been demonstrated in Karlsson and Rönnberg [14,15], Lam et al. [16], Lindh et al. [18], Hooker [9, 10], Riedler and Raidl [20], Benini et al. [2], and Sadykov [21]. 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 [9], Benini et al. [2], and Coban and Hooker [8]. Hooker [9] 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 Coban and Hooker [8], where the master problem assigns jobs to time segments and the subproblem schedules them. Our paper evaluates the greedy algorithm, solving both problems from Coban and Hooker [8] and Hooker [9] for different objective types. While fast and easy to implement, our computational experiments will show that the greedy approach is often outperformed by other cut-strengthening techniques.

Riedler and Raidl [20] and Lam et al. [16] apply a cut-strengthening algorithm that has the same structure as a deletion filter. Riedler and Raidl [20] 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. [16] 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 paper, where we show that deletion filter is one of the best-performing techniques.

Lindh et al. [18] apply cut strengthening in their LBBD scheme to solve a short-term scheduling problem for a cut-and-fill mine. 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 [15] 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. Our computational evaluation demonstrates that the DFBS algorithm is one of the best-performing cut-strengthening algorithms.

Cambazard et al. [3] use the iterative conflict detection algorithm QuickXplain [12] 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 [21] proposes a branch-and-bound-type algorithm to strengthen no-good cuts. The proposed algorithm is a modified version of the Carlier algorithm [4]. The algorithm is implemented within a hybrid branch-and-check [23] 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. Our study is focused on general cut-strengthening techniques, therefore the modified Carlier algorithm is not within the scope of this paper.

In the conference paper [14], 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. Our paper extends the contribution in Karlsson and Rönnberg [14] by including the strengthening of optimality cuts and considering additional problem formulations.

3 Logic-based Benders' scheme and cut strengthening

The cut-strengthening techniques evaluated in this paper can be applied to a variety of LBBD schemes. To facilitate the general discussion of cut-strengthening techniques we present a generic problem formulation and decision scheme. Fur-

ther, this formulation will be used in the mathematical description of applications, highlighting the general nature of the evaluated approaches.

3.1 Logic-based Benders' decomposition

LBBD can be applied to problems of the form

$$\begin{aligned} [\mathrm{P}] \quad \min \; & f(x,y), \\ \text{s.t.} \; & A(x), \\ & H(y), \\ & C(x,y), \\ & x \in D_x, \\ & y \in D_y. \end{aligned} \tag{1}$$

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 (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 $[\mathrm{SP}(\bar{x})]$, is termed the subproblem, where $\bar{x}$ is a fixed solution for x.

The subproblem takes the form

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

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:

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

where $A \overset{P}{\Longrightarrow} B$ means that proof P deduces B from A, and $\mathcal{P}$ 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 [9]. 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 [9]:

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 (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.

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 (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

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

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 paper, the [Valid inequalities] include relaxations of constraints present in the subproblems, but also auxiliary variables and constraints.

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

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

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 changing an $x_j = 1$ 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.

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:

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

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

$$\sum_{j \in \mathcal{J}(x^k)} (1 - x_j) \geq 1, \tag{2}$$

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

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 $v^*(y)$ be the optimal value of the subproblem. Then, a value cut $f(x, y) \geq B_{x^k}(x)$ should bound $f(x, y)$ for any solution x by the value $B_{x^k}(x^k) = v^*(y)$. The no-good formulation of this cut is given by

$$f(x, y) \geq v^*(y)\Big(1 - \sum_{j \in \mathcal{J}(x^k)} (1 - x_j)\Big). \tag{3}$$

The inequality (3) provides the tightest bound when all the values of x_j for $j \in \mathcal{J}(x^k)$ are equal to 1.

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 $[\mathrm{SP}(x^k)]$ then decouples into a separate problem $[\mathrm{SP}_i(x^k)]$ for each such block i:

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

Solving subproblem i generates cuts that only include variables from $\mathcal{J}_i$. It has been highlighted in Karlsson and Rönnberg [14] and Ciré et al. [7] that separating the subproblem significantly improves the runtime of the LBBD scheme. Not all of the problems considered in this study are separable.

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 $\mathcal{J}(\bar{x})$, denoted by $\mathcal{J}(\hat{x})$, where the corresponding variables induce a subproblem with an optimal objective equal to $\bar{v}$. The set $\mathcal{J}(\hat{x})$ is identified by systematically solving subproblem $[\mathrm{SP}(\bar{x})]$ with trial values corresponding to different subsets of $\mathcal{J}(\bar{x})$. Different trial values lead to different relaxations of subproblem $[\mathrm{SP}(\bar{x})]$. The problem structure described in Section 3.1 allows to obtain a relaxation of the subproblem by changing trial values of decision variables to 0. 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 [14].

Definition 1. *Let $\bar{v}$ be the optimal value of subproblem $[\mathrm{SP}(\bar{x})]$. A subset $\mathcal{J}(\hat{x})$ of $\mathcal{J}(\bar{x})$ 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 $\mathcal{J}(\tilde{x}) \subset \mathcal{J}(\hat{x})$, it holds that $[\mathrm{SP}(\tilde{x})]$ has an optimal value $\tilde{v} < \hat{v}$.*

Note that there can be multiple sets $\mathcal{J}(\hat{x})$ meeting the irreducibility criteria, and the sets often share overlapping subsets $\mathcal{J}(\tilde{x})$. Hence, the number of variables in an irreducible cut does not need to be minimal 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 and Rönnberg [14].

Note that most cut-strengthening algorithms can be tailored to a specific problem by specifying the order in which the variables $(\bar{x}_j)_{j \in \mathcal{J}(\bar{x})}$ get selected to form subsets. To maintain generality in our comparison, we do not exploit this possibility and rely on using a random order.

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 $j \in \mathcal{J}(\bar{x})$ 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 1.

Data: A set $\mathcal{J}(\bar{x})$
Result: A subset $\mathcal{J}(\hat{x})$
let v_k be the optimal value of $[SP(\bar{x})]$, where $v_k = \infty$ if $[SP(\bar{x})]$ is infeasible;
while *True* **do**
  Select an index $j \in \mathcal{J}(\bar{x})$;
  $\bar{x}_j \leftarrow 0$;
  let v'_k be the new optimal value of $[SP(\bar{x})]$;
  if $v'_k \neq v_k$ **then**
    $\bar{x}_j \leftarrow 1$;
    $\mathcal{J}(\hat{x}) = \{j \in \mathcal{J}(\bar{x}) | x_j = 1\}$;
    return $\mathcal{J}(\hat{x})$;
  end
end

Algorithm 1: The greedy cut-strengthening algorithm

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 [5]. There is one iteration for each $j \in \mathcal{J}(\bar{x})$ 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 [5]. A pseudo-code for the deletion filter algorithm that finds an irreducible value cut is given in Algorithm 2.

Data: A set $\mathcal{J}(\bar{x})$
Result: An irreducible subset $\mathcal{J}(\hat{x})$
let v_k be the optimal value of $[SP(\bar{x})]$, where $v_k = \infty$ if $[SP(\bar{x})]$ is infeasible;
for $j \in \mathcal{J}(\bar{x})$ **do**
  $\bar{x}_j \leftarrow 0$;
  let v'_k be the new optimal value of $[SP(\bar{x})]$;
  if $v'_k \neq v_k$ **then**
    $\bar{x}_j \leftarrow 1$;
  end
end
$\mathcal{J}(\hat{x}) = \{j \in \mathcal{J}(\bar{x}) | x_j = 1\}$;
return $\mathcal{J}(\hat{x})$;

Algorithm 2: The deletion filter algorithm

Additive method The additive method was introduced for detecting IIS in linear programming by Tamiz et al. [22]. Chinneck [6] 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 3.

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 $j \in \mathcal{J}(\bar{x})$. 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

Data: A set $\mathcal{J}(\bar{x})$
Result: An irreducible subset $\mathcal{J}(\hat{x})$
1 $S \leftarrow \mathcal{J}(\bar{x}); T \leftarrow \emptyset; I \leftarrow \emptyset$;
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 $T \leftarrow I$;
6 **for** $j \in S$ **do**
7 $\quad T \leftarrow T + j$;
8 $\quad \bar{x}_i \leftarrow 1, \quad i \in T$;
9 $\quad$ let v'_k be the new optimal value of $[SP(\bar{x})]$;
10 $\quad$ **if** $v'_k = v_k$ **then**
11 $\quad\quad I \leftarrow I + j$;
12 $\quad\quad x'_i \leftarrow 1, i \in I$;
13 $\quad\quad$ let v'_k be the optimal value of $[SP(x')]$;
14 $\quad\quad$ **if** $v'_k = v_k$ **then**
15 $\quad\quad\quad \mathcal{J}(\hat{x})$=I;
16 $\quad\quad\quad$ return $\mathcal{J}(\hat{x})$
17 $\quad\quad$ **else**
18 $\quad\quad\quad$ go to Line 3
19 $\quad\quad$ **end**
20 $\quad$ **else**
21 $\quad\quad$ go to Line 5
22 $\quad$ **end**
23 **end**

Algorithm 3: The additive method algorithm

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 4.

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

Algorithm 4: The additive/deletion algorithm

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 $T = \mathcal{J}(\bar{x})$. 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 5 and it is based on the presentation in [1] 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 [13], and is used to strengthen cuts in Cambazard et al. [3].

Example 1. The example presented in Figure 1 illustrates DFBS applied to the set $\mathcal{J} = \{1, 2, 3, 4, 5, 6, 7, 8\}$. Let v be the objective value of $[SP(\mathcal{J})]$. Set T is initially equal to $\mathcal{J}$, 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 $I = \{3, 5\}$.

Data: A set $\mathcal{J}(\bar{x})$
Result: An irreducible subset $\mathcal{J}(\hat{x})$
1 $T \leftarrow \mathcal{J}(\bar{x}); S \leftarrow \emptyset$; $\bar{x}_j \leftarrow 0, \quad j \in \mathcal{J}(\bar{x})$;
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 | | | $\mathcal{J}(\hat{x}) = I$;
10 | | | return $\mathcal{J}(\hat{x})$;
11 | | **end**
12 | | $T \leftarrow S; S \leftarrow \emptyset$;
13 | | **if** $|T| \geq 2$ **then**
14 | | | go to Line 4;
15 | | **end**
16 | | $T_2 \leftarrow T; T_1 \leftarrow \emptyset$
17 | **else**
18 | | Split T into T_1 and T_2;
19 | **end**
20 | $\bar{x}_j \leftarrow 1, \quad j \in S \cup T_1 \cup I$;
21 | let v'_k be the new optimal value of $[SP(\bar{x})]$;
22 | **if** $v'_k = v_k$ **then**
23 | | $T \leftarrow T_1$;
24 | **else**
25 | | $S \leftarrow S + T_1; T \leftarrow T_2$;
26 | **end**
27 | $\bar{x}_j \leftarrow 0, \quad j \in \mathcal{J}(\bar{x})$;
28 **end**

Algorithm 5: The DFBS cut-strengthening algorithm

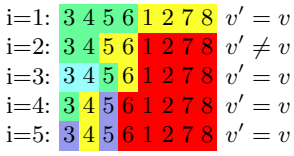


Fig. 1: DFBS example: T_1= , T_2= , S= , I=

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.

4.1 Cumulative facility scheduling with fixed costs

An LBBD scheme for cumulative facility scheduling with fixed costs was introduced in Hooker [9]. The problem is to first allocate a set of jobs $\mathcal{J} = \{1, ..., n\}$ to a set of facilities $\mathcal{F}$ where the jobs are then scheduled for processing. Each job $j \in \mathcal{J}$ is assigned to exactly one facility $f \in \mathcal{F}$. 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 $\mathcal{C}_f$ 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 (1), is given by

$$\min \ h(x) + v(y), \tag{4}$$

$$\text{s.t.} \ \sum_{f \in \mathcal{F}} x_{jf} = 1, \quad j \in \mathcal{J}, \tag{5}$$

$$\textsc{Cumulative}((y_{jf}|j \in \mathcal{J}), (p_{jf}|j \in \mathcal{J}), (c_{jf}|j \in \mathcal{J}), \mathcal{C}_f), \quad f \in \mathcal{F}, \tag{6}$$

$$x_{jf} \rightarrow r_j \leq y_{jf} \leq d_j - p_{jf}, \quad j \in \mathcal{J}, f \in \mathcal{F}, \tag{7}$$

$$x_{jf} \in \{0, 1\}, \quad j \in \mathcal{J}, f \in \mathcal{F}, \tag{8}$$

$$y_{jf} \in [r_j, d_j], \quad j \in \mathcal{J}, f \in \mathcal{F}. \tag{9}$$

Constraints (5), corresponding to $A(x)$ in problem (1), ensure that each job is assigned to exactly one facility. Constraints (6) correspond to $H(y)$ and Constraints (7) correspond to $C(x, y)$. Constraints (6) control the resource consumption and Constraints (7) ensure that the jobs are processed within a specified time window. The domain D_x is given by Constraints (8) and domain D_y is given by Constraints (9).

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

$$\begin{aligned} &\min \ h(x), \\ &\text{s.t.} \ \sum_{f \in \mathcal{F}} x_{jf} = 1, \quad j \in \mathcal{J}, \\ &\quad h(x) \geq B_{x^i}(x), \quad i = 1, ..., k-1, \\ &\quad \text{[Valid inequalities]}, \\ &\quad x_{jf} \in \{0, 1\}, \quad j \in \mathcal{J}, f \in \mathcal{F}. \end{aligned} \tag{10}$$

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

$$\begin{aligned} &\min\ v(y), \\ &\text{s.t. } \textsc{Cumulative}((y_{jf}|j\in\mathcal{J}),(p_{jf}|j\in\mathcal{J}),(c_{jf}|j\in\mathcal{J}),\mathcal{C}_f), \quad f\in\mathcal{F}, \\ &\qquad x_{jf}^k = 1 \rightarrow r_j \le y_{jf} \le d_j - p_{jf}, \quad j\in\mathcal{J}, f\in\mathcal{F}, \end{aligned} \tag{11}$$

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

Different objective functions are considered for the cumulative scheduling problem by defining the two components $h(x)$ and $v(y)$ of the objective function (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.

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

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

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 [9] shows that it is important to include a relaxation of the subproblem (11) in the formulation of the master problem (10). In this case, the [Valid inequalities] in problem (10) contain a relaxation of constraints (6)–(7) as follows. Let $\mathcal{J}(t_1,t_2) = \{j\in\mathcal{J}|t_1 \le r_j, d_j \le t_2\}$ 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 $\mathcal{C}_f$, the total amount of resource available at facility f in the interval (t_1,t_2) is $\mathcal{C}_f(t_2-t_1)$. Therefore, the total resource consumption $\sum_{j\in\mathcal{J}(t_1,t_2)} p_{jf}c_{jf}$ of jobs assigned to facility f cannot exceed $\mathcal{C}_f(t_2-t_1)$. This can be formulated as

$$\frac{1}{\mathcal{C}_f}\sum_{j\in\mathcal{J}(t_1,t_2)} p_{jf}c_{jf}x_{jf} \le t_2 - t_1. \tag{12}$$

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 (11) is feasible, then (x^k,y^k) is the optimal solution to the original problem. Otherwise, let $\mathcal{J}_{kf} = \{j|x_{jf}^k = 1\}$ be the set of jobs assigned to facility f in iteration k and define a Benders' cut in the form of a nogood inequality (2) as

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

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

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

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

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

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

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

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 (2) is generated. Otherwise, if the subproblem is feasible and has an optimal makespan M^*_{kf}, an optimality cut

$$M \geq M^*_{kf}\Big(1 - \sum_{j \in \mathcal{J}_{kf}} (1 - x_{jf})\Big) \tag{13}$$

is generated. The cut (13) provides the tightest bound on M when all of the jobs in $\mathcal{J}_{kf}$ are assigned to facility f.

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

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

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

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

The master problem can be strengthened by [Valid inequalities]. For each job $i \in \mathcal{J}$, let $\mathcal{J}(0, d_i)$ 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 $j \in \mathcal{J}(0, d_i)$ assigned to facility f is bounded below by

$$\max\left\{\frac{1}{\mathcal{C}_f}\sum_{j\in\mathcal{J}(0,d_i)} p_{jf}c_{jf} - d_i, 0\right\}, \quad i \in \mathcal{J}. \tag{14}$$

[Valid inequalities] can be derived from the bound (14). These are given by

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

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

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

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

where T_{fi} is a tardiness of job i on facility f and $\mathcal{U}_{fi}$ 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 $p_{\pi(1)} \leq ... \leq p_{\pi(n)}$, see details in Hooker [9].

Generation of optimality cuts is based on the same principle as the cuts given by inequality (13). Let T^*_{fk} be the minimum tardiness on facility f in iteration k when the jobs in $\mathcal{J}_{fk}$ are assigned to it. Then the Benders' cut in iteration k is given by

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

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

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 $\mathcal{J}$ to run at a single facility. Each job $j \in \mathcal{J}$ 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 and Hooker [8]. The main difference to the formulation in Coban and Hooker [8] 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 $s \in \mathcal{S} = \{1, ..., m\}$. 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 $j \in \mathcal{J}$ 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 (1) as

$$\min \; h(x) + v(y) + 2\sum_{j\in\mathcal{J}} p_j u_j, \tag{15}$$

$$\text{s.t.} \; \sum_{s\in\mathcal{S}} x_{js} + u_j = 1, \quad j \in \mathcal{J}, \tag{16}$$

$$\textsc{Disjunctive}((y_j|j \in \mathcal{J}), (p_j|j \in \mathcal{J})), \tag{17}$$

$$x_{js} \rightarrow r_j \leq y_j \leq d_j - p_j, \quad s \in \mathcal{S}, j \in \mathcal{J}, \tag{18}$$

$$x_{js} \rightarrow a_s \leq y_j \leq a_{s+1} - p_j, \quad s \in \mathcal{S} \setminus m, j \in \mathcal{J}, \tag{19}$$

$$x_{js} \in \{0, 1\}, \quad j \in \mathcal{J}, s \in \mathcal{S}, \tag{20}$$

$$u_j \in \{0, 1\}, \quad j \in \mathcal{J}, \tag{21}$$

$$y_j \in [r_j, d_j], \quad j \in \mathcal{J}, s \in \mathcal{S}. \tag{22}$$

Constraints (16) correspond to $A(x)$ and ensure that all jobs are assigned to no more than one segment. Constraint (17) corresponds to $H(y)$ and ensures that jobs do not overlap and run sequentially. Constraints (18)–(19) correspond to $C(x, y)$. Constraints (18) ensure the time windows of jobs are observed. Constraints (19) ensure jobs are processed within the time segments. The domains D_x and D_y are given by Constraints (20) and (22) respectively.

The master problem in iteration k is given by

$$\begin{aligned}
&\min \; h(x) + 2\sum_{j\in\mathcal{J}} p_j u_j, \\
&\text{s.t.} \; \sum_{s\in\mathcal{S}} x_{js} + u_j = 1, \quad j \in \mathcal{J}, \\
&\quad h(x) \geq B_{x^i}(x), \quad i = 1, ..., k-1, \\
&\quad \text{[Valid inequalities]}, \\
&\quad x_{js} \in \{0, 1\}, \quad j \in \mathcal{J}, s \in \mathcal{S}, \\
&\quad u_j \in \{0, 1\}, \quad j \in \mathcal{J}.
\end{aligned} \tag{23}$$

The master problem is augmented by [Valid inequalities] that contain the relaxation described in Coban and Hooker [8]. 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 $s \in \mathcal{S}$, it is sufficient to enumerate all distinct intervals $[r_j, d_i]$ with $r_j < d_i$ for $i, j \in \mathcal{J}$. 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

$$\sum_{l \in \mathcal{J}(r_j, d_i)} p_l x_{ls} \leq \tilde{d}_{si} - \tilde{r}_{sj}, \quad s \in \mathcal{S}, \quad \forall \text{ distinct } [r_j, d_i], \tag{24}$$

where $\mathcal{J}(r_j, d_i)$ 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:

$$\begin{aligned} \min \ & v(y) \\ \text{s.t.} \ & \textsc{Disjunctive}((y_j | i \in \mathcal{J}), (p_j | i \in \mathcal{J})), \\ & x_{js} \rightarrow r_j \leq y_j \leq d_j - p_j, \quad s \in \mathcal{S}, j \in \mathcal{J}, \\ & x_{js} \rightarrow a_s \leq y_j \leq a_{s+1} - p_j, \quad s \in \mathcal{S} \setminus m, j \in \mathcal{J}. \end{aligned} \tag{25}$$

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

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 $2\sum_{j \in \mathcal{J}} p_j u_j$. Similar to cumulative facility scheduling (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 (15)–(22). Otherwise, feasibility cuts are generated for each segment s. Let $\mathcal{J}_{ks} = \{j | x^k_{js} = 1\}$ be the set of jobs that are assigned to segment s in iteration k. The feasibility cuts are given by

$$\sum_{j \in \mathcal{J}_{ks}} (1 - x_{js}) \geq 1, \quad s \in \mathcal{S}. \tag{26}$$

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

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

and the penalty term $2\sum_{j \in \mathcal{J}} p_j u_j$. 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] (24), the master problem for the minimising makespan can be strengthened by the following bound for each distinct r_j and each segment s

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

The subproblem can be separated into an optimisation problem for each segment s. If the subproblem is infeasible, feasibility cuts in the form of (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

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

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

Minimising total tardiness The objective function to minimise tardiness is given by the components

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

and the penalty term $2\sum_{j \in \mathcal{J}} p_j u_j$. 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 $T = \sum_{s \in \mathcal{S}} T_s$ is minimised in the master problem.

Since the deadlines are now due dates, the time window constraints (18) are modified to

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

The [Valid inequalities] in the master problem (23) contain a modified version of the relaxation (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 $\mathcal{J}(r_j, \infty)$ 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 $j \in \mathcal{J}(r_j, \infty)$ that are assigned to segment s to not exceed $a_{s+1} - \tilde{r}_{sj}$. The inequalities are given by

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

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

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

Since the subproblem is an optimisation problem, either a feasibility cut in the form of (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

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

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 and Van Hentenryck [17] and a LBBD scheme for solving it was derived in Lam et al. [16].

Let R be the set of requests for goods and let $\mathcal{L}$ be the set of locations. Each request $i \in R$ is to be delivered to one location $l_i \in \mathcal{L}$ 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 $l \in \mathcal{L}$. 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 $G = (\mathcal{N}, \mathcal{A})$ is defined to model the routing component of the problem. The set of nodes $\mathcal{N} = R \cup \{O^-, O^+\}$ 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 $\mathcal{A} = \{(i, j) \in \mathcal{N} \times \mathcal{N} | i \neq j\}$ 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 $i \in \mathcal{N}$. 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

$$\min \ h(x) + v(y), \tag{27}$$

$$\text{s.t.} \sum_{i:(i,j)\in\mathcal{A}} x_{ij} = 1, \quad j \in R, \tag{28}$$

$$\sum_{j:(i,j)\in\mathcal{A}} x_{ij} = 1, \quad i \in R, \tag{29}$$

$$\textsc{Cumulative}((y_i^{\text{start}} | i \in R_l), (p_i | i \in R_l), (1 | i \in R_l), C_l), \quad l \in \mathcal{L}, \tag{30}$$

$$x_{ij} \rightarrow y_i^{\text{weight}} + q_j \leq y_j^{\text{weight}}, \quad (i,j) \in \mathcal{A}, \tag{31}$$

$$x_{ij} \rightarrow y_i^{\text{start}} + p_i + c_{ij} \leq y_j^{\text{start}}, \quad (i,j) \in \mathcal{A}, \tag{32}$$

$$x_{ij} \in \{0,1\}, \quad (i,j) \in \mathcal{A}, \tag{33}$$

$$y_i^{\text{start}} \in [r_i, d_i], \quad i \in \mathcal{N}, \tag{34}$$

$$y_i^{\text{weight}} \in [q_i, Q], \quad i \in \mathcal{N}. \tag{35}$$

Constraints (28)–(29), which correspond to $A(x)$ in problem (1), ensure that each request is assigned to exactly one vehicle. The Cumulative Constraints (30) correspond to $H(y)$ and enforce capacity limit at each location. Constraints (31)–(32) correspond to $C(x,y)$ in problem (1). The vehicle weight limits are enforced by constraints (31). Constraints (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

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

The [Valid inequalities] comprise constraints (31)–(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

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

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

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

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

$$\sum_{(i,j)\in\mathcal{J}_k} (1 - x_{ij}) \geq 1. \tag{36}$$

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 (36) is generated. If the subproblem has an optimal solution with objective value M_k^* in iteration k, an optimality cut similar to inequality (13) is generated. This cut is given by

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

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

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

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 (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

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

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

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), where a tight bound is only given when each variable in $\mathcal{J}$ 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), or as an alternative. The analytic cuts can also be strengthened using the cut-strengthening techniques described in Section 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 [9].

Cumulative scheduling Let $\mathcal{J}_{fk}$ 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

$$M_f \geq M_{fk}^* - \sum_{j \in \mathcal{J}_{fk}} p_{jf}(1 - x_{jf}) - \max_{j \in \mathcal{J}_{fk}} \{d_j\} + \min_{j \in \mathcal{J}_{fk}} \{d_j\}. \tag{37}$$

The bound complies with the properties of the bounding function outlined in Section 3.1. When the supbroblem has an optimal solution with the minimum

makespan M^*_{fk}, an analytic Benders' cut of form (37) can be generated instead of, or alongside an optimality cut of form (13).

Analytic Bender's cuts for the minimum tardiness problem are derived based on the same principle as the analytic cuts (37) for the minimising makespan problem. Let T^*_{fk} be the minimum tardiness corresponding to $\mathcal{J}_{fk}$. An analytic cut that bounds the total tardiness T over all facilities is given by

$$T \geq \sum_{f\in\mathcal{F}} \left(T^*_{fk} - \sum_{i\in\mathcal{J}_{fk}} \max\Big\{ \sum_{j\in\mathcal{J}_{fk}} p_{jf} - d_i, 0 \Big\}(1 - x_{if}) \right). \tag{38}$$

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

$$\begin{aligned} M_s \geq M^*_{sk} - \sum_{j\in\tilde{\mathcal{J}}_{sk}} p_{js}(1 - x_{js}) - \max_{j\in\tilde{\mathcal{J}}_{sk}} \{d_j\} \\ + \min_{j\in\tilde{\mathcal{J}}_{sk}} \{d_j\} - M^*_{sk} \sum_{j\in\mathcal{J}_{sk}\setminus\tilde{\mathcal{J}}_{sk}} (1 - x_{js}). \end{aligned} \tag{39}$$

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 and Hooker [8].

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

$$r^{\max}_s + \sum_{j\in\mathcal{J}_s} p_{js} \leq a_{s+1} \tag{40}$$

holds, the problem is feasible. Based on inequality (40), the analytic Benders' cut for minimising tardiness on segment s is

$$T_s \geq \begin{cases} \left(T^*_{sk} - \sum_{i\in\mathcal{J}_{sk}} \left(r^{\max}_s + \sum_{j\in\mathcal{J}_{sk}} p_{js} - d_i \right)^+ (1 - x_{js}) \right), \text{if (40) holds} \\ \left(T^*_{sk} - \left(1 - \sum_{i\in\mathcal{J}_{sk}} (1 - x_{js}) \right) \right), \quad \text{otherwise,} \end{cases}$$

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

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

6 Computational evaluation

The effectiveness of the cut-strengthening techniques, described in Section 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 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 4.2 and vehicle routing 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 [14] 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.

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[5]. For the cumulative facility scheduling problem with fixed costs, referred to as Problem 4.1, we use 336 instances from Hooker [9]. For the single machine scheduling problem with sequence-dependent setup times and multiple time windows, referred to as Problem 4.2, we use instances generated based on the description in Coban and Hooker [8]. For the vehicle routing problem with location congestion, referred to as Problem 4.3, we use 450 instances from Lam et al. [16].

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

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

size of the cuts generated by the cut-strengthening techniques. Additional data is provided in Tables in Appendix B.

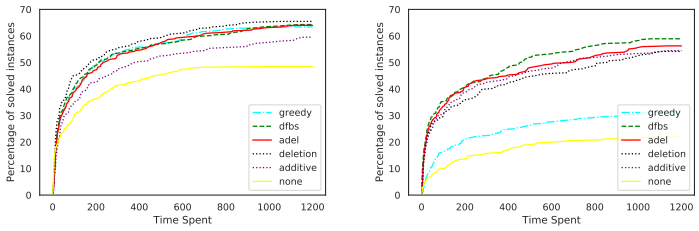
For the run time plots, the horizontal axis gives the time and the vertical axis gives the percentage of solved instances. A point (x, y) on the curve means that y% of instances can be solved in less than x seconds. The cut size figures are histograms, a point (x, y) means average cut size x has frequency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

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.

Minimising makespan for cumulative facility scheduling problem It can be seen in the Figure 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 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 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 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.

Minimising tardiness for cumulative facility scheduling problem The first thing to notice in the run time Figure 5 is that the split subproblem gives significantly higher results than the no-split subproblem. The best effectiveness



(a) Split (b) No-split

Fig. 2: Cumulative scheduling: Percentage of solved instances for minimising makespan problem for with strengthened optimality cuts

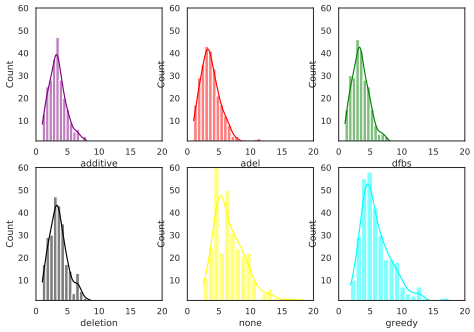


Fig. 3: Cumulative scheduling: Average size of optimality cuts for the minimising makespan problem with split subproblem

for the no-split subproblem is shown in Figure 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 solved instances is notably behind other algorithms. The results in Figure 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

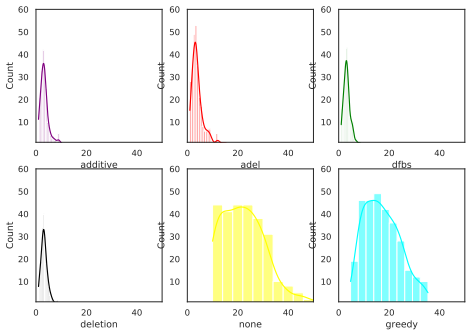
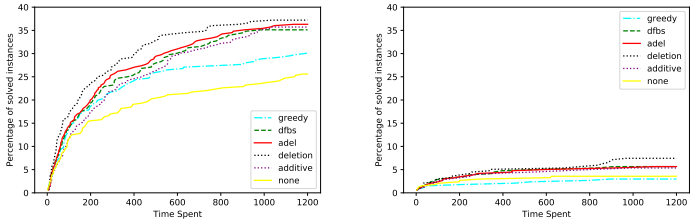


Fig. 4: Cumulative scheduling: Average size of optimality cuts for the minimising makespan problem for with no-split subproblem

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 [9].



(a) Split (b) No-split

Fig. 5: Cumulative scheduling: Percentage of solved instances for minimising tardiness problem with strengthened optimality cuts

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 in run time Figure 8. At the same time, applying cut-strengthening to the split subproblem still gives a significant boost to the results: the deletion filter, DFBS,

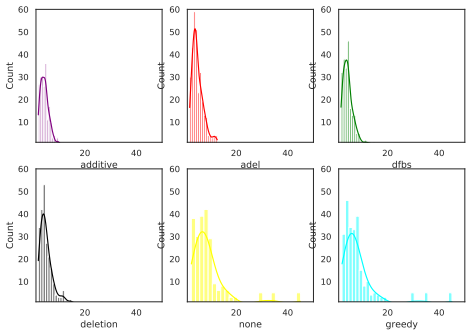


Fig. 6: Cumulative scheduling: Average size of optimality cuts for the minimising tardiness problem with split subproblem

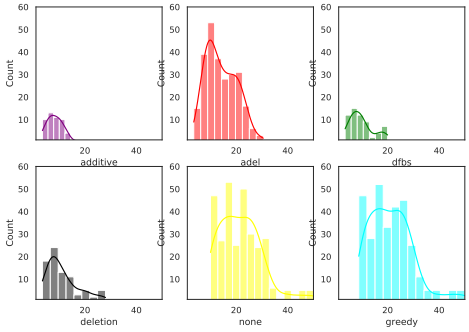
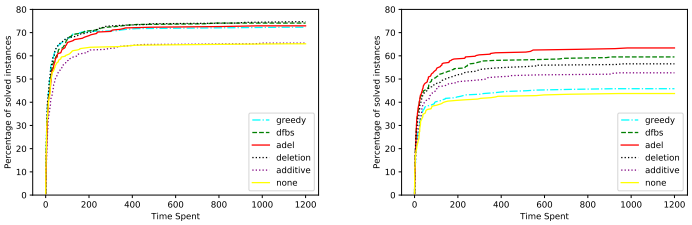


Fig. 7: Cumulative scheduling: Average size of optimality cuts for the minimising tardiness problem with no-split subproblem

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 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,



(a) Split (b) No-split

Fig. 8: Cumulative scheduling: Percentage of solved instances for minimising total cost problem with strengthened feasibility cuts

the difference in effectiveness 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 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.

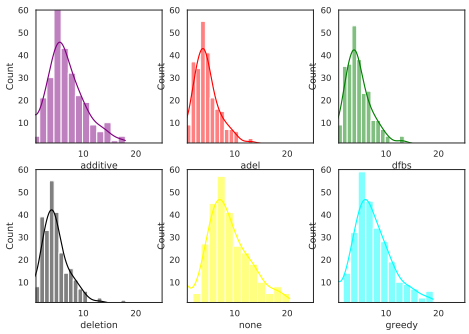


Fig. 9: Cumulative scheduling problem: Average size of feasibility cuts for the minimising total cost with split subproblem

Minimising makespan for disjunctive scheduling problem The run time results in Figure 11a show that using cut-strengthening techniques does not

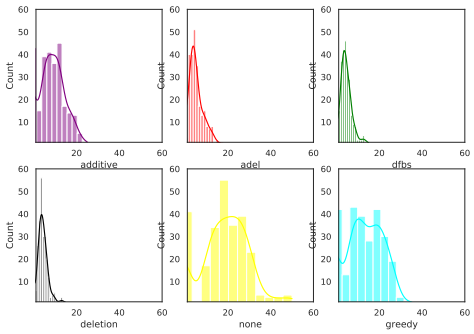
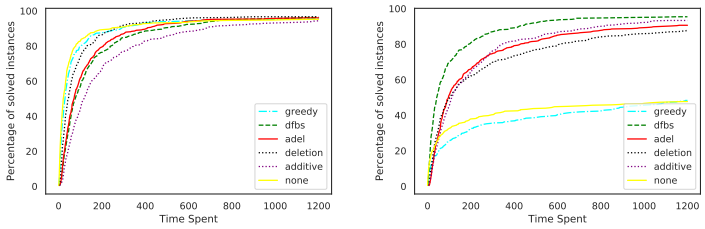


Fig. 10: Cumulative scheduling: Average size of Feasibility cuts for the Minimising Total Cost problem with No-Split Subproblem

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 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 11a.



(a) Split (b) No-split

Fig. 11: Disjunctive scheduling: Percentage of solved instances for minimising makespan with strengthened optimality cuts

By contrast, Figure 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 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

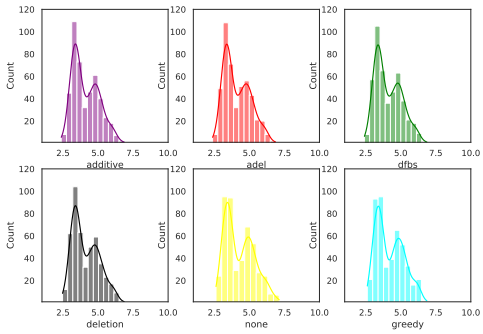


Fig. 12: Disjunctive scheduling: Average size of optimality cuts for the minimising makespan problem with split subproblem

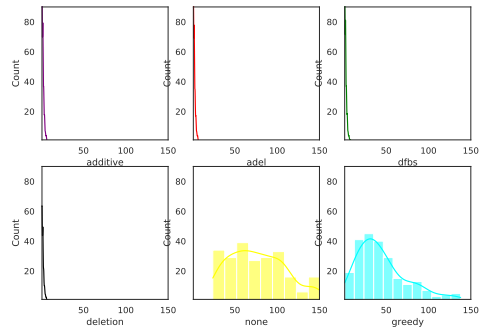
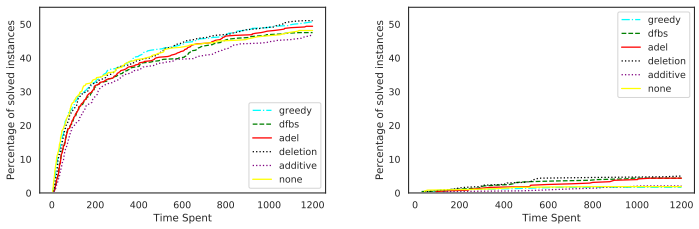


Fig. 13: Disjunctive scheduling: Average size of optimality cuts for the minimising makespan problem with no-split subproblem

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.

Minimising tardiness for disjunctive scheduling problem The most noticeable observation from the run time Figure 14 is that cut strengthening is not an effective way to accelerate the solution. Only splitting the subproblem 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 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 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.



(a) Split (b) No-split

Fig. 14: Disjunctive scheduling: Percentage of solved instances for minimising tardiness with strengthened optimality cuts

Finding a feasible schedule for disjunctive scheduling problem As can be seen from the comparison of the run time in Figure 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

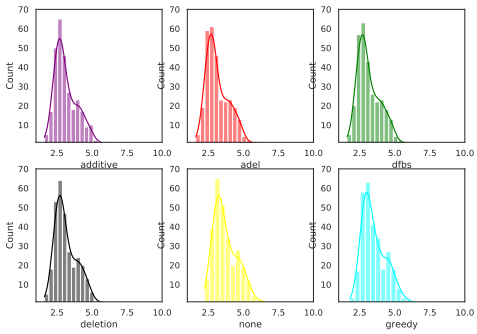


Fig. 15: Disjunctive scheduling: Average size of optimality cuts for the minimising tardiness problem with split subproblem

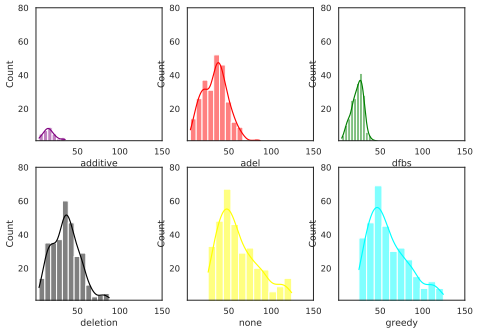
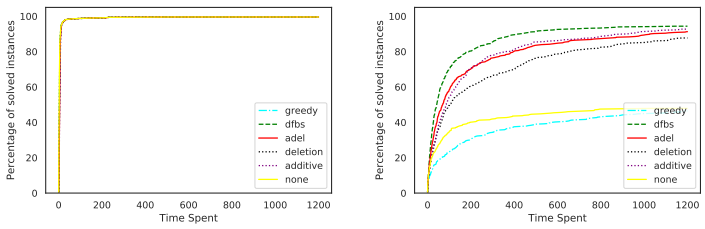


Fig. 16: Disjunctive scheduling: Average size of optimality cuts for the minimising tardiness problem with no-split subproblem

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.



(a) Split (b) No-split

Fig. 17: Disjunctive scheduling: Percentage of solved instances for finding a feasible schedule problem with strengthened feasibility cuts

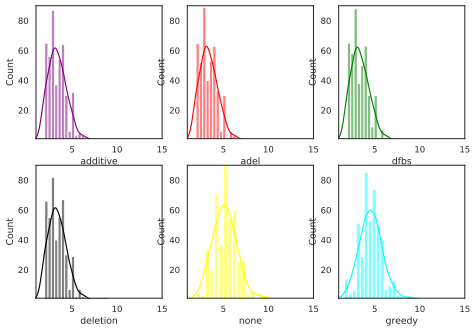


Fig. 18: Disjunctive scheduling: Average size of feasibility cuts for the finding a feasible schedule problem with split subproblem

Minimising makespan for vehicle routing problem The first thing to notice in Figure 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 21 shows that the average size of the generated cuts is similar for the

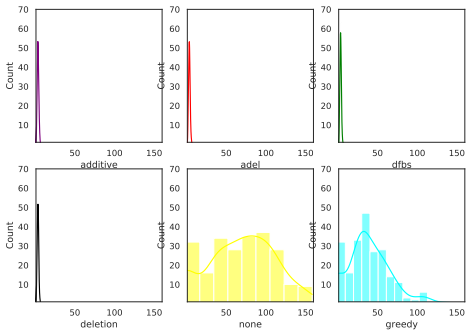
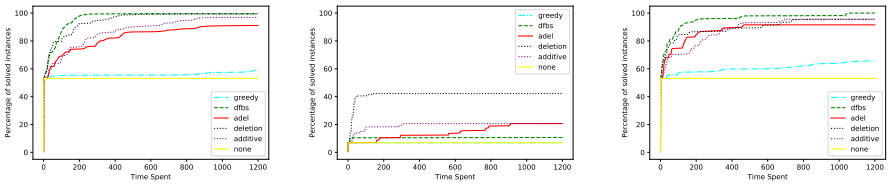


Fig. 19: Disjunctive scheduling: Average size of feasibility cuts for the finding a feasible schedule problem with no-split subproblem

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.



(a) Makespan (b) Tardiness (c) Total Travel Time

Fig. 20: Vehicle routing problem: Percentage of solved instances for minimising makespan, minimising tardiness, and minimising total travel time problems

Minimising total tardiness for vehicle routing problem As can be seen in Figure 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 dead-

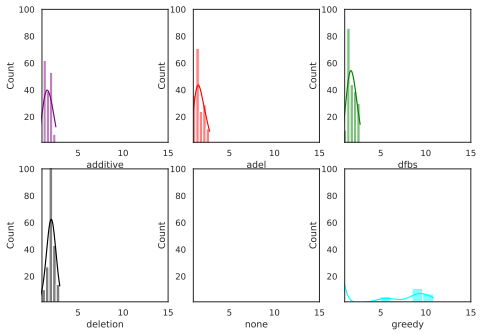


Fig. 21: VRPLC: Average size of feasibility cuts for the minimising makespan problem

lines. 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 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.

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 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

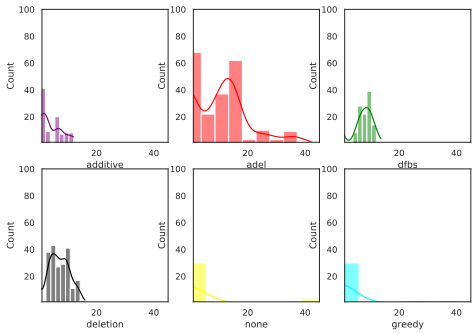


Fig. 22: VRPLC: Average size of optimality cuts for the minimising tardiness problem

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.

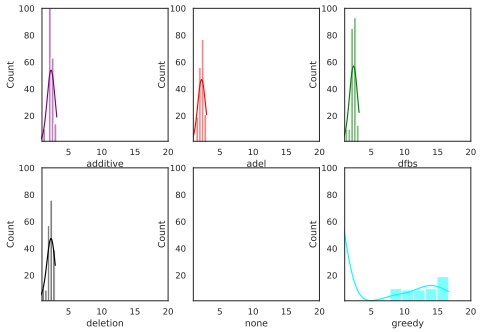


Fig. 23: VRPLC: Average size of feasibility cuts for the minimising total travel time problem

7 Concluding remarks

This paper 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 [15] 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.

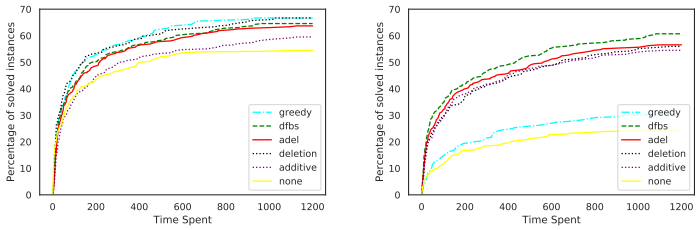
Another observation is that the separation of the subproblem does not always strongly dominate cut strengthening in terms of benefiting the solution process. In fact, there are cases when splitting the subproblem and applying cut strengthening are interchangeable. Therefore some of the techniques can 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.

A Analytic cuts

A.1 Minimising makespan for cumulative facility scheduling problem

As can be seen in Figure 24a 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 24b 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.



(a) Split (b) No-split

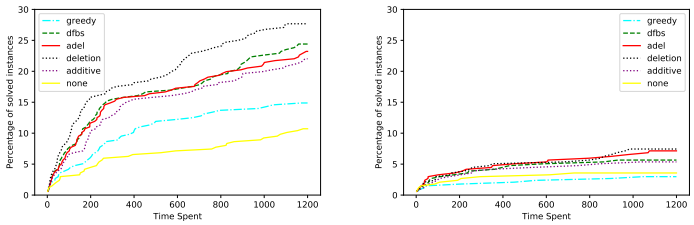
Fig. 24: Cumulative scheduling:Percentage of solved instances for minimising makespan problem with analytic optimality cuts

A.2 Minimising tardiness for cumulative facility scheduling problem

It can be seen in Figure 25b 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.

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

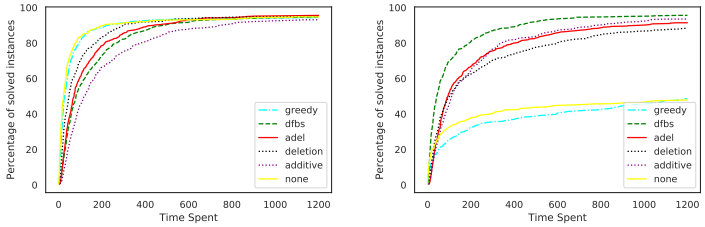


(a) Split (b) No-split

Fig. 25: Cumulative scheduling: Percentage of solved instances for minimising tardiness problem with analytic optimality cuts

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 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.



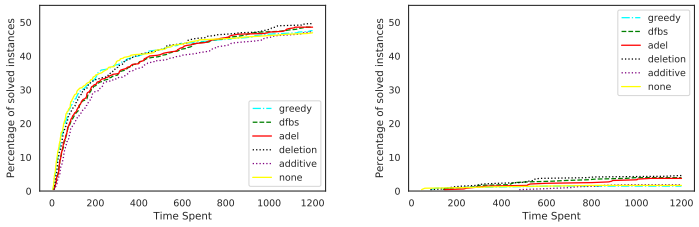
(a) Split (b) No-split

Fig. 26: 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.



(a) Split (b) No-split

Fig. 27: Disjunctive scheduling: Percentage of solved instances for minimising tardiness with analytic cuts

B Tables with additional data

Table 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	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 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_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 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-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 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	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 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_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 6: Disjunctive scheduling: Results for the variants of the Finding a feasible schedule 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 7: Vehicle routing problem: Results for the Minimising Makespan 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 8: Vehicle routing problem: Results for the Minimising Tardiness 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 9: 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 10: Vehicle routing problem: Results for the Minimising Total Travel time 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

References

1. Atlihan, M.K., Schrage, L.: Generalized filtering algorithms for infeasibility analysis. Computers & Operations Research **35**, 1446–1464 (2008), https://doi.org/10.1016/j.cor.2006.08.005
2. Benini, L., Lombardi, M., Mantovani, M., Milano, M., Ruggiero, M.: Multi-stage Benders decomposition for optimizing multicore architectures. In: Perron, L., Trick, M. (eds.) CPAIOR 2008: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 36–50. Springer-Verlag Berlin Heidelberg (2008), https://doi.org/10.1007/978-3-540-68155-7_6
3. Cambazard, H., Hladik, P.E., Déplance, A.M., Jussien, N., Trinquet, Y.: Decomposition and learning for a hard real time task allocation problem. In: Wallace, M. (ed.) Principles and Practice of Constraint Programming – CP 2004. pp. 153–167. Springer-Verlag Berlin Heidelberg (2004), https://doi.org/10.1007/978-3-540-30201-8_14
4. Carlier, J.: The one-machine sequencing problem. European Journal of Operational Research **11**(1), 42–47 (1982). https://doi.org/https://doi.org/10.1016/S0377-2217(82)80007-6, https://www.sciencedirect.com/science/article/pii/S0377221782800076, third EURO IV Special Issue
5. Chinneck, J.W., Dravnieks, E.W.: Locating minimal infeasible constraint sets in linear programs. ORSA Journal of Computing **3**, 157–168 (1991), https://doi.org/10.1287/ijoc.3.2.157
6. Chinneck, J.W.: Feasibility and viability. In: Advances in Sensitivity Analysis and Parametic Programming, pp. 491–531. Springer (1997)
7. Ciré, A., Coban, E., Hooker, J.: Logic-based benders decomposition for planning and scheduling: a computational analysis. The Knowledge Engineering Review **31**, 1–12 (12 2016). https://doi.org/10.1017/S0269888916000254
8. Coban, E., Hooker, J.N.: Single-facility scheduling by logic-based Benders decomposition. Annals of Operations research **210**, 245–272 (2013), https://doi.org/10.1007/s10479-011-1031-z
9. Hooker, J.N.: Planning and scheduling by logic-based Benders decomposition. Operations research **55**, 588–602 (2007), https://doi.org/10.1287/opre.1060.0371
10. Hooker, J.N.: Logic-based Benders decomposition for large-scale optimization. In: Velásquez-Bermúdez, J.M., Khakifirooz, M., Fathi, M. (eds.) Large Scale Optimization in Supply Chains and Smart Manufacturing: Theory and Applications, pp. 1–26. Springer International Publishing (2019), https://doi.org/10.1007/978-3-030-22788-3_1
11. Hooker, J.N., Ottosson, G.: Logic-based Benders decomposition. Mathematical Programming **96**, 33–60 (2003), https://doi.org/10.1007/s10107-003-0375-9
12. Junker, U.: QuickXPlain: Conflict detection for arbitrary constraint propagation algorithms. In: IJCAI01 Workshop on Modeling and Solving Problems with Constraints (CONS-1) (2001)
13. Junker, U.: QuickXPlain: Preferred explanations and relaxations for over-constrained problems. In: In Proceedings of AAAI'04. pp. 167–172 (2004)
14. Karlsson, E., Rönnberg, E.: Strengthening of feasibility cuts in logic-based benders decomposition. In: Stuckey, P.J. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 45–61. Springer International Publishing, Cham (2021)
15. Karlsson, E., Rönnberg, E.: Logic-based benders decomposition with a partial assignment acceleration technique for avionics scheduling. Computers and Operations Research **146**, 105916 (2022).

https://doi.org/https://doi.org/10.1016/j.cor.2022.105916, https://www.sciencedirect.com/science/article/pii/S0305054822001769
16. Lam, E., Gange, G., Stuckey, P.J., Van Hentenryck, P., Dekker, J.J.: Nutmeg: a MIP and CP hybrid solver using branch-and-check. SN Operations Research Forum **1**, 22:1–22:27 (2020), https://doi.org/10.1007/s43069-020-00023-2
17. Lam, E., Van Hentenryck, P.: A branch-and-price-and-check model for the vehicle routing problem with location congestion. Constraints **21**, 394–412 (2016), https://doi.org/10.1007/s10601-016-9241-2
18. Lindh, E., Olsson, K., Rönnberg, E.: Scheduling of an underground mine by combining logic-based benders decomposition and a priority-based heuristic. In: International Conference on the Practice and Theory of Automated Timetabling-PATAT 2022 (2022)
19. Rahmaniani, R., Crainic, T.G., Gendreau, M., Rei, W.: The Benders decomposition algorithm: A literature review. European Journal of Operational Research **259**, 801–817 (2017), https://doi.org/10.1016/j.ejor.2016.12.005
20. Riedler, M., Raidl, G.: Solving a selective dial-a-ride problem with logic-based benders decomposition. Computers & Operations Research **96**, 30–54 (2018)
21. Sadykov, R.: 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 (2008). https://doi.org/https://doi.org/10.1016/j.ejor.2006.06.078, https://www.sciencedirect.com/science/article/pii/S0377221707005991
22. Tamiz, M., Mardle, S., Jones, D.: Detecting iis in infeasible linear programmes using techniques from goal programming. Computers and Operations Research **23**(2), 113–119 (1996). https://doi.org/https://doi.org/10.1016/0305-0548(95)00018-H, https://www.sciencedirect.com/science/article/pii/030505489500018H
23. Thorsteinsson, E.S.: Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In: International conference on principles and practice of constraint programming. pp. 16–30. Springer (2001)