Using maximal feasible subset of constraints to accelerate a logic-based Benders decomposition scheme for a multiprocessor scheduling problem

Department of Mathematics, Linköping University

Alexander Grgic and Filip Andersson

LiTH-MAT-EX–2022/06–SE

Credits: 30 hp
Level: A
Supervisors: Elina Rönnberg,
Department of Mathematics, Linköping University

Emil Karlsson,
Saab AB

Examiner: Nils-Hassan Quttineh,
Department of Mathematics, Linköping University

Linköping: June 2022
Abstract

Logic-based Benders decomposition (LBBD) is a strategy for solving discrete optimisation problems. In LBBD, the optimisation problem is divided into a master problem and a subproblem and each part is solved separately. LBBD methods that combine mixed-integer programming and constraint programming have been successfully applied to solve large-scale scheduling and resource allocation problems. Such combinations typically solve an assignment-type master problem and a scheduling-type subproblem. However, a challenge with LBBD methods that have feasibility subproblems are that they do not provide a feasible solution until an optimal solution is found.

In this thesis, we show that feasible solutions can be obtained by finding and combining feasible parts of an infeasible master problem assignment. We use these insights to develop an acceleration technique for LBBD that solves a series of subproblems, according to algorithms for constructing a maximal feasible subset of constraints (MaFS). Using a multiprocessor scheduling problem as a benchmark, we study the computational impact from using this technique. We evaluate three variants of LBBD schemes. The first uses MaFS, the second uses irreducible subset of constraints (IIS) and the third combines MaFS with IIS.

Computational tests were performed on an instance set of multiprocessor scheduling problems. In total, 83 instances were tested, and their number of tasks varied between 2794 and 10,661. The results showed that when applying our acceleration technique in the decomposition scheme, the pessimistic bounds were strong, but the convergence was slow. The decomposition scheme combining our acceleration technique with the acceleration technique using IIS showed potential to accelerate the method.

Keywords:
logic-based Benders decomposition, maximal feasible subset of constraints, irreducible infeasible subset of constraints, multiprocessor scheduling

URL for electronic version:
http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-186255
Acknowledgements

First, we would like to thank Saab AB for giving us the opportunity to carry out our master’s thesis with them. It has been a fun and challenging experience.

We would also like to thank our industrial supervisor Emil Karlsson and our academic supervisor Elina Rönnberg for committing their time and effort into the project and for their invaluable help. We also want to direct a thank you to Nils-Hassan Quttineh for taking on the responsibility as examiner.

Lastly, we want to thank our family and friends for their support over the past five years during our time as students at Linköping University.

The computations were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC), partially funded by the Swedish Research Council through grant agreement no. 2018-05973.
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LBBD</td>
<td>Logic-based Benders decomposition</td>
</tr>
<tr>
<td>MaFS</td>
<td>Maximal feasible subset of constraints</td>
</tr>
<tr>
<td>MuFS</td>
<td>Maximum feasible subset of constraints</td>
</tr>
<tr>
<td>IIS</td>
<td>Irreducible infeasible subset of constraints</td>
</tr>
<tr>
<td>IFC</td>
<td>Irreducible feasibility cut</td>
</tr>
<tr>
<td>MFS</td>
<td>Maximal feasible solution</td>
</tr>
<tr>
<td>MIP</td>
<td>Mixed-integer programming</td>
</tr>
<tr>
<td>CP</td>
<td>Constraint programming</td>
</tr>
<tr>
<td>DFBS</td>
<td>Depth-first binary search</td>
</tr>
<tr>
<td>NSC</td>
<td>National Supercomputer Centre</td>
</tr>
</tbody>
</table>
# Contents

1 Introduction 1
   1.1 Contributions ................................. 1
   1.2 Scope ........................................ 2
   1.3 Outline ...................................... 2

2 Theory 3
   2.1 Logic-based Benders decomposition ............... 3
   2.2 Maximum feasible subset of constraints .......... 6
   2.3 Irreducible infeasible subset of constraints .... 9
   2.4 Strengthened feasibility cuts in LBBD using IISs .. 12

3 Acceleration techniques for logic-based Benders decomposition 17
   3.1 Feasible solutions in LBBD ...................... 17
   3.2 Maximal feasible solutions in LBBD .............. 19
   3.3 LBBD using MaFS ................................ 19
   3.4 Depth-first binary search algorithm designed for finding MFSs in LBBD ........................................ 21
   3.5 MFSs and IFCs in LBBD ............................. 21
   3.6 Enhanced use of IISs in LBBD ..................... 24

4 Problem formulation 27

5 Implementation of a logic-based Benders decomposition 31
   5.1 Mathematical model ............................... 31
   5.2 Decomposition .................................. 32

6 Results 35
   6.1 Computational environment ....................... 35
   6.2 Test setup ...................................... 36
   6.3 Solving the instances .............................. 39
Chapter 1

Introduction

Solving large-scale scheduling problems efficiently has for a long time been interesting both from a scientific and a practical standpoint. Practical applications within this domain can include many things, such as scheduling staff work hours, warehouse logistics and computer scheduling. Logic-based Benders decomposition (LBBD) [8, 9], derived from classical Benders decomposition [1, 5], has been successfully applied to solve these types of problems. As for any decomposition method, the LBBD divides the optimisation problem into smaller parts, typically a master problem and a subproblem, that are solved separately. LBBD has performed especially well for problems where the master problem is of assignment-type and solved by mixed-integer programming (MIP), and the subproblem is of sequencing-type and solved by constraint programming (CP) [6]. This structure often occurs in resource and scheduling problems.

For an LBBD method to perform well, some form of acceleration technique is required. Using irreducible infeasible subset of constraints (IIS) to strengthen feasibility cuts has proven to be important to obtain good performance [10]. A challenge with LBBDs with feasibility-type subproblems is that, when using a standard implementation, no feasible solutions are obtained until an optimal solution is found. This study is focused on accelerating a hybrid MIP-CP LBBD by constructing feasible solutions in each iteration.

1.1 Contributions

In this master thesis, we show that feasible solutions to a hybrid MIP-CP LBBD problem can be constructed by combining feasible parts of an infeasible master problem assignment. Thus, the subproblem is feasible for the combination of
these parts. From this, we develop an acceleration technique for LBBD that constructs feasible solutions in each iteration by finding such combined parts for which the subproblem is feasible. We use the constructed solutions to calculate pessimistic bounds on the objective function. To obtain strong pessimistic bounds, our technique solves a series of subproblems according to algorithms for constructing a maximal feasible subset of constraints (MaFS) [4, 14].

We perform computational evaluations on a multiprocessor scheduling problem, originating from problems from avionics scheduling [2, 12, 13]. The thesis was performed at and together with Saab AB. The evaluation contains a comparison between LBBD schemes using (i) our acceleration technique, (ii) an IIS cut strengthening acceleration technique [10], and (iii) a combination of the two techniques. The dataset\(^1\) is taken from [11] and includes 384 problem instances.

1.2 Scope

In this master thesis, only the specific MIP-CP hybrid LBBD method mentioned above is addressed. We also limit our computational tests to a dataset of multiprocessor scheduling problem instances with a specific structure, including multiple time windows, exact time lags and positive time lags. Only the computational performance of depth-first binary search (DFBS) algorithms for finding MaFSs and IISs is within the scope of this study.

1.3 Outline

Chapter 2 gives a background regarding LBBD, MaFS and IIS. Chapter 3 presents the main outcomes of the study including an LBBD scheme that uses the concept of MaFS as an acceleration method. Chapter 4 describes the multiprocessor scheduling problem that the computational evaluations are performed on. Chapter 5 presents the LBBD for the problem introduced in Chapter 4. Chapter 6 presents computational results from the study. Chapter 7 includes final comments on the computational results.

\(^1\)https://gitlab.liu.se/eliro15/multiprocessor_scheduling_inst
Chapter 2

Theory

This chapter introduces the concepts and theory necessary to understand the optimisation methods and the computational results in this report. The chapter is divided into three different sections introducing the topics logic-based Benders decomposition, maximum feasible subset of constraints and irreducible infeasible subset of constraints, respectively. To further broaden the understanding of the concepts, a few simple examples are presented. Lastly, irreducible infeasible subset of constraints is presented in context of logic-based Benders decomposition.

2.1 Logic-based Benders decomposition

Logic-based Benders decomposition (LBB) [8, 9] is a row generation-based decomposition method for optimisation problems developed from classical Benders decomposition (BD) [1, 5]. It is a generalisation of the classical technique in the sense that the obtained subproblem can be any type of optimisation problem and it is hence not restricted to being a linear or non-linear programming problem. In theoretical terms this is achieved by instead of solving the LP dual for the subproblem to generate cuts for the master problem, an inference dual [7] is solved. The underlying theory regarding the details of the inference dual is not addressed in this report. Instead, we present LBB from an implementation point of view.

Consider a general optimisation problem [P] with variables \(x\) and \(y\) and constraints \(C(x), C(x, y)\). Then the general idea behind LBB and BD is to first relax the given problem [P] by removing the constraints \(C(x, y)\) and the variables \(y\). This problem is then called the master problem [MP] and it is
solved for the remaining variables $x$. Assigning the variables $x$ in $[P]$ the values in the solution to $[MP]$ yields the so called subproblem $[SP]$ which is solved for the remaining variables $y$. The subproblem then sends feedback to the master problem.

The process of iteratively solving $[MP]$ and $[SP]$ is performed until optimality or feasibility (depending on the problem) is reached or until infeasibility is proven. Convergence for the method is achieved by in each iteration adding a constraint to the master problem which decreases the number of solutions. These constraints are called Benders cuts. The constraint generation is constructed so that the master problem will converge to an equivalent representation of $[P]$.

LBBD can be applied on a variety of different problem structures. In this study, the LBBD will be applied to a problem of the structure

$$
[P] \quad \min \quad f(x) \\
\text{s.t.} \quad C(x, y), \\
\quad C(x), \\
\quad x_i \in \{0, 1\}, i \in I \\
\quad y \in D_y,$$

where the binary variables $x = (x_i)_{i \in I}$ are said to be the master problem variables. Consequently, the variables $y \in D_y$ are the subproblem variables. Further, $C(x, y)$ and $C(x)$ are sets of constraints depending on both $x$ and $y$, and only on $x$, respectively. Note that in this formulation, the objective value depends only on the variables $x$. The constraints in $C(x, y)$ relate the master problem variables to the subproblem variables. We assume that $C(x, y)$ has a structure such that only the assignment $x_i = 1$ restricts the values $y$ can take. The sets of constraints can consist of a variety of different constraint types, e.g., $a^T x \leq b$, 	extsc{Disjunctive} and 	extsc{AllDifferent}. The domain $D_{y_i}$ can, for each $y_i$, be a different type of set, e.g., $\{0, 1\}$, $\{0, 1, 2, 3, \ldots\}$ and $\{-10, -2, 0, 3, 41\}$. The decomposition yields, in iteration $k$, a master problem on the form

$$
[MP^k] \quad \min \quad f(x) \\
\text{s.t.} \quad C(x), \\
\quad B^k(x), \\
\quad x_i \in \{0, 1\}, i \in I
$$

where $B^k(x)$ is the set of Benders cuts generated in previous iterations. At the beginning, $k=1$, the set of Benders cuts is empty, i.e., $B^1(x) = \emptyset$. We denote a solution to the problem $[MP^k]$ as $\bar{x}^k$. 
2.1. Logic-based Benders decomposition

By fixing $x$ to a master problem solution $\bar{x}^k$ in $[P]$, the obtained subproblem is of feasibility-type. This yields a subproblem, which is a constraint set, on the form

$$[SP(\bar{x}^k)] \quad \min \quad f(\bar{x}^k) = c$$
$$\text{s.t.} \quad C(\bar{x}^k, y),$$
$$y \in D_y.$$  

Since $\bar{x}^k$ is fixed in the subproblem the objective function is a constant, $c$. This is equivalent to solving the problem without an objective function, i.e., a feasibility problem. If a solution to $[SP(\bar{x}^k)]$ is obtained, the process terminates since an optimal solution $(\bar{x}^k, \bar{y}^k)$ is found. Otherwise, a feasibility cut,

$$[B(\bar{x}^k)] \quad \sum_{i \in I(\bar{x}^k)} (1 - x_i) \geq 1,$$

where $I(\bar{x}^k) = \{i \in I : \bar{x}_i^k = 1\}$, is found. This cuts off the current master problem solution $\bar{x}^k$ and it is added to the set of Benders cuts by the update, i.e., $B^{k+1}(x) = B^k(x) \cup B^k(\bar{x})$. To ensure convergence, at least $\bar{x}^k$ must be cut off, i.e., $\bar{x}^k$ must be infeasible in $[MP^{k+1}]$. It is possible to accelerate the method by using stronger feasibility cuts which removes a larger portion of the master problem solutions [10].

A visualisation of the LBBD scheme is found in Figure 2.1.

Figure 2.1: Graphical visualisation of the LBBD scheme

Pseudo-code of the LBBD scheme, where MP denotes the master problem with the set of feasibility cuts as input, is presented in Algorithm 1.
Algorithm 1: Pseudo-code of the LBBD scheme

Data: Problem [P]  
Result: A solution to problem [P]  
1 \( k \leftarrow 1; B^k(x) \leftarrow \emptyset; \)
2 \( \textbf{while } \text{True } \textbf{do} \)
3 \( \bar{x}, f(\bar{x}) \leftarrow \text{MP}^k; \)
4 \( \textbf{if } SP(\bar{x}) \text{ is feasible } \textbf{then} \)
5 \( \bar{y} \leftarrow SP(\bar{x}); \)
6 \( \text{return } \bar{x}, \bar{y}, f(\bar{x}); \)
7 \( \textbf{else} \)
8 \( B(\bar{x}) \leftarrow SP(\bar{x}); \)
9 \( B^{k+1}(x) \leftarrow B^k(x) \cup B(\bar{x}); \)
10 \( \textbf{end} \)
11 \( k \leftarrow k + 1; \)
12 \( \textbf{end} \)

2.2 Maximum feasible subset of constraints

A general optimisation problem usually contains a set of constraints that defines its feasible solutions. A feasible subset of constraints, for a problem,

\[
[P] \quad \min \quad f(x) \\
\text{s.t.} \quad C(x, y), \quad C(x), \quad x_i \in \{0, 1\}, i \in I \\
y \in D_y,
\]

is a subset that admits a feasible solution when the problem [P] is infeasible.

**Definition 2.2.1** (Maximal feasible subset of constraints). Given the optimisation problem [P], a maximal feasible subset of constraints (MaFS) is a subset \( \tilde{C} \subseteq C(x) \cup C(x, y) \) such that \( \tilde{C} \) includes a feasible solution and any subset \( \{ \tilde{C} : \tilde{C} \subset \tilde{C} \subseteq C(x) \cup C(x, y) \} \) does not.

**Definition 2.2.2** (Maximum feasible subset of constraints). The maximum feasible subsets of constraints (MuFS) of [P], are the MaFSs of [P] with the largest cardinality \( |\tilde{C}| \).

**Remark.** The number of constraints in an MaFS can vary and MaFSs and MuFSs are not necessarily unique.
Example 2.2.1. A small example is shown in Figure 2.2. Here, the problem is to place the circles so that there is no overlap. For example, the set \( \{A, B\} \), is infeasible since\( A \cap B \neq \emptyset \). To create an MaFS we start with a single circle, let’s say \( S = \{A\} \). Because A is alone, the set \( S \) is feasible. It is not possible to add either \( B \) or \( C \), but \( D \) is eligible. We update \( S = S \cup \{D\} = \{A, D\} \). We then see that \( E \) can be added while still ensuring feasibility. We update \( S = S \cup \{E\} = \{A, D, E\} \). We have now iterated through every circle and therefore \( S = \{A, D, E\} \) is an MaFS.

Remark. Similarly, if we had started with \( B \) we could have obtained both \( \{B, C\} \) and \( \{B, D, E\} \). Therefore, they are both MaFSs. Note that both \( \{B, D, E\} \) and \( \{A, D, E\} \) are MuFSs.

The problem finding an MaFS is easier than finding an MuFS. If \([P]\) is an LP-problem, the problem finding an MuFS, usually known as the maximum feasible subset problem is NP-hard [4]. There are multiple methods to find one or more MaFSs. Two such algorithms are presented. The grow algorithm will find an MaFS to the given set of constraints and the MARCO algorithm will use the grow algorithm to produce all MaFSs and thus can be used to find any MuFSs.

Grow

In [14], the grow algorithm is described. It starts from some feasible subset \( \text{seed} \subset C(x) \cup C(x, y) \) and will iteratively attempt to add constraints, checking each newly constructed set for feasibility and keeping changes where the set remains feasible. The algorithm is a generic method for finding an MaFS of a set of constraints, regardless of the problem type. It could also be modified...
to exploit characteristics for a specific problem. The pseudo-code is given in Algorithm 2.

**Algorithm 2:** Pseudo-code of the grow algorithm

```
Data: Problem [P]; Feasible set of constraints seed ⊂ C(x) ∪ C(x, y)
Result: An MaFS with respect to [P]
1 for s ∈ C(x) ∪ C(x, y) \ seed do
2     if seed ∪ {s} is feasible then
3         seed ← seed ∪ {s};
4     end
5 end
6 return seed;
```

**Remark.** Note that if seed = ∅, then the algorithm creates a new MaFS from an infeasible set. In Example 2.2.1 this algorithm is used.

**MARCO**

In [14], the MARCO algorithm is described. It is a method to enumerate all the MaFS to an infeasible constraint set C. The idea is to enumerate MaFS by repeatedly selecting unexplored subsets Ĉ ⊂ C. This is done by exploring a power set and exploiting the fact that a power set can be analyzed and manipulated by performing set operations within the power set. The algorithm contains a function MAP : \(\mathcal{P}(C) \to \{0, 1\}\), such that MAP\(\left(C'\right) = 1\) means that \(C' ⊂ C\) has not been checked. Hence, if \(m[x_i]\) returns True, then \(C_i\) has not been checked and is added to seed. If seed is feasible, GROW(seed) is used to create an MaFS. If seed is infeasible, SHRINK(seed) is used to create an irreducible infeasible subset of constraints (IIS). The operation BLOCKDOWN(seed), if added to MAP, will ensure that at least one of the constraint not in seed is included in a future search. The operation BLOCKUP(seed) will exclude at least one constraints in seed. The pseudo-code for is given in Algorithm 3.
2.3 Irreducible infeasible subset of constraints

Given the general optimisation problem \([P]\) in Section 2.2, we define the concept of an irreducible infeasible subset of constraints (IIS).

**Definition 2.3.1 (Irreducible infeasible subset of constraints).** Given the optimisation problem \([P]\), an irreducible infeasible subset of constraints (IIS) with respect to \([P]\) is a subset of constraints \(\tilde{C} \subseteq C(x) \cup C(x, y)\) such that \(\tilde{C}\) includes no feasible solutions and any subset \(\{\overline{C} : \overline{C} \subset \tilde{C}\}\) does.

**Remark.** The number of constraints in an IIS can vary and IISs are not necessarily unique.

**Example 2.3.1.** We reuse the example structure in Figure 2.2 to display the finding of an IIS. Here, the problem is to place the circles such that there are overlaps but if any of the included circles were to be removed, there would be no overlap left. To create an IIS we start with all circles, \(S = \{A, B, C, D, E\}\). Since there are overlaps between the circles in \(S\), the set is infeasible. First, we see that \(A\) can be removed from the set \(S\) to still ensure infeasibility, hence we remove \(A\) from the set. We update \(S = S \setminus \{A\} = \{B, C, D, E\}\). We then see that \(B\) also can be removed from the set to still ensure infeasibility. We update \(S = S \setminus \{B\} = \{C, D, E\}\). However, if \(C\) was removed from the set, there would be no overlaps, i.e., it would yield a feasible solution. Therefore,
Chapter 2. Theory

C is not removed from the set. We then see that D can be removed to ensure infeasibility. We update $S = S \setminus \{D\} = \{C, E\}$. Lastly, we see that if E was to be removed, there would be no overlaps. We have now iterated through every circle and found that the set $S = \{C, E\}$ is an IIS.

Remark. Similarly, if we had tried to remove the circles from the set in a different order the result could have been a different IIS. In this case, the IIS is not unique since \{A, B\}, \{A, C\} and \{C, D\} are also all IISs. Also note that the cardinality of each IIS in this example is 2. However, this is not always the case.

Finding an IIS is not always as trivial as in the above example, but there are different methods that guarantee that an IIS is produced. Two such algorithms are presented below. They both rely on solving a series of subproblems, which are used as infeasibility oracles. Note that there are also other methods that rely on using the problem structure [3].

Shrink

In [14], the shrink algorithm is described. It starts from some infeasible subset of constraints $seed \subseteq C(x) \cup C(x, y)$ and will iteratively attempt to remove one constraint in each iteration, checking each newly constructed set for infeasibility and keeping those changes that maintain the infeasibility. The algorithm is a generic method for finding an IIS of a set of constraints regardless of the problem structure, but it can be modified to exploit characteristics for a specific problem. The method is also known as deletion filtering [10]. Pseudo-code is given in Algorithm 4. The shrink algorithm is the one used in Example 2.3.1.

**Algorithm 4:** Pseudo-code of the shrink algorithm

| Data: Problem [P]; Infeasible set of constraints $seed \subseteq C(x) \cup C(x, y)$ |
| Result: An IIS with respect to [P] |
| 1 for $s \in seed$ do |
| 2 | if $seed \setminus \{s\}$ is infeasible then |
| 3 | seed $\leftarrow$ seed $\cup \{s\}$; |
| 4 | end |
| 5 end |
| 6 return seed; |

Depth-first binary search

In [10], the depth-first binary search (DFBS) algorithm is described. Just like the shrink algorithm, it is a method to find an IIS. Unlike the shrink algorithm,
it does not only evaluate one index at a time, but it instead evaluates larger subsets. The purpose of evaluating larger subsets at a time is to decrease the number of iterations needed to find an IIS. The algorithm starts from some infeasible subset of constraints $seed \subseteq C(x) \cup C(x, y)$ and will iteratively attempt to remove subsets of constraints, checking each newly constructed set for infeasibility and keeping those changes. If the newly constructed set becomes feasible, a larger subset of constraints $s \subseteq seed$ is chosen to be evaluated in the next iteration. Once an infeasible subset is found, the algorithm will never explore anything not included in that subset but is instead diving further into that subset. This is repeated until an IIS is acquired. Pseudo-code is given in Algorithm 5.

\begin{algorithm}
\textbf{Algorithm 5:} Pseudo-code of the DFBS-algorithm for finding an IIS.
\begin{algorithmic}[1]
\Data: Problem $[P]$; Infeasible set of constraints $seed \subseteq C(x) \cup C(x, y)$ 
\Result: An IIS with respect to $[P]$
\begin{algorithmic}
\State $T \leftarrow seed; S \leftarrow \emptyset; IIS \leftarrow \emptyset$
\While {$\text{True}$}
\If {$|T| \leq 1$}
\If {$T \cup IIS$ is infeasible}
\State return $T \cup IIS$
\EndIf
\State $T \leftarrow S; S \leftarrow \emptyset$
\EndIf
\If {$|T| \geq 2$}
\State go to Line 3;
\EndIf
\State $T_{2} \leftarrow T; T_{1} \leftarrow \emptyset$
\Else
\State Split $T$ into $T_{1}$ and $T_{2}$;
\EndIf
\If {$S \cup T_{1}$ is feasible}
\State $S \leftarrow S + T_{1}; T \leftarrow T_{2}$;
\Else
\State $T \leftarrow T_{1}$;
\EndIf
\EndWhile
\end{algorithmic}
\end{algorithm}
2.4 Strengthened feasibility cuts in LBBD using IISs

A feasibility cut $B(\bar{x})$ is defined as a strengthened feasibility cut with respect to some other feasibility cut $B(\bar{x}^k)$ if it prohibits additional, but only infeasible, solutions. Because of the assumption that only $x_i$ imposes a restriction in $[P]$, a feasibility cut $B(\bar{x}^k)$ is trivially strengthened if an updated feasibility cut $B(\bar{x})$ contains fewer variables than $B(\bar{x}^k)$ and SP($\bar{x}$) is infeasible for $I(\bar{x}) \subseteq I(\bar{x}^k)$.

**Example 2.4.1.** Let the set of binary master problem variables be $x = (x_1, x_2, x_3, x_4, x_5)$. Given that $I(\bar{x}^k) = \{1, 2, 3, 4, 5\}$, for which SP($\bar{x}^k$) is infeasible, the standard feasibility cut $B(\bar{x}) = (1 - x_1) + (1 - x_2) + (1 - x_3) + (1 - x_4) + (1 - x_5) \geq 1$, cuts off the solution $(1, 1, 1, 1, 1)$. If, for example, $I(\bar{x}) = \{1, 2, 3\}$, where $I(\bar{x}) \subseteq I(\bar{x}^k)$ makes SP($\bar{x}$) infeasible, a strengthened cut $B(\bar{x}) = (1 - x_1) + (1 - x_2) + (1 - x_3) \geq 1$, cuts off the master problem solutions $(1, 1, 1, 0, 0)$, $(1, 1, 1, 0, 1)$, $(1, 1, 1, 1, 0)$ and $(1, 1, 1, 1, 1)$ which are all infeasible in $[P]$ since they were infeasible in the subproblem. Thus, more infeasible solutions have now been cut from the solution space by strengthening the feasibility cut.

**Remark.** The number of solutions that are cut off is equal to two to the power of the number of variables removed from the original cut, e.g., in Example 2.4.1 the number of solutions cut in the strengthened cut is equal to $2^2 = 4$.

Using some method to strengthen the feasibility cuts in an LBBD scheme is a commonly used acceleration technique. In a previous work [10], the concept of IISs, described in Section 2.3, is used in order to obtain a strongest feasibility cut in an LBBD scheme like the one presented in Section 2.1. An irreducible feasibility cut is defined in an LBBD with respect to a master problem solution.

**Definition 2.4.1** (Irreducible feasibility cut in LBBD). A feasibility cut $B(\bar{x})$ is an irreducible feasibility cut (IFC) if subproblem SP($\bar{x}$) is infeasible and subproblem SP($\bar{x}$) is feasible for each $\tilde{x}$ such that $I(\tilde{x}) \subseteq I(\bar{x})$ holds.

In the context of LBBD, an IFC is constructed from the master problem variables $\bar{x}^k$ by solving a series of subproblems according to algorithms for finding an IIS. An IFC consists of an irreducible set of master problem variables.
that restricts the subproblem in such a way that it is infeasible. If a variable is
removed from the IFC, feasible solutions will be cut from the solution space, i.e
the cut cannot be strengthened further.

In Figure 2.3, it is visualised how IFCs are used in the LBBD scheme. The
function IFC(B(\bar{x}^k)) produces an IFC B(\bar{x}).

\begin{center}
\begin{tikzpicture}
    \node [circle,draw] (MP) at (0,0) {MP\(^k\)};
    \node [circle,draw] (B_bar_x) at (-1,-1) {B(\bar{x})};
    \node [circle,draw] (SP_bar_x) at (1,-1) {SP(\bar{x}^k)};
    \node [circle,draw] (IFC_bar_x) at (0,-2) {IFC(B(\bar{x}^k))};
    \draw [->] (MP) to (SP_bar_x);
    \draw [->] (SP_bar_x) to node [below] {Feasible} (IFC_bar_x);
    \draw [->] (IFC_bar_x) to node [left] {Not feasible} (B_bar_x);
    \draw [->] (B_bar_x) to (MP);
    \node at (0,-3) {Solution found: f(\bar{x}^k),(\bar{x}^k,\bar{y}^k)};
\end{tikzpicture}
\end{center}

Figure 2.3: Graphical visualisation of the LBBD scheme using IFCs

The corresponding pseudo-code describing the LBBD scheme using IFCs is
presented in Algorithm 6. The function denoted IFC(B(\bar{x}^k)) can be any algo-
rithm that constructs an IFC. Two examples of such algorithms are presented
in [10]. The depth-first binary search (DFBS) algorithm described is a version
of Algorithm 5, modified to suit an LBBD scheme [12, 15]. It can be used as
the function IFC(B(\bar{x}^k)) and its pseudo-code is presented in Algorithm 7.
Algorithm 6: Pseudo-code of the LBBD scheme using IFCs

Data: Problem [P]; Algorithm for constructing an IFC
Result: A solution to problem [P]

1. $k \leftarrow 1; B^k(x) \leftarrow \emptyset$
2. while True do
   3. $\bar{x}^k, \bar{f}(\bar{x}^k) \leftarrow MP^k$
   4. if $SP(\bar{x}^k)$ is feasible then
      5. $\bar{y}^k \leftarrow SP(\bar{x}^k)$
      6. return $\bar{x}^k, \bar{y}^k, \bar{f}(\bar{x}^k)$
   else
      8. $B(\bar{x}^k) \leftarrow SP(\bar{x}^k)$
      9. $B(\bar{x}) \leftarrow IFC(B(\bar{x}^k))$
     10. $B^{k+1}(x) \leftarrow B^k(x) \cup B(\bar{x})$
   end
   11. $k \leftarrow k + 1$
3. end
Algorithm 7: Pseudo-code of the DFBS-algorithm for IFC

Data: A feasibility cut $B(\bar{x}^k)$
Result: An IFC $B(\bar{x})$

1: $T \leftarrow I(\bar{x}); S \leftarrow \emptyset; \bar{x}_i \leftarrow 0, i \in I$
2: while True do
3:  if $|T| \leq 1$ then
4:     $\bar{x}_i \leftarrow 1, i \in T$
5:     if $SP(\bar{x})$ is infeasible then
6:         return $B(\bar{x})$
7:     end
8:     $T \leftarrow S; S \leftarrow \emptyset$
9:     if $|T| \geq 2$ then
10:        go to Line 3;
11:     end
12:    $T_2 \leftarrow T; T_1 \leftarrow \emptyset$
13:  else
14:     Split $T$ into $T_1$ and $T_2$
15:  end
16:  $\bar{x}_i \leftarrow 1, i \in S \cup T_1$
17:  if $SP(\bar{x})$ is feasible then
18:     $S \leftarrow S + T_1; T \leftarrow T_2$
19:  else
20:     $T \leftarrow T_1$
21:  end
22:  $\bar{x}_i \leftarrow 0, i \in S \cup T_1$
23: end
Chapter 3

Acceleration techniques for logic-based Benders decomposition

While logic-based Benders decomposition (LBBDD) is a good technique for solving large-scale optimisation problems, the technique itself does not provide any information about a feasible solution if the subproblem is of feasibility-type as in Section 2.1. This chapter contains our main contributions about how the concept of a maximal feasible subset of constraints, described in Section 2.2, can be applied to an LBBD scheme to accommodate this.

First, we present some theory regarding when a feasible solution can be obtained when solving a subproblem. Second, we define the concept of a maximal feasible solution. Third, a connection is made between maximal feasible solutions in LBBD and the concept of maximal feasible subset of constraints (MaFS). Finally, the last three sections describe an algorithm for finding a maximal feasible solution in LBBD, how the concepts of maximal feasible solutions and irreducible feasibility cuts (IFC) can be combined in LBBD and how an enhanced version of the IISs can be used in LBBD, respectively.

3.1 Feasible solutions in LBBD

When the subproblem defined in Section 2.1 is infeasible for some $x = \bar{x}^k$, there exists no $y$ such that $(\bar{x}^k, y)$ is feasible in [P]. Instead of letting $M^k+1$ present a new solution, one can modify the existing solution $\bar{x}^k$ to $\tilde{x}^k$ in an attempt to
find some $y$ such that $(\bar{x}^k, y)$ is feasible in $[P]$. Such a search can be performed by solving additional subproblems. To approach this theoretically, we have made a few assumptions about the structure of the problem $[P]$ and a decomposition with a master problem $\text{MP}^k$ and a subproblem $\text{SP}(x^k)$.

**Assumption 1.** Let $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_n)^T$ be feasible in $\text{MP}^k$, then for any $i \in \{1, \ldots, n\}$, $\bar{x} = (\bar{x}_1, \ldots, \bar{x}_{i-1}, 0, \bar{x}_{i+1}, \ldots, \bar{x}_n)^T$ is also feasible in $\text{MP}^k$.

**Assumption 2.** The function $f(x)$, $x = (x_1, \ldots, x_n)^T$ is written on the form $f(x) = \sum_{i=1}^n a_i x_i$, where $a_i \leq 0$.

**Remark.** As assumed in Section 2.1, $C(x, y)$ has a structure such that only $x_i = 1$ restricts the values $y$ can take.

From this point on, we will assume that these assumptions apply. Generally, this means that we only handle variables $x$ which are solutions to $\text{MP}^k$. If $\text{SP}(x')$ is feasible with the solution $y'$, then $(x', y')$ solves the problem $[P]$. This follows directly from the decomposition strategy presented in Section 2.1, where the subproblem is extracted by fixing $x$ in $[P]$.

Conversely, if $x$ is chosen such that $\text{SP}(x)$ is infeasible and there exists a subset $I(x') \subset I(x)$ such that $\text{SP}(x')$ is feasible with the solution $y'$. Then $x'$ is a solution to the master problem and a solution to $[P]$ is given by $(x', y')$ with the objective value $f(x')$. Since $(x', y')$ is feasible in $[P]$, a pessimistic bound on the objective is given by $f(x')$ and since $[P]$ is a minimisation problem, the pessimistic bound is an upper bound. This is formalised in the following proposition.

**Proposition 1.** Let $x = \bar{x}$ be a solution to $\text{MP}^k$ for which $\text{SP}(\bar{x})$ is infeasible. For any subset $I(x') \subset I(\bar{x})$, for which $\text{SP}(x')$ is feasible with the solution $y'$, the pair $(x', y')$ is feasible in $[P]$ with the objective value $f(x') \geq f(\bar{x})$.

**Proof.** Since $\text{SP}(x')$ is feasible with the solution $y'$, $C(x', y')$ and $y \in D_y$ is trivially satisfied. Further, since $I(x') \subset I(\bar{x})$, we know that the only difference between the values of $x'$ and $\bar{x}$ are that for some indices $i \in I(\bar{x})$, $x'_i = 0$. That is, $x'$ is binary and according to Assumption 1, changing a value from 1 to 0 in a feasible solution will never cause infeasibility. That is, the constraints $C(x')$ and $x' \in \{0, 1\}$ are always satisfied. Therefore, all constraints in $[P]$ is satisfied by the solution $(x', y')$. Because $\sum_{i=1}^n a_i x'_i$ has fewer non-zero terms than $\sum_{i=1}^n a_i x_i$ and $a_i \leq 0$, we have $f(x') \geq f(x)$.
3.2 Maximal feasible solutions in LBBD

When exploring feasible solutions in the LBBD it might be of interest to find such a solution that includes a maximal number of variables. These types of solutions can strengthen the pessimistic bound as much as possible. For that purpose, we define a maximal feasible solution in a LBBD with respect to a master problem solution.

**Definition 3.2.1** (Maximal feasible solution in LBBD). A solution \( \hat{x} \) is a maximal feasible solution (MFS) of \( \bar{x} \) if the subproblem \( \text{SP}(\hat{x}) \) is feasible and the subproblem \( \text{SP}(\tilde{x}) \) is infeasible for each \( \tilde{x} \) such that \( I(\hat{x}) \subset I(\tilde{x}) \subseteq I(\bar{x}) \) holds.

**Remark.** The number of non-zero variables in an MFS can vary.

**Remark.** The solution \( \bar{x} \) is an MFS if \( \text{SP}(\bar{x}) \) is feasible.

3.3 LBBD using MaFS

We note that the master problem variables \( x \) have a direct connection to the subproblem constraints \( C(x, y) \). We can directly associate each variable in \( x \) to a constraint in \( C(x, y) \) and vice versa. Due to the similarities, we can therefore use MaFS algorithms to find MFSs in an LBBD scheme.

Unlike using some cut strengthening method, e.g., the one described in Section 2.4, constructing MFSs to obtain pessimistic bounds will not necessarily provide a faster convergence in the LBBD scheme. However, while constructing an MFS, it is possible to find subsets of a master problem solution for which the subproblem is infeasible. This property makes it possible to find feasibility cuts that are stronger than the standard feasibility cut. We can therefore speed up the convergence of the LBBD scheme, without impairing the computational performance of the algorithm.

We introduce a function \( \text{MFS}(B(\bar{x}^k)) \) that takes a feasibility cut as input and constructs an MFS to obtain a pessimistic bound \( \bar{z} \) and a strengthened feasibility cut \( B(\bar{x}) \). In Figure 3.1, it is visualised how the concept of MFSs is applied to the LBBD scheme, via the function \( \text{MFS}(B(\bar{x}^k)) \) that is performed after encountering an infeasible subproblem.

The pseudo-code corresponding to the LBBD scheme in Figure 3.1, is presented in Algorithm 8. The function \( \text{MFS}(B(\bar{x}^k)) \) can be any algorithm that constructs an MFS. The depth-first binary search algorithm for constructing an MFS is an example of such an algorithm. The algorithm is presented in Section 3.4.
Figure 3.1: Graphical visualisation of the LBBD scheme using MFSs

**Algorithm 8:** Pseudo-code of the LBBD scheme using MFSs

**Data:** Problem \([P]\); Algorithm for constructing an MFS \(\textbf{MFS}(B(\bar{x}^k))\)

**Result:** A solution to problem \([P]\), \((\bar{x}^k, \bar{y}^k), f(\bar{x}^k)\)

1. \(k \leftarrow 1; B^k(x) \leftarrow \emptyset;\)
2. \(\textbf{while} \ True \ \textbf{do}\)
3. \(\quad \bar{x}^k, f(\bar{x}^k) \leftarrow \text{MP}^k;\)
4. \(\quad \textbf{if} \ SP(\bar{x}^k) \ \textbf{is feasible} \ \textbf{then}\)
5. \(\quad \quad \bar{y}^k \leftarrow SP(\bar{x}^k);\)
6. \(\quad \quad \text{return } \bar{x}^k, \bar{y}^k, f(\bar{x}^k)\)
7. \(\quad \textbf{else}\)
8. \(\quad \quad B(\bar{x}^k) \leftarrow SP(\bar{x}^k);\)
9. \(\quad \quad B(\bar{x}), z \leftarrow \textbf{MFS}(B(\bar{x}^k));\)
10. \(\quad \quad B^{k+1}(x) \leftarrow B^k(x) \cup B(\bar{x})\)
11. \(\quad \textbf{end}\)
12. \(\quad k \leftarrow k + 1;\)
13. \(\textbf{end}\)
3.4 Depth-first binary search algorithm designed for finding MFSs in LBBD

To construct MFSs in an LBBD scheme, as described in Section 3.2, we develop a depth-first binary search (DFBS) algorithm that constructs an MFS $\bar{x}$ from an infeasible solution $\tilde{x}$. From this MFS, a pessimistic bound can easily be obtained. A brief description of the algorithm is given below, and its pseudo-code is presented in Algorithm 9.

The algorithm iteratively attempts to make two types of permanent assignments, $\bar{x}_i = 0$ or $\bar{x}_i = 1$ for $i \in I(\tilde{x})$, by evaluating subsets of indices. Note that, as mentioned in Section 3.1, $\bar{x}_i = 1 \rightarrow 0$ relaxes the subproblem for $i \in I(\tilde{x})$. When $\bar{x}_i$ has been permanently assigned a value for all $i \in I(\tilde{x})$, the algorithm is finished. Initially, $\bar{x}_i = 0$ for all $i \in I(\tilde{x})$. Temporary assignments of the type $\bar{x}_i = 1$ are made iteratively for subsets of multiple indices $i \in I(\tilde{x})$. After each temporary assignment, a subproblem is solved. If the subproblem is feasible, the temporary assignments $\bar{x}_i = 1$, for $i \in I(\tilde{x})$, become permanent assignments. However, if a subproblem becomes infeasible, the temporary assigned variables are further explored until either a smaller subset becomes feasible in the subproblem or until only one variable is temporarily assigned $\bar{x}_i = 1$ for $i \in I(\tilde{x})$. If, in the latter case, the subproblem becomes infeasible, the permanent assignment $\bar{x}_i = 0$, for $i \in I(\tilde{x})$ is made. Further, all still non-permanently assigned variables are explored. When all variables $\bar{x}_i$, for $i \in I(\tilde{x})$, has been permanently assigned a value, an MFS $\bar{x}$ has been found. A pessimistic bound is then obtained by calculating the objective $f(\bar{x})$.

3.5 MFSs and IFCs in LBBD

To develop an LBBD scheme that constructs both IFCs and MFSs, we combine the concepts presented in Sections 2.4 and 3.3, respectively. In Figure 3.2, it is visualised how MFSs and IFCs, together, are applied to the LBBD scheme. The function $\text{MFS}(B(\bar{x}^k))$ provides a pessimistic bound and a strengthened feasibility cut that is then further strengthened by the function $\text{IFC}(B(\bar{x}^k))$ to an IFC.

The pseudo-code corresponding to the LBBD scheme in Figure 3.2, is presented in Algorithm 10. The functions $\text{IFC}(B(\bar{x}^k))$ and $\text{MFS}(B(\bar{x}^k))$ can be any arbitrary algorithms that construct an IFC and an MFS, respectively. For instance, the algorithms presented in Algorithms 6 and 8 can be used, respectively.
Algorithm 9: Pseudo-code of the DFBS-algorithm for MFS

Data: A feasibility cut $B(\bar{x}^k)$; An objective function $f$

Result: A pessimistic bound $f(\bar{x})$; A strengthened feasibility cut $B(\bar{x})$

1. $T \leftarrow I(\bar{x}); S \leftarrow \emptyset; F \leftarrow \emptyset; J \leftarrow \emptyset; \bar{x}_i \leftarrow 0, i \in I; B(\bar{x}) \leftarrow B(\bar{x}^k)$

2. while True do

3.     if $|T| \leq 1$ then

4.         if $|F| + |J|$ is equal to $|I|$ then

5.             $\bar{x}_i \leftarrow 1, i \in F$;

6.             return $f(\bar{x}), B(\bar{x})$

7.         end

8.     $\bar{x}_i \leftarrow 1, i \in F \cup T$;

9.     if $SP(\bar{x})$ is feasible then

10.        $F \leftarrow F + T$

11.        $J \leftarrow J + T$

12.    end

13.    $T \leftarrow S; S \leftarrow \emptyset$

14.    end

15.    Split $T$ into $T_1$ and $T_2$; $\bar{x}_i \leftarrow 1, i \in F \cup T_1$

16.    if $SP(\bar{x})$ is feasible then

17.        $F \leftarrow F + T_1; T \leftarrow S + T_2; S \leftarrow \emptyset$

18.    else

19.        $B(\hat{x}) \leftarrow SP(\bar{x})$

20.        if $B(\hat{x})$ is stronger cut than $B(\bar{x})$ then

21.            $B(\bar{x}) \leftarrow B(\hat{x})$

22.        end

23.        $\bar{x}_i \leftarrow 0, i \in T_1, \bar{x}_i \leftarrow 1, i \in T_2$

24.        if $SP(\bar{x})$ is feasible then

25.            $F \leftarrow F + T_2; T \leftarrow S + T_1; S \leftarrow \emptyset$

26.        else

27.            $T \leftarrow T_1; S \leftarrow S + T_2$

28.        end

29.    end

30. end

31. $\bar{x}_i \leftarrow 0, i \in S \cup T \cup J$
Algorithm 10: Pseudo-code of the LBBD scheme using MFSs and IFCs

**Data:** Problem [P]; Algorithm for constructing an MFS \( \text{MFS}(B(\bar{x}^k)) \);
Algorithm for constructing an IFC \( \text{IFC}(B(\bar{x})^k) \);

**Result:** A solution to problem [P], \((\bar{x}^k, \bar{y}^k), f(\bar{x}^k)\)

1. \( k \leftarrow 1; B^k(x) \leftarrow \emptyset \);
2. while True do
3. \( \bar{x}^k, f(\bar{x}^k) \leftarrow \text{MP}^k \);
4. if \( \text{SP}(\bar{x}^k) \) is feasible then
5. \( \bar{y}^k \leftarrow \text{SP}(\bar{x}^k) \);
6. return \( \bar{x}^k, \bar{y}^k, f(\bar{x}^k) \);
7. else
8. \( B(\bar{x}^k) \leftarrow \text{SP}(\bar{x}^k) \);
9. \( B(\bar{x}), \bar{z} \leftarrow \text{MFS}(B(\bar{x}^k)) \);
10. \( B(\bar{x}) \leftarrow \text{IFC}(B(\bar{x})) \);
11. \( B^{k+1}(x) \leftarrow B^k(x) \cup B(\bar{x}) \);
12. end
13. \( k \leftarrow k + 1 \);
14. end

---

Figure 3.2: Graphical visualisation of the LBBD using MFSs and IFCs
3.6 Enhanced use of IISs in LBBD

In Section 2.4, it is described how the concept of IIS in previous work has been used to obtain IFCs in an LBBD scheme. During this study we have broadened the scope of the IFCs finding algorithm, Algorithm 7, to also provide a pessimistic bound. This is done by tracking feasible solutions encountered while constructing the IFCs. The method does not give any guarantees on the pessimistic bound, but it is found without further burdening the method from a computational standpoint. This is especially useful when comparing different methods in Chapter 6. In Figure 3.3 it is visualised how the enhanced IFC finding algorithm is applied to the LBBD scheme. The corresponding pseudo-code of the LBBD scheme using the enhanced IFC algorithm is presented in Algorithm 11.

![Figure 3.3: Graphical visualisation of the LBBD scheme using enhanced IFCs](image)

Solution found: $f(\bar{x}^k), (\bar{x}^k, \bar{y}^k)$
3.6. Enhanced use of IISs in LBBD

Algorithm 11: Pseudo-code of the LBBD scheme using an enhanced IFC algorithm

Data: Problem [P]; Algorithm for constructing an IFC $\text{IFC}(B(\bar{x}^k))$

Result: A solution to problem [P], $(\bar{x}^k, \bar{y}^k), f(\bar{x}^k)$

1. $k \leftarrow 1; B^k(x) \leftarrow \emptyset$
2. while True do
   3. $\bar{x}^k, f(\bar{x}^k) \leftarrow \text{MP}^k$
   4. if $\text{SP}(\bar{x}^k)$ is feasible then
      5. $\bar{y}^k \leftarrow \text{SP}(\bar{x}^k)$
      6. return $\bar{x}^k, \bar{y}^k, f(\bar{x}^k)$
   7. else
      8. $B(\bar{x}^k) \leftarrow \text{SP}(\bar{x}^k)$
      9. $B(\bar{x}), z \leftarrow \text{IFC}(B(\bar{x}^k))$
     10. $B^{k+1}(x) \leftarrow B^k(x) \cup B(\bar{x})$
     11. end
   12. $k \leftarrow k + 1$
13. end
Chapter 4

Problem formulation

This chapter introduces a multiprocessor scheduling problem with multiple time windows, positive time lags and exact time lags that we have used in our computational evaluations.

The multiprocessor scheduling problem originates from [2, 13] and aims to create a schedule for a set of processors. The details of the problem we address are as follows. Each processor processes several tasks. Each processor can only process one task at a time. Each task has been assigned to exactly one processor beforehand. Each task has a processing time and a number of time windows during which the task can be scheduled. Each time window is specified by a release time and a deadline that indicates the first possible start time and last possible end time for the task, respectively. Additionally, some tasks have a dependence in between them. These dependencies are referred to as having either exact or positive time lags. They connect a pair of tasks through their start times such that there must be an exact or minimal time between them, respectively.

A feasible schedule is obtained when each scheduled task is completely processed within one of its time windows, there are no overlapping tasks on a processor, and all exact and positive time lags are respected for all scheduled tasks. For our problem instances, it is not known in advance if all tasks can be feasibly scheduled or not. The aim is therefore to find a solution where the number of non-scheduled tasks is minimised. For non-scheduled tasks, all time lags connected to these tasks are ignored. This is true even if one of the tasks in a time lag pair is scheduled.

Remark. Note that a schedule with no tasks is trivially feasible.

To be able to formulate the requirements of a feasible solution mathemati-
cally, we first define notation for the information given in a problem instance.

Let the set of available processors for an instance be denoted $H$ and let the set $I_h$ contain of tasks beforehand assigned to processor $h \in H$. Each task $i \in I_h$ assigned to processor $h \in H$ has a processing time $p_i \in \mathbb{Z}^+$ and a set of time windows $Q_i$. Each time window $q \in Q_i$ for a given task $i \in I_h$ on processor $h \in H$ consists of a release time $r_{iq} \in \mathbb{Z}^+$ and a deadline $d_{iq} \in \mathbb{Z}^+$. Each instance also contains two sets of task pair indices, $T_p \subset [(i_1, i_2) \in \cup \cup \cup I_h \times \cup \cup \cup I_h]$ and $T^e \subset [(i_1, i_2) \in \cup \cup \cup I_h \times \cup \cup \cup I_h]$. The task pair indices in $T_p$ and $T^e$ will, for simplicity, be referred to as positive time lags and exact time lags, respectively. For each positive time lag $(i_1, i_2) \in T_p$, there is minimum length $l_{i_1i_2} \in \mathbb{Z}^+$, which is the positive lag. For each exact time lag, there is an exact length $l_{i_1i_2} \in \mathbb{Z}^+$, which is the exact lag. For the pair of indices $(i_1, i_2) \in T_p \cup T^e$, $i_1$ is referred to as the start task and $i_2$ is referred to as the end task.

The set of scheduled tasks is defined as $\bar{I} = \{i \in \cup \cup \cup I_h : i \text{ is scheduled}\}$ where $\bar{I} \subseteq \cup \cup \cup I_h$. The solution for a problem instance is presented as a schedule in the form of a set of scheduled tasks $\bar{I}$ together with a set $S = \{s_i \in \mathbb{Z}^+ : i \in \bar{I}\}$ which states the integer start times $s_i$ of each task $i \in \bar{I}$. We are now ready to formulate the objective function and the requirements to obtain a feasible schedule for the problem.

**Objective function:** The aim is to minimise the number of tasks $i \notin \bar{I}$.

**Req.1:** Each task $i \in \bar{I}$ is scheduled completely within one time window $q \in Q_i$ for the duration of its processing time $p_i$. This is fulfilled if for all starting times $s_i$ for scheduled tasks $i \in \bar{I}$ it holds that $s_i \geq r_{iq}$ and $s_i + p_i \leq d_{iq}$ for some $q \in Q_i$.

**Req.2:** A maximum of one task is processed on each processor $h \in H$ at any given time. This is true if the inequality $|\{i \in \cup \cup \cup I_h : s_i \leq t < s_i + p_i\}| \leq 1$ holds for each time step $t \in \mathbb{Z}^+$. Thus, it is allowed that a task ends, and another task begins at the same time step.

**Req.3:** For each positive time lag $(i_1, i_2) \in T_p$, task $i_1$ is to start at least $l_{i_1i_2}$ time steps after the start of task $i_2$ if $i_1, i_2 \in \bar{I}$. This holds if $s_{i_1} \leq s_{i_2} + l_{i_1i_2}$ is true for all positive time lags $(i_1, i_2) \in T_p$ such that tasks $i_1, i_2 \in \bar{I}$.

**Req.4:** For each exact time lag $(i_1, i_2) \in T^e$, task $i_1$ is to start at exactly $l_{i_1i_2}$ time steps after the start of task $i_2$ if $i_1, i_2 \in \bar{I}$. This holds if $s_{i_1} = s_{i_2} + l_{i_1i_2}$ is true for all exact time lags $(i_1, i_2) \in T^e$ such that tasks $i_1, i_2 \in \bar{I}$.

**Example 4.1.1.** Given an instance with the processors $H = \{0, 2\}$ and the tasks $I_1 = \{1, 2, 3\}$ and $I_2 = \{4, 5\}$ the aim is to find a schedule which schedules as many tasks as possible and at the same time fulfills the Req.1–4 above. The tasks
have the processing times \( (p_i)_{i=1}^{5} = (3, 3, 2, 5, 5) \). The available time windows for each task are described in Table 4.1 whereas all time lags are described in Table 4.2. One possible solution to the problem is illustrated in Figure 4.1.1 where the orange part of the time windows indicates exactly when each task is to be processed.

Table 4.1: Length of the respective time windows

<table>
<thead>
<tr>
<th>Time windows</th>
<th>Tasks</th>
<th>Task1</th>
<th>Task2</th>
<th>Task3</th>
<th>Task4</th>
<th>Task5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1, q_4, q_6, q_{10}, q_{11} )</td>
<td>0 → 3</td>
<td>3 → 6</td>
<td>1 → 3</td>
<td>4 → 12</td>
<td>0 → 5</td>
<td></td>
</tr>
<tr>
<td>( q_2, q_5, q_7, -, q_{12} )</td>
<td>6 → 10</td>
<td>8 → 12</td>
<td>4 → 6</td>
<td>11 → 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_3, -, q_8, -, - )</td>
<td>12 → 15</td>
<td>8 → 10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( -, -, q_9, -, - )</td>
<td>14 → 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Length of the respective time lags

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Time lag type</th>
<th>Exact</th>
<th>Positive</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (\text{Task2}, \text{Task3}) )</td>
<td>2</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>( (\text{Task4}, \text{Task3}) )</td>
<td>-</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>( (\text{Task5}, \text{Task2}) )</td>
<td>4</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>( (\text{Task5}, \text{Task4}) )</td>
<td>-</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Remark. An alternative schedule would be obtained if Task1 instead had been scheduled during time window \( q_1 \). This schedule would also have scheduled all tasks.
Figure 4.1: A solution to the Example 4.1.1. The orange color represents a scheduled task.
Chapter 5

Implementation of a logic-based Benders decomposition

This chapter describes the implementation of the LBBD scheme used for solving the problem presented in Chapter 4. First, a mathematical description of the full problem is presented as a combination of a mixed-integer problem (MIP) and a constraint programming (CP) problem. It is followed by a description of the decomposition used and an explanation of its including elements.

5.1 Mathematical model

The binary variable $x_{iq}$ indicates whether task $i \in I_h$ on processor $h \in H$ is scheduled in one of its time windows $q \in Q_i$ and defined as

$$x_{iq} = \begin{cases} 1, & \text{if task } i \text{ is scheduled in time window } q, \\ 0, & \text{otherwise.} \end{cases}$$

The positive integer variable $y_i$ represents when task $i \in I_h$ on processor $h \in H$ is scheduled to start and defined as

$$y_i = \text{start time of task } i.$$ 

A mathematical model of the problem presented in Chapter 4, given on the
same form as [P] in Section 2.1, is

\[
[P] \quad \min \sum_{h \in H} \sum_{i \in I_h} \left( 1 - \sum_{q \in Q_i} x_{iq} \right)
\]

s.t.

\[
\sum_{q \in Q_i} x_{iq} \leq 1, \quad i \in I_h, \ h \in H,
\]

(5.2)

\[
x_{iq} \rightarrow r_{iq} \leq y_i \leq d_{iq} - p_i, \quad q \in Q_i, \ i \in I_h, \ h \in H,
\]

(5.3)

\[
y_i + l_{ij} \leq y_j, \quad (i, j) \in T^p,
\]

(5.4)

\[
y_i + l_{ij} = y_j, \quad (i, j) \in T^e,
\]

(5.5)

\[
x_{iq} \in \{0, 1\}, \quad q \in Q_i, \ i \in I_h, \ h \in H,
\]

\[
y_i \in \mathbb{Z}^+, \quad i \in I_h, \ h \in H.
\]

(5.6)

The objective function (5.1) minimizes the number of non-scheduled tasks. The constraint (5.2) ensures that each task can at most be scheduled in one of its time windows (Req.1 in Chapter 4). The disjunctive constraint (5.3) handles that no tasks overlap (Req.2 in Chapter 4). The constraint (5.4) restricts the start time \(y_i\) to only be set within the time window \(q\) that satisfies \(x_{iq} = 1\) and connects the assigned time windows with decision about start times. The constraints (5.5)–(5.6) handles positive and exact time lag, respectively, (Req.3–4 in Chapter 4).

**Remark.** The variables \(x\) from Section 2.1 relate as \(x = \{x_{iq} : q \in Q_i, i \in I_h\}\).

The constraint (5.2) corresponds to \(C(x)\) in [P] from Section 2.1. In (5.4), assigning the value \(x_{iq} = 1\) imposes a restriction on the variable \(y_i\) for \(q \in Q_i\) for \(i \in I_h\) for \(h \in H\) while \(x_{iq} = 0\) imposes no restriction. The constraints (5.3)–(5.6) corresponds to \(C(\bar{x}, y)\) in [P] from Section 2.1.

The following sections will describe the decomposition.

## 5.2 Decomposition

To solve problem [P], a logic-based Benders decomposition is presented. The master problem has an assignment-type characteristic. Each choice \(x_{iq} = 1\) imposes a restriction on \(y_i\) for \(q \in Q_i\) for \(i \in I_h\) for \(h \in H\), that does not depend on other decisions \(x_{jq} = 1\) for \(q \in Q_j\) for \(j \in I_h \setminus \{i\}\) for \(h \in H\).

To solve the master problem, a MIP-model is presented. The master problem
in iteration \( k \) is

\[
\begin{align*}
[\text{MP}^k] \quad & \min \sum_{h \in H} \sum_{i \in I_h} \left( 1 - \sum_{q \in Q_i} x_{iq} \right) \\
\text{s.t.} \quad & (5.2) \\
& B^k(x), \quad (5.7) \\
& \text{[Subproblem relaxation]}, \quad (5.8) \\
& x_{iq} \in \{0, 1\}, \quad q \in Q_i, \ i \in I_h, \ h \in H,
\end{align*}
\]

where constraint (5.7) are the set of feasibility cuts, generated in the previous LBBBD iterations. The [Subproblem relaxation] constraints (5.8) are used to strengthen the master problem solution and will be presented in Section 5.2.1.

We denote an optimal solution to the master problem in iteration \( k \) as \( \bar{x}^k \).

The subproblem obtained in iteration \( k \) is

\[
\begin{align*}
[\text{SP}(\bar{x}^k)] \quad & \min 0 \\
\text{s.t.} \quad & \text{DISJUNCTIVE}((y_i | i \in I_h), (p_i | i \in I_h)), \quad h \in H, \\
& \text{STARTBEFORESTART}(y_{i_1}, y_{i_2}, l_{i_1 i_2}), \quad (i_1, i_2) \in T^p, \\
& \text{STARTATSTART}(y_{i_1}, y_{i_2}, l_{i_1 i_2}), \quad (i_1, i_2) \in T^e,
\end{align*}
\]

where constraint \text{DISJUNCTIVE} is related to (5.3). The constraints \text{STARTBEFORESTART} and \text{STARTATSTART} relate to (5.5) and (5.6), respectively. A more detailed description of the definitions of the constraints is given in Appendix A.1.

If \( \text{SP}(\bar{x}^k) \) is feasible with the solution \( \bar{y}^k \), an optimal solution \((\bar{x}^k, \bar{y}^k)\) to problem [P] is obtained. If \( \text{SP}(\bar{x}^k) \) is infeasible, a feasibility cut according to Section 2.1 is chosen to make the solution \( \bar{x}^k \) infeasible in \( \text{MP}^{k+1} \). The feasibility cut generated in iteration \( k \) is

\[
[\text{B}(\bar{x}^k)] \quad \sum_{(i, q) \in \bar{I}^k} (1 - x_{iq}) \geq 1,
\]

where \( \bar{I}^k = \{(i, q) : q \in Q_i, i \in \bigcup_{h \in H} I_h, \bar{x}_{iq}^k = 1\} \). Let \( B^1(x) = \emptyset \) and thereafter \( B^{k+1}(x) = B^k(x) \cup B(\bar{x}^k) \).

### 5.2.1 Subproblem relaxation

In order to decrease the number of master problem iterations, a subproblem relaxation is added to the master problem. The intention is to avoid performing unnecessary master problem iterations. This is done by preventing the master
Chapter 5. Implementation of a logic-based Benders decomposition

problem from choosing values of the master problem variables that yield infeasibility in SP. Our subproblem relaxation consists of two parts, handling overlap and time lags, respectively.

The subproblem relaxation that handles overlap is based on creating segments from all possible release times to all possible deadlines, for every task. The subproblem relaxation enforces that the total processing time for tasks scheduled during time windows, that are completely enclosed within a segment, cannot exceed the length of the segment. This is a relaxation of the constraint DISJUNCTIVE and is formulated as

\[ \sum_{i \in I} \sum_{q \in Q_i(t_1, t_2)} p_i x_{iq} \leq t_2 - t_1, \quad (t_1, t_2) \in T, \]

where \( T = \{(r_{iq}, d_{iq}) \in Q_i \times Q_{i'}, (i, i') \in I \times I, d_{iq} > r_{iq}\} \) is the set of release time and deadline pairs for each task. The set \( Q_i(t_1, t_2) = \{q \in Q_i : t_1 \leq r_{iq}, d_{iq} \leq t_2\} \) gives the time windows \( q \in Q_i \) for task \( i \in I_h \) on processor \( h \in H \) that starts after time step \( t_1 \) and ends before time step \( t_2 \), i.e., every time window that is completely inside of \( (t_1, t_2) \in T \). Some of the calculated segments are viewed as unnecessary and they are removed in the implementation for computational reasons.

Further, the subproblem relaxation that handles time lags is obtained by relaxing the constraints STARTBEFORESTART and STARTATSTART. The resulting constraints are

\begin{align*}
\min_{q \in Q_i} d_{iq} + \sum_{q \in Q_i} (d_{iq} - \min_{q \in Q_i} d_{iq}) x_{iq} &\leq z_i, \quad i \in I_h, h \in H, \\
\max_{q \in Q_i} d_{iq} - \sum_{q \in Q_i} (\max_{q \in Q_i} d_{iq} - d_{iq}) x_{iq} &\geq z_i, \quad i \in I_h, h \in H, \\
z_j &\geq z_i + l_{ij}, \quad (i, j) \in T^p, \\
z_j &\geq z_i + l_{ij}, \quad (i, j) \in T^e,
\end{align*}

which takes a continuous variable \( z_i \geq 0 \) to symbolise the start time of tasks \( i \in I_h \) for \( h \in H \). These constraints enforce that each task \( i \) is scheduled in a time window which allows all time lags associated with the task to be scheduled.
Chapter 6

Results

This chapter describes the computational results achieved by different logic-based Benders decomposition (LBBD) methods. The results originate from an LBBD implementation constructing maximal feasible solutions (MFS), an LBBD implementation constructing irreducible feasibility cuts (IFC) and an implementation consisting of the fusion of the two implementations constructing both IFCs and MFSs. The IFC implementation is mostly used as a reference, while the two methods constructing MFSs are part of our contribution.

First, the computational environment is described. It contains information about the instances and how they were solved. Second, a brief description of the test setup is given. Third, we present the computational results from the tests in different box plots with some comments. Further, the results from a small sample group of instances are presented to visualise some example behaviours of the LBBD methods. Last, we present some instances on which the LBBD implementation performed better than when applying a constraint programming solver directly on the complete problem.

6.1 Computational environment

The LBBD method has been implemented using Python 3.8. The MIP model was solved using Gurobi Optimizer version 9.5.1 and the CP model was solved using IBM ILOG CP Optimizer 2.23. All tests were carried out at the National Supercomputer Centre (NCS), at Linköping University, on a computer with two Intel Xeon Gold 6130 processors, with 16 cores and 2.1 GHz each, and 96 GB RAM. Each instance was given a time of 4 hours and the MIP-gaps were set to $10^{-6}$ for each master problem.
6.1.1 Instances

All the instances tested are taken from previous work. There were 384 problem instances, with various difficulty, available for testing. The instances are marked easy, medium or hard if a complete CP-solver using IBM ILOG CP Optimizer could solve the problem in less than five minutes, between five minutes and 24 hours or was unable to solve it in 24 hours, respectively. Information about the instances is given in [11] and all the instances can be accessed directly from the repository.

The instances difficulty distribution is presented in Table 6.1.

<table>
<thead>
<tr>
<th>Information</th>
<th>Difficulty</th>
<th>Easy</th>
<th>Medium</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of instances</td>
<td></td>
<td>207</td>
<td>10</td>
<td>167</td>
</tr>
<tr>
<td>Percentages (%)</td>
<td></td>
<td>53.9</td>
<td>2.6</td>
<td>43.5</td>
</tr>
<tr>
<td>Average no. of tasks</td>
<td></td>
<td>6,950</td>
<td>8,419</td>
<td>6,843</td>
</tr>
<tr>
<td>Average no. of pos. time lags</td>
<td></td>
<td>12,720</td>
<td>14,834</td>
<td>12,548</td>
</tr>
<tr>
<td>Average no. of exact time lags</td>
<td></td>
<td>1,139</td>
<td>1,280</td>
<td>1,166</td>
</tr>
<tr>
<td>Average no. of processors</td>
<td></td>
<td>2.81</td>
<td>2.70</td>
<td>2.77</td>
</tr>
</tbody>
</table>

6.2 Test setup

A problem formulation for the instances is given in Chapter 4. To solve the instances, the LBBD scheme model is implemented as described in Chapter 5. Three different types of LBBD schemes were tested. Pseudo-code for the three implemented LBBD schemes are given in Algorithms 6, 8 and 10, respectively. Below, we talk about pessimistic bounds on the objective instead of MFSs, to measure the strength of the solutions.

**Remark.** MFSs can be used to calculate pessimistic bounds.

**MDFBS** - An LBBD scheme that uses the DFBS algorithm for MFS to find feasible solutions that can be used to obtain pessimistic bounds. It also provides strengthened feasibility cuts. The pseudo-code of the DFBS algorithm for MFS is given in Algorithm 9.

**IDFBS** - An LBBD scheme that uses the enhanced DFBS algorithm for IFC, described in Section 3.6, to obtain IFCs and pessimistic bounds. The pseudo-code for the DFBS algorithm for IFC is given in Algorithm 7.

1https://gitlab.liu.se/elirol5/multiprocessor_scheduling_inst
MIDFBS - An LBBD scheme that uses a combination of the DFBS algorithm for MFS and the enhanced DFBS algorithm for IFC. The DFBS algorithm for MFS is used to obtain pessimistic bounds and strengthened feasibility cuts, and the DFBS algorithm for IFC is then used to obtain IFCs from the strengthened feasibility cuts.

Originally, MDFBS and IDFBS only provided pessimistic bounds and IFCs, respectively. However, to benchmark the results to one another, we modified MDFBS and IDFBS, without loss of computational capacity, to produce strengthened feasibility cuts and pessimistic bounds, respectively. This could be done since when searching for an MFS, one will find proof of infeasibility and similarly when searching for an IFC, one will find feasible solutions. This information can be used to construct feasibility cuts stronger than the standard cut and to obtain pessimistic bounds on the objective. Though, there are no guarantees on the quality of the strengthened feasibility cuts or the pessimistic bounds in these cases.

Two tests were performed, in which we tested each LBBD scheme separately for different selections of instances. Due to lack of computational time available at NSC, only 83 out of the 384 available instances were tested.

For the first test, referred to as TEST-A, we tested 50 instances with difficulty distributed as described in Table 6.2. This specific distribution was chosen to represent the entire instance set. Though, note that the instances of medium difficulty are over-represented, percentage-wise. This is because we did not want a too narrow sample size of medium difficulty instances to make the results unreliable. For each respective difficulty, the instances were chosen at random. A list of the instance names in this test set is given in Appendix B.1.

<table>
<thead>
<tr>
<th>Information</th>
<th>Difficulty</th>
<th>Easy</th>
<th>Medium</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of instances</td>
<td>22</td>
<td>6</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Percentages (%)</td>
<td>44.0</td>
<td>12.0</td>
<td>44.0</td>
<td></td>
</tr>
</tbody>
</table>

It was observed in the first test that some of the 50 instances did not provide interesting results since they did not complete any iterations. To broaden results regarding the MDFBS we performed a second test, referred to as TEST-B, testing 33 other instances from the instance set. These instances were chosen by running all available instances using the MDFBS for 7 minutes. Instances that performed at least one LBBD iteration and did not find an optimal solution were included in this test set. The distribution between the difficulties for the
instances is given in Table 6.3. Note that no instances of medium difficulty were included in the set of tested instances. A list of the instance names in this test set is given in Appendix B.2.

<table>
<thead>
<tr>
<th>Information</th>
<th>Difficulty</th>
<th>Easy</th>
<th>Medium</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of instances</td>
<td></td>
<td>22</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>Percentages (%)</td>
<td></td>
<td>66.7</td>
<td>0</td>
<td>33.3</td>
</tr>
</tbody>
</table>

In the computational evaluation of the performance of each LBBD scheme, each instance in each test was given a total time of 4 hours. Some instances were solved within this time. To get an overview of the performance, we present the size of the gap between the optimistic bound and the pessimistic bound in each iteration, the size of the feasibility cut in each iteration, the total number of master problem iterations and the number of subproblems performed in each iteration. This is done for each LBBD scheme, MDFBS, IDFBS and MIDFBS, for each instance difficulty. The results of our computational evaluation are presented in Figures 6.1 and 6.2 for TEST-A and TEST-B, respectively. In each subplot, the horizontal axis states the LBBD scheme and the instances’ difficulty. The vertical axes are of logarithmic scale. The settings for the box plots follow below.

- The red central mark indicates the median.
- The blue bottom and top edges indicate the 25th and 75th percentiles, respectively.
- The red dots indicate outliers.
- The black line indicates a whisker that is stretched to the most extreme data points, not considered to be outliers.

**Remark.** Data is obtained after each LBBD iteration, meaning that if a test is timed out before an iteration is completed, the data from that iteration will not be included in the results. Also, since the pessimistic bounds obtained from the LBBD iterations are not monotone, instead the best found pessimistic bound is kept and used in the plots.
6.3 Solving the instances

We start by studying the results from TEST-A, presented in Figure 6.1. In Figure 6.1a, it is clear that MDFBS and MIDFBS produced strong bounds on the objective compared to IDFBS. MDFBS and MIDFBS provided pessimistic bounds that yielded gaps of order $10^0$, while IDFBS yielded gaps of order $10^3$. Figure 6.1b illustrates that the cuts generated in IDFBS and MIDFBS are strong compared to the ones obtained from MDFBS. IDFBS and MIDFBS provided cuts of order $10^0$, while IDFBS yielded cuts of order $10^3$. IDFBS is observed to have a higher deviation than the MIDFBS, with some outliers of order $10^1$.

Figures 6.1c–6.1d give the number of solved master problems and subproblems, respectively. For the former, it is observed that IDFBS generally solved more master problems than MDFBS and MIDFBS while MDFBS generally solved marginally more master problems than MIDFBS. As seen in the latter, this is directly correlated to the number of subproblems solved in each iteration. IDFBS generally solved the fewest subproblems in each iteration, but with the highest deviation. MDFBS solved fewer subproblems than MIDFBS. Overall, the instances difficulties do not seem to affect the computational performance of the LBBD schemes.

We also study the results from TEST-B, presented in Figure 6.2. As mentioned earlier, these instances were specifically chosen to obtain more data by ensuring that more instances perform iterations. We compare the results given in Figure 6.1 with the results given in Figure 6.2, to investigate whether they are a good representation of how the LBBD schemes perform overall. The results are more often than not similar. However, it differs somewhat for the deviation in master problem iterations performed, where TEST-B performed more consistently but with more extreme outliers. Also, TEST-B typically provided stronger gaps for instances of difficulty easy, when MIDFBS and MDFBS were used.

6.4 Comments on the results

This section discusses the observations from Figures 6.1–6.2. The main observation is that the results are very similar for the two different samples of instances. We can therefore conclude that the results obtained from TEST-A are a good representation of the overall performance for the methods, despite the absence of data. We also observe that there are no obvious differences in quality of the gaps and cuts between instances of different difficulties.

The results show that MDFBS provides strong gaps, while IDFBS provides strong feasibility cuts. The MIDFBS algorithm produces both strong gaps and
Figure 6.1: Results from TEST-A. (a) gap sizes, (b) cut sizes, (c) number of master problem iterations and (d) number of subproblem iterations.
6.4. Comments on the results

Figure 6.2: Results from TEST-B. (a) gap sizes, (b) cut sizes, (c) number of master problem iterations and (d) number of subproblem iterations
strong cuts, with slightly more subproblems solved. However, it would be logical to assume that the number of subproblems performed in MIDFBS would be approximately equal to the sum of subproblems performed in MDFBS and IDFBS. A reason why this is not completely true is that when calculating a pessimistic bound, we obtain a strengthened feasibility cut which is then used as a seed when calculating an irreducible cut. Thus, depending on the strength of the seed, the LBBD scheme avoids solving a number of subproblems.

We see that the gaps obtained by IDFBS have a high deviation. As mentioned, this method has no guarantees to create good pessimistic bounds and since these bounds are not monotone, the best one is kept. If a low pessimistic bound is found early, this is kept for the remaining iterations and can therefore be a significant part of the data collected for that method.

We observe that IDFBS generally performs more master problem iterations than the other LBBD schemes. This can be explained partly by the fact that IDFBS typically performed fewer subproblem iterations. Another reason for this behaviour can originate from how we construct the bounds and cuts. When constructing a pessimistic bound, the subproblems become increasingly more restrictive, while when constructing a feasibility cut, the subproblems become decreasingly less restrictive. This could affect the computational time and thus contribute to the number of master problems that each LBBD scheme performed.

*Remark.* A subproblems’ restrictiveness is based on the number of tasks and number of time lags included.

We noted that some instances had very simple subproblems. If these did not terminate due to an optimal solution being found, many master problem iterations were performed due to the easily solved subproblems. This was mainly a problem for MDFBS since its convergence was slow. This behaviour is especially visualised by its extreme outliers in Figure 6.2c for instances of easy difficulty. In general, the instances were either solved or timed out within a few iterations. Hence the low number of master problem iterations.

### 6.5 Samples of tested instances

In this section, we give four sample plots that in a good way visualise different behaviours on the optimistic and pessimistic bounds that we encountered in our testing when using MDFBS and IDFBS. The sample results are presented in Figure 6.3. MDFBS was used in Figures 6.3a–6.3c, while IDFBS was used in Figure 6.3d. The vertical axes in the plots indicate the number of non-scheduled tasks for a solution. It is easily observed that the pessimistic bounds are not monotone. The results in Figures 6.3c–6.3d are obtained from the same instance.
Figure 6.3: Optimistic and pessimistic bounds for some samples of the tested instances

using MDFBS and IDFBS, respectively. From these figures it is observed that MDFBS provides stronger pessimistic bounds, overall and more consistently.

6.6 Solutions to the instance set

In [11], where the instance set first was introduced, a complete constraint programming solver was used for 24 hours to determine whether an instance had a feasible solution or not. The instances marked with difficulty hard could not be solved within these 24 hours using a single CP model. An instance is here considered solved if a solution that schedules each task is found or if it could
be proven that no such solution exists. In our model, this means that we either
must solve an instance to optimality with objective value zero or prove that
the optimal objective value is greater than zero. By using our IDFBS on the
instances marked with difficulty hard for 4 hours, we managed to solve some
instances that had not previously been solved. IDFBS solved 106 out of the
total 167 instances marked as hard, which is 63.5%.
Chapter 7

Concluding remarks

The purpose of this master thesis was to investigate whether maximal feasible subset of constraints (MaFS) could be used to accelerate a logic-based Benders decomposition (LBBD) scheme for a multiprocessor scheduling problem. In Chapter 3, we showed that if a part of a master problem solution is feasible in the subproblem, it is also a feasible solution to the original problem. We also present the concept of a maximal feasible solution (MFS) to measure the strength of a solution. Such solutions are found by using algorithms for finding MaFS. We developed an LBBD scheme that constructs an MFS in each iteration to provide a pessimistic bound on the objective. An LBBD scheme, using algorithms designed for finding an irreducible infeasible subset of constraints (IIS) to construct irreducible feasibility cuts (IFC) to improve the convergence, was also presented. Further, we developed a method that finds both MFSs and IFCs, to provide pessimistic bounds and speed up the convergence, respectively.

When testing these methods on a set of instances, the results show that an LBBD scheme that produces MFSs provides strong pessimistic bounds on the objective, unlike other acceleration methods. In each iteration, we can enclose the optimal objective value in a small interval. However, if an optimal objective value is considered close, an optimal solution can still be far from a practical perspective in terms of computational time and master problem iterations. Therefore, acceleration methods that speed up the convergence should also be applied to the scheme. To address this challenge, we developed the method that finds both MFSs and IFCs. This method showed potential by complementing the other algorithms’ weaknesses.
7.1 Initial vision

The primary intention when using the concept of MaFS to accelerate an LBBD scheme was to use pessimistic bounds to terminate earlier. This would however require an objective value that increases in at least some iteration, i.e., the optimistic bound increases. If a pessimistic bound was found in an iteration and in a later iteration the objective value given from the master problem was equal to that value, the scheme could terminate instantly since a solution with that objective value had already been found. To visualise the desired behaviour we present a small example.

Example 7.1.1. Let the objective value given from the master problem in the LBBD scheme in each iteration and the pessimistic bound constructed in each iteration be presented in Table 7.1.

Table 7.1: Objective value and pessimistic bound in each iteration

<table>
<thead>
<tr>
<th>Iteration</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective value</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Pessimistic bound</td>
<td>9</td>
<td>11</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>Best pes. bound</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

Let an optimal solution be found in iteration 13 with objective value 4. Since a pessimistic bound of 4 on the objective value is found already in iteration 7 the process could have been terminated instantly in iteration 9 when the objective value is 4. Thus, we have already found a feasible schedule that schedules all tasks but 4. That would save time not having to solve the later master problems and their corresponding subproblems.

There were two major reasons why this could not be exploited fully. The first reason was due to the instances. Many instances had an optimal objective value equal to zero, i.e., all tasks could be scheduled. Then there is not possible to find a pessimistic bound that could be an optimal solution since the pessimistic bound constructed in each solution has to be strictly greater than the objective value in the corresponding iteration. The second reason was that the master problem formulation was strong, possibly because of the very effective subproblem relaxations made. For instances that did not have objective value zero, the master problem found the best objective value in the first iteration, even though it often took many iterations to find an assignment for which the subproblem was feasible. This is a problem for the same reason as before, we need the objective value to increase in order to terminate early.
7.2 Future work

Since we were able to derive a method for finding MFSs to a given master problem assignment, the concept of MaFS in an LBBD is considered promising. However, a drawback with the method was that the construction of these solutions was computationally expensive. Based on this, it is of interest to find faster and more sustainable ways to generate MFSs. It would for example be relevant to explore the possibilities to exploit the problem structure to construct MFSs more efficiently.

Another promising idea for the future is to examine the performance of heuristic methods for constructing strong feasible solutions. These solutions may not always yield the strongest pessimistic bounds, but they might be computationally much cheaper to construct than using algorithms for finding an MFS. The difference in computational cost between the methods might outweigh the difference in quality on the pessimistic bounds.

We initially considered implementing the MARCO algorithm, presented in Section 2.2, to enumerate all MaFSs. This would provide the maximum feasible subset of constraints (MuFS). In the context of LBBD, this means that all MFSs are constructed and the strongest possible pessimistic bound would be found in each iteration. However, since MARCO relies on an algorithm for finding an MaFSs, which when using our implementation was expensive enough to find, we never bothered enumerating all of them. However, if a more efficient way of finding an MFSs is found, implementing the MARCO algorithm would possibly be relevant.

It could also be of interest to test the method on an implementation with a weaker subproblem relaxation. This would possibly yield an objective value behaving like the one described in Section 7.1 and might perhaps also provide an easier master problem.
Bibliography


Appendix A

Constraint programming

A.1 Interval variables and constraints

To solve the constraint programming model, $y_i$ is represented as an interval variable with a start time and an end time for each $i \in I_h$ for $h \in H$. The variable $y_i$ is connected to a release time $r_i \in \mathbb{Z}$ and a deadline $d_i \in \mathbb{Z}$. The domain of $y_i$ is $[r_i, d_i]$. Each interval variable $y_i$ also has a processing time $p_i \in \mathbb{Z}$. The decision of the variable is to determine its start time $\text{start}(y_i)$ and the end time will be given as $p_i + \text{start}(y_i)$.

The Disjunctive constraint is defined for an array of the interval variables corresponding to the same processor $(y_1, \ldots, y_n)$ and an array of processing times $(p_1, \ldots, p_n)$. The interval variable $y_i, i = 1, \ldots, n$, is active in the constraint if $\text{start}(y_i) \leq t \leq \text{start}(y_i) + p_i$ for each $t \in \mathbb{Z}$. The STARTAtSTART constraint takes as input two interval variables $y_i, y_j$ and an integer $z \in \mathbb{Z}$. The constraint is active if

$$\text{start}(y_i) + z = \text{start}(y_j)$$

(A.1)

and the STARTBeforeSTART takes the same input but the constraint is active if

$$\text{start}(y_i) + z \leq \text{start}(y_j).$$

(A.2)
Appendix B

Tested instances

B.1 Instances in TEST-A

- 22_371_12_InstanceD28_cmid0_hard
- 19_266_4_InstanceD16_cmid0_medium
- 19_265_4_InstanceD16_cmid0_medium
- 17_225_38_InstanceD14_cmid0_hard
- 4_013_3_InstanceD12_cmid0_easy
- 21_306_4_InstanceD27_cmid1_easy
- 9_077_28_InstanceD14_cmid0_hard
- 2_006_1_InstanceB26_cmid1_hard
- 21_310_4_InstanceD27_cmid1_easy
- 19_287_4_InstanceD16_cmid0_medium
- 15_192_36_InstanceD14_cmid0_hard
- 22_351_12_InstanceD28_cmid0_easy
- 19_264_4_InstanceD16_cmid0_medium
- 13_162_33_InstanceD14_cmid0_hard
- 21_339_4_InstanceD27_cmid1_easy
• 4_034_3_InstanceD12_cmid0_hard
• 13_159_33_InstanceD14_cmid0_hard
• 22_360_12_InstanceD28_cmid0_easy
• 22_358_12_InstanceD28_cmid0_easy
• 16_205_37_InstanceD14_cmid0_easy
• 5_037_3_InstanceD14_cmid5_hard
• 19_251_4_InstanceD16_cmid0_easy
• 21_331_4_InstanceD27_cmid1_hard
• 16_201_37_InstanceD14_cmid0_easy
• 19_260_4_InstanceD16_cmid0_easy
• 1_003_1_InstanceA7_cmid0_hard
• 19_245_4_InstanceD16_cmid0_easy
• 22_352_12_InstanceD28_cmid0_easy
• 19_275_4_InstanceD16_cmid0_hard
• 17_228_38_InstanceD14_cmid0_hard
• 4_021_3_InstanceD12_cmid0_easy
• 4_029_3_InstanceD12_cmid0_hard
• 19_289_4_InstanceD16_cmid0_hard
• 13_156_33_InstanceD14_cmid0_hard
• 4_018_3_InstanceD12_cmid0_easy
• 19_292_4_InstanceD16_cmid0_easy
• 19_277_4_InstanceD16_cmid0_hard
• 4_030_3_InstanceD12_cmid0_hard
• 16_208_37_InstanceD14_cmid0_easy
• 16_211_37_InstanceD14_cmid0_easy
B.2 Instances in TEST-B

- 18_232_1_InstanceD16_cmid0_easy
- 4_011_3_InstanceD12_cmid0_easy
- 19_238_4_InstanceD16_cmid0_easy
- 4_026_3_InstanceD12_cmid0_medium
- 19_268_4_InstanceD16_cmid0_hard
- 19_282_4_InstanceD16_cmid0_hard
- 19_272_4_InstanceD16_cmid0_hard
- 21_317_4_InstanceD27_cmid1_easy
- 1_002_1_InstanceA7_cmid0_easy
- 4_025_3_InstanceD12_cmid0_medium

B.2 Instances in TEST-B

- 17_216_38_InstanceD14_cmid0_easy
- 15_187_36_InstanceD14_cmid0_easy
- 15_183_36_InstanceD14_cmid0_easy
- 19_257_4_InstanceD16_cmid0_easy
- 17_217_38_InstanceD14_cmid0_easy
- 17_215_38_InstanceD14_cmid0_easy
- 14_166_35_InstanceD14_cmid0_easy
- 22_368_12_InstanceD28_cmid0_easy
- 21_315_4_InstanceD27_cmid1_easy
- 13_163_33_InstanceD14_cmid0_hard
- 22_350_12_InstanceD28_cmid0_easy
- 9_078_28_InstanceD14_cmid0_hard
- 14_181_35_InstanceD14_cmid0_hard
• 17_227_38_InstanceD14_cmid0_hard
• 18_233_1_InstanceD16_cmid0_easy
• 20_299_1_InstanceD18_cmid6_easy
• 15_182_36_InstanceD14_cmid0_easy
• 22_349_12_InstanceD28_cmid0_easy
• 2_005_1_InstanceB26_cmid1_easy
• 21_322_4_InstanceD27_cmid1_easy
• 22_374_12_InstanceD28_cmid0_hard
• 8_065_17_InstanceD14_cmid0_hard
• 17_220_38_InstanceD14_cmid0_hard
• 14_176_35_InstanceD14_cmid0_hard
• 1_001_1_InstanceA7_cmid0_easy
• 22_376_12_InstanceD28_cmid0_hard
• 15_189_36_InstanceD14_cmid0_easy
• 9_075_28_InstanceD14_cmid0_easy
• 16_207_37_InstanceD14_cmid0_easy
• 7_056_14_InstanceD14_cmid0_hard
• 7_052_14_InstanceD14_cmid0_easy
• 6_046_6_InstanceD14_cmid0_hard
• 2_004_1_InstanceB26_cmid1_easy