Logic-Based Benders Decomposition applied to the Setup Assembly Line Balancing and Scheduling Problem

By

Kenneth D. Young

Supervisor: Dr. Alysson M. Costa

A thesis submitted to the University of Melbourne
for the degree of Master of Science (Mathematics and Statistics)
School of Mathematics and Statistics
August 2017

Typeset in LaTeX 2$_\varepsilon$.

Except where acknowledged in the customary manner, the material presented in this thesis is, to the best of my knowledge, original and has not been submitted in whole or part for a degree in any university.

Kenneth Young

Acknowledgements

This thesis is a result of almost two years of research undertaken at the University of Melbourne. As my first major step into the world of academic research, I can say that this project has been extremely challenging, whilst also tremendously rewarding. During the final semester of the research project, my life was filled with modelling, coding, debugging, more debugging and writing. Each of which was thoroughly enjoyable.

All the wonderful people which I was surrounded by deserve a non-trivial share of the credit. Their help and encouragement throughout the project has been invaluable, and for that I am grateful.

First and foremost, I must thank the one who deserves it the most, Alysson Costa, my supervisor. Without his guidance, this project may have taken numerous wrong turns along the way. Alysson has previously studied both the problem I tackled and the method I have used, and so his familiarity with the areas provided a much needed level of understanding in the early days of the project. I really appreciate the theoretical contributions that Alysson has made, but possibly more importantly, I truly appreciate his kindness. He has an uncanny ability to find the humour in life and seems to always have an extra smile to offer when one is needed. It has been a pleasure to work with the most enthusiastic Operations Researcher I've met.

My friends, both old and new, have filled these days of my life with jokes, pranks and joy; and I cannot thank them enough for it. I wish to thank Ai, Anupama, Ali, Ria, Lotte, Mark, Mim, Shian, John, Curtis, Heather, Scott, Lachlan, Georgia, Ben, Constance and Vincent. In particular, Ria's help and friendship over the whole course of my Masters needs a special mention. Working together on the group projects during our course has been nothing but a pleasure. Her drive to become the very best she can — at anything she puts her mind to — has been contagious. Even in this thesis, her contributions to extending the formulations of the CP subproblems were pivotal in their successful performance.

I wish to express my gratitude to my sister, Margaret. Miraculously, all the house work that I was unable to assist with this semester disappeared without request or mention: for this I am thankful. She provided an immense amount of help by taking the time out of her life to assist me in the proof reading process. I thank Margaret for ensuring the reader does not need to endure a complete butchering of the English

language.

The fellow members of my research group have also provided wonderful feedback during the course of this project. Without fail, they have been able to find new fruitful directions to take the project, whilst also steering me away from the directions not filled with quite so much fruit. I wish to thank Landir, Allen, Cheng, Gala, Ashwani, Simran, Pamela and Chathranee.

The entire experience has been terrific.

And I would like to disacknowledge my mortal enemy, Anupama Pilbrow.

Abstract

Balancing the workload of an assembly line leads to a decision problem where the aim is to assign a set of tasks to a sequence of work stations, whilst optimising a measure of performance, *e.g.*, maximising the production rate. The SetUp Assembly Line Balancing and Scheduling Problem (SUALBSP) is a recent variant on the classical Assembly Line Balancing Problem (ALBP). This new problem introduces sequence-dependent setup times between any pair of consecutive tasks processed within a work station's task sequence. Considering these setup times leads an additional layer of complexity as the problem now involves a set of scheduling decisions for each station along the production line.

Benders decomposition naturally suggests itself as an appropriate approach for the SUALBSP as we can divide the problem's decisions into two distinct sets. We propose a general logic-based Benders decomposition framework which leads to the assignment portion being decided by a relaxed master problem and a number of sub-problems which together handle the scheduling portion. Each sub-problem is highly combinatorial in nature and so we consider state-of-the-art solving technology from both Operations Research and Artificial Intelligence to solve the scheduling problem efficiently. A range of Benders cuts are devised and their practical effectiveness is one of our primary contributions to the literature.

We test the best-known exact methods to solve the SUALBSP and use these models as a benchmark against which to scrutinise our hybrid solution methodology. In total, 396 instances of the problem were tested and our Benders decomposition approach vastly outperformed the best models from the literature. Due to this variant of the ALBP being notably under-studied, we conclude the paper by remarking on a range of possible future directions which the research community could investigate.

Contents

List of Figures

List of Tables

—Everything should be made as simple as possible, but not simpler.

Albert Einstein

1 Introduction

Assembly lines are flow-based production systems widely used by the manufacturing industry. Designing these systems to meet the requirements set by the industry gives rise to the Assembly Line Balancing Problem (ALBP). An assembly line problem consists of a sequence of *stations* and a set of *tasks*. Stations are fixed points along the assembly line which are operated by either human personnel or autonomous machinery. A workpiece is launched down the line and a subset of the tasks are performed on it at each station. Once all required tasks are finished, the completed product exits the assembly line. The workload of tasks must be divided among the stations to optimize some performance measure, while also respecting physical restrictions. A standard example of such a restriction would be a precedence relation, which can require one task to be completed before another can begin processing on the product. The line itself is a mechanism used to transport the products between stations; most commonly the line is a conveyor belt.

There are two common objectives when designing an assembly line, which each lead a distinct problem. The first is called the type-1 problem and is concerned with the design of a new assembly line where the production rate of the product is predefined and the aim is to construct the line with as few stations as possible, in-order to meet this demand. The second problem, called type-2, considers the redesign of an existing assembly line where the number of stations is fixed and the rate of production is optimized. The second case can arise when a manufacturing company alters their production process or changes the products they offer. Our thesis focuses on solving the type-2 problem.

The *cycle time* of an assembly line defines the time each station is given to perform its assigned tasks. As such, finding the minimal possible cycle time is the primary aim when optimizing the production rate of an assembly line. Since all

stations are connected by a single line, the station with the largest workload will define the overall cycle time of the line. To achieve the minimal cycle time value, we aim to balance the workload of the tasks as evenly as possible among the stations.

The structure of the paper is as follows. The remainder of Chapter 1 provides illustrative examples of the problem and closes with some motivation. In Chapter 2 we detail a selection of the relevant theory for our problem, spanning the fields of Operations Research and Artificial Intelligence. Chapter 3 presents the mixed-integer programs which were used to compare our solution methodology against. The construction of our logic-based Benders decomposition is presented in Chapter 4, together with many possible modelling choices. The results of our computational experiments are given in Chapter 5. We conclude in Chapter 6 with some remarks on the performance of our method and finally note a range of possible directions for future research on this topic.

1.1 Preliminaries and Examples

The fundamental problem in this area is called the Simple ALBP, or SALBP; we denote the type-2 version by SALBP-2. We now detail some straightforward examples to illustrate how the workload of an assembly line can be balanced. In Example 1 we provide an instance of the SALBP-2 with a possible feasible solution and the optimal solution given in Figures 1.2 and 1.3 respectively. Note, that t_i denotes the processing time of task i and c denotes the cycle time of the assembly line.

Example 1. Consider an instance of the problem with four tasks $T_1, \ldots, T_4$, and three stations $S_1, \ldots, S_3$, along the assembly line. The aim is to find an assignment of the tasks to the stations which minimizes the cycle time. This assignment must respect the precedence relations listed in Figure 1.1, *e.g.*, task T_1 must be completed before task T_2. The processing time of each task is included in Figure 1.1 next to each task.

A feasible solution to this problem is given in Figure 1.2, where the stations are spaced along the horizontal axis. Each station's workload begins at the horizontal axis and proceeds upward through time until all assigned tasks are completed. In this solution, the workload of S_3 is the largest and as such defines the cycle time as $t_3 + t_4 = 2 + 9 = 11$.

This feasible solution can be improved by moving task T_3 from station S_3 to S_2. For this assignment to remain feasible, T_3 must be processed after T_2, due to the precedence relations. This new assignment leads to the optimal solution listed in Figure 1.3 with a cycle time of nine. □

The variant of the ALBP which we are concerned with adds the consideration of sequence-dependent setup times between consecutive tasks within a station's workload. Theoretical approaches to the ALBP usually assume that the assembly line workers can decide on an arbitrary precedence-feasible sequence to execute their

Figure 1.1: Precedence graph

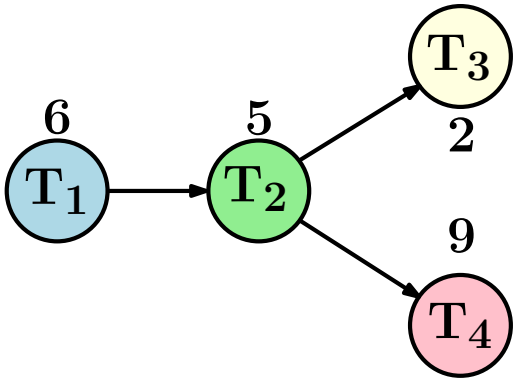


Figure 1.2: Feasible solution

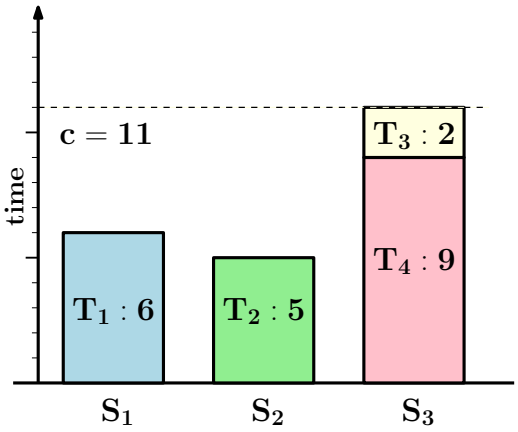
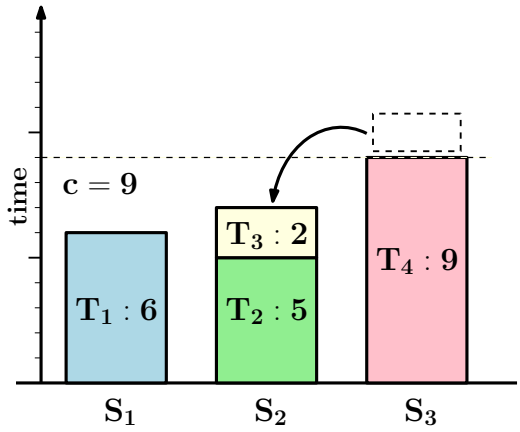


Figure 1.3: Optimal solution

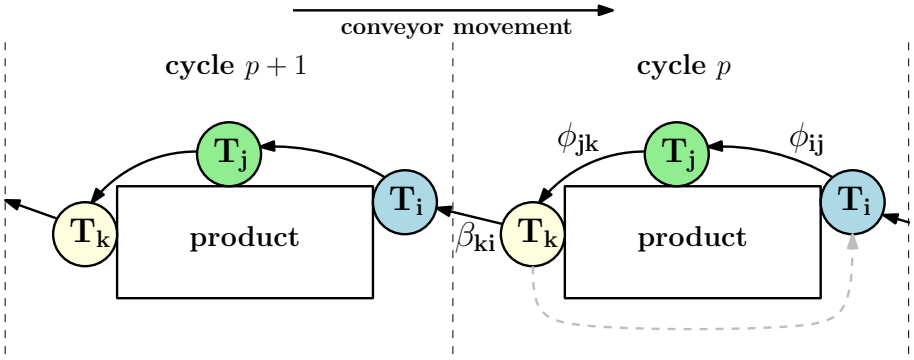


assigned tasks, which will not affect the station's total processing time. However, in practice there can be non-trivial setup costs, due to walking times or tool changes, which can account for a considerable amount of a station's processing time. Adding this consideration to the problem leads to a scheduling problem arising within each station. This variant of the problem is called the SetUp Assembly Line Balancing and Scheduling Problem (SUALBSP).

To realistically model the setup costs that occur in assembly lines, two types of setups are introduced; *forward* and *backward* setups. A forward setup time, denoted by ϕ, occurs between two consecutive tasks within a station's task sequence if both tasks are performed on the same product. Whereas a backward setup time, denoted by β, occurs between the last task performed on a product and the first task performed on the next product along the line.

To see how these setups are differentiated, Figure 1.4 depicts the sequence of tasks T_i, T_j, T_k which are performed in a cyclic manner on consecutive products. The execution of tasks proceeds chronologically from right to left, *i.e.*, T_1 is the first task performed in cycle p. Forward setups arise between consecutive tasks

Figure 1.4: Cyclic task sequence of a station (adapted from [54])



operating on the same product. A backward setup cost is incurred between the last task of cycle p and the first task of the next cycle, $p+1$. Each cycle begins at time zero with the execution of task T_i taking t_i time units to process. Then after performing the forward setups ϕ_{ij} and ϕ_{jk} together with processing times t_j and t_k, the station operator must perform the necessary backward setup operation before moving to the next workpiece along the line. Thus, when including this backward setup time β_{ki}, the following condition must be satisfied to have a feasible cycle time $t_i + \phi_{ij} + t_j + \phi_{jk} + t_k + \beta_{ki} \leq c$. In Figure 1.4, the gray arrow from T_k to T_i implies the cyclic nature of the sequence.

Example 2. Again consider the instance from Example 1, but now with sequence-dependent setup times between tasks. The arrays of forward and backward setup costs can be found in Tables 1.1 and 1.2 respectively. Note, that some of the entries in these arrays are omitted as the corresponding sequence of tasks is not possible due to the precedence relations or logical restrictions.

The previous optimal solution is amended to include the required setup costs and is given in Figure 1.5. Note that the setup cost of any task to itself is defined as zero. To reach the optimal solution we must now consider the sequence of tasks within each station.

By moving task T_3 to station S_3 and considering the sequencing of T_3 and T_4, we find the optimal solution to this problem, given in Figure 1.6. The optimal cycle time is 13 in this case. □

1.2 Motivation

In this section we provide motivation for why the ALBP with setup costs is an important problem to be solved and why the approach we chose to utilize is well-suited to the problem.

Table 1.1: Forward setup times

ϕ	T_1	T_2	T_3	T_4
T_1	–	3	3	3
T_2	–	–	2	3
T_3	–	–	–	1
T_4	–	–	2	–

Table 1.2: Backward setup times

β	T_1	T_2	T_3	T_4
T_1	0	–	–	–
T_2	3	0	–	–
T_3	3	5	0	3
T_4	3	3	1	0

Figure 1.5: Feasible solution with setup

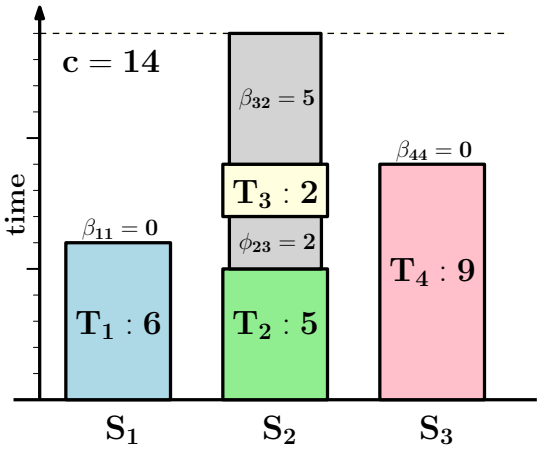
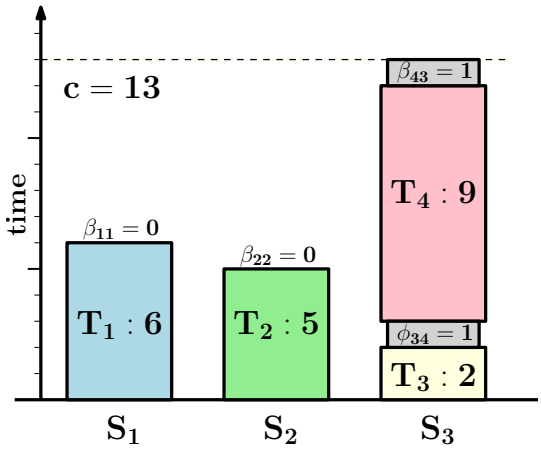


Figure 1.6: Optimal solution with setup



1.2.1 Why the SUALBSP?

Assembly lines were originally designed for the production of a single type of product in high volumes. Although, assembly lines able to produce only a single product are not suitable for consumer-centric markets, where there is a need to tailor the products more closely to the user's needs. Sequence-dependent setup times are commonly considered in job shop scheduling problems and a range of other production problems (*cf.* [2–4]). In those problems, the setup times arise because the products are typically assumed to be highly diverse, leading to additional configuration effort being required between any two tasks. In the context of operational planning of assembly lines, sequence-dependent setup times have been sparsely considered by the literature. Scholl *et al.* (2013) list a number of possible practical settings where setup times between tasks should not be taken as negligible. Some include the following:

- In automotive assembly lines where large workpieces (vehicles) need to be constructed, work can be performed at numerous mounting positions on the item. The size of the car body can result in walking distances between mounting positions being non-trivial. Thus, the preparation time (setup) can be a critical aspect when sequencing the required tasks.

- Fixed material containers can be placed along the moving conveyor system to allow workers to collect parts or tools needed. Further to walking times between these containers and the workpiece, the time taken to retrieve what is required from the container must also be accounted for. At a major German car manufacturer, these times contributed $10 - 15\%$ of the total cycle time [54].

- Specific tools are often required to perform a task. If consecutive tasks need different tools for their execution a tool-change between tasks will be necessary. Robotic assemblers used to perform the tasks of a station can be built to be highly flexible in the types of tasks they can perform. However, this flexibility can mean that numerous tool changes are required to switch from one task to another in a sequence. Thus, these sequence-dependent setup costs from tool-changes are typical in robotic assembly lines.

Consequently, accurately modelling the setup costs incurred can be an important aspect when balancing an assembly line.

In practice, setup times are accounted for in simpler ways, such as using an approximation or incorporating the setup time into the task's processing time. Scholl *et al.* note that by using such approximations "the planning team need to strike a fragile balance between an underestimation of setups, which lead to infeasible line balances, and an overestimation of setups, which leads to an allocation of excessive resources." The procedure used to estimate the cost of setups can be time consuming and is prone to getting stuck in sub-optimal solutions.

For these reasons we feel that further research into the effect of setup times on the ALBP is of interest to the academic community and the manufacturing industry.

1.2.2 Why Benders Decomposition?

Benders decomposition is a well-known approach to optimization problems. It is naturally suited to problems where the decisions can be separated into two distinct sets. Advances in the past few decades to the theory of this method has broadened its applicability to a wider array of problems. In our case, a solution to the SUALBSP-2 can naturally be divided into the assignment portion, which assigns each task to a station, and the scheduling portion, which decides on the time that each task begins execution within a station.

We can view this separation of concerns in the following way. The manager of an assembly line needs a new item to be put into production and so assigns the required set of tasks to the stations along the line. When calculating the line's cycle time the manager only estimates the setup times within each station. Each station operator must then decide if a possible sequencing of their assigned tasks exists which can respect the manager's cycle time; or if a precedence-feasible sequence exists at all. If at least one worker cannot find a feasible task-sequence that respects the manager's

cycle time estimate, then a revision may need to be made to the assignment of tasks. The give-and-take between the assignment and scheduling portions continues until a mutually satisfactory solution is found. This informal process of communicating information between the station workers and the assembly line manager could be time consuming and not guarantee optimality.

The informal feedback loop between the two halves of the decision problem suggests that Benders decomposition could be employed to mimic this process. The decision making process can naturally be divided into a master problem, which assigns the tasks to the stations along the line, and several independent scheduling problems. In total, there is a sub-problem for each of the m stations, which needs to decide the exact execution time of all tasks that station has been assigned. Each sub-problem is similar to the asymmetric Travelling Salesperson Problem (TSP) with some forbidden paths due to the precedence relations. With this interpretation, the tasks are viewed as the cities and distances between them are the setup times. To complete the iterative loop, the sub-problems relay their information back to the master as Benders cuts and the assignment problem is re-optimized. In the words of J. N. Hooker, "the Benders cuts added to the master problem are the mathematical equivalent of telephone calls" from the station workers to the line manager [29].

The reader may be familiar with the classical version of the Benders decomposition method due to the work of Benders (1962) and Geoffrion (1972), however for the SUALBSP, this approach is inappropriate. Due to the sub-problems being highly-combinatorial discrete scheduling problems, formulating them as a linear or non-linear program will not be practical. We instead explore how the more general method of *logic-based* Benders decomposition can be used to solve the SUALBSP-2.

By employing this approach we are able to exploit the comparative advantages of multiple solving technologies to tackle each portion of the problem. Mixed-Integer Programs (MIPs) are well-suited to the assignment problem of the master and Constraint Programming (CP), from the Computer Science discipline, is an effective solving technology for scheduling problems.

1.3 Problem Definition

Here we present the core notation that will be used for the remainder of the thesis. Along the conveyor belt of the assembly line there are *work stations* $K = \{1, 2, \ldots, m\}$. Workpieces (or products/items) move down the conveyor belt from station to station. A workpiece remains at each station for one cycle, which has a duration given by the *cycle time*. The objective of the ALBP is to optimally partition the total work required among the stations with respect to a performance measure.

The work required to complete a single piece is separated into a set of *non-preemptive tasks* $V = \{1, 2, \ldots, n\}$, each with a discrete processing time t_i. Physical and technical conditions impose a set of *precedence relations* $E \subseteq V \times V$, which

prevent some non-allowed task orderings from occurring. If we consider the graph $G = (V, E)$ with tasks as the vertex set and directed edges defined by the precedence relations, then G is a directed acyclic graph (DAG). Without loss of generality, we may assume that the vertices are numbered topologically so that the following holds: $(i, j) \notin E$ if $i > j$.

Between each pair of tasks, discrete *forward* and *backward setup* times are predefined. We denote the forward setup time between i and j by ϕ_{ij} and the backward setup time by β_{ij}. When $i = j$, the forward setup time is undefined as a task can never follow itself in a station's forward work load. The backward setup β_{ii}, *i.e.*, the setup cost of a task to itself, is defined as zero.

To fully specify a feasible solution to the SUALBSP-2, one must give the following: an assignment of tasks to the stations and the execution time window of each task within its assigned station. From this specification, the cycle time is calculated by the total processing time of the station with the largest workload.

2

Theoretical Background

In this chapter we provide the theoretical background needed for the work presented in the following chapters. As such, we give summaries of the relevant literature from four fields spanning mathematics and computer science. These include background knowledge of the problem of assembly line balancing (Sect. 2.1); modelling and solving technology from the areas of linear programming (Sect. 2.2) and constraint programming (Sect. 2.3); and lastly we delve into some of the recent advances into the theory of Benders decomposition (Sect. 2.4).

2.1 Assembly Line Balancing

Optimizing the construction of an assembly line to maximize or minimize some performance measure has been a problem of great interest to the research community since its introduction by Henry Ford in 1915 [26]. Simply, it can be stated as a decision problem where we seek optimal partition of work to assemble a product. The work must be divided among discrete stations along a line (*e.g.*, conveyor belt), while respecting a series of restrictions imposed by physical limitations. All variants of the ALBP begin from this common starting point and add further restrictions or considerations which results in a range of generalizations. We refer the interested reader to one of the surveys of Baybars (1986), Becker and Scholl (2006) or Boysen *et al.* (2007) for a detailed snapshot of the problem's various extensions.

In this section we detail the basic version of the problem and the particular variant which is relevant to our work.

2.1.1 Simple Assembly Line Balancing Problem

The Simple Assembly Line Balancing Problem is the basic type of the ALBP [8, 53]. It consists of a serial production line which produces a single product. The tasks which need to be executed to achieve a product's completion all have fixed processing times and all stations along the line are uniform, *i.e.*, each is equivalently equipped and manned by homogeneous workers or machines.

There are two primary parameters of the assembly line which dictate the aim when solving the problem; these parameters are the *cycle time* and the number of stations, denoted by c and m respectively. We denote the *load* of station k by l_k, which is defined as the sum of all processing times of tasks assigned to station k. The cycle time of an assembly line is defined as follows,

$$c = \max\{\, l_k \mid k \in K \,\},$$

i.e., the largest load among all stations. Physically, the cycle time of an assembly line can be interpreted as how much time each station is given to work on a product, before the product moves to the next station. A lower cycle time means the conveyor belt can move faster and products are output at a higher rate.

From the parameters c and m, the ALBP can be formulated as three different but highly related problems: type-1, type-2 and type-E. We now summarize each variety and its particular qualities in turn,

- Type-1: Fixed c, variable m.
  The type-1 variety, abbreviated as SALBP-1 in the simple case, considers the cycle time to be fixed and the number of stations to be decided. This most commonly occurs when constructing a new assembly line. As an example, the SALBP-1 is characterized using the classification scheme of Boysen *et al.* (2007) by $[-|-|m]$.

- Type-2: Variable c, fixed m.
  For the type-2 variety, the aim is to find the optimal cycle time given that the number of stations along the line is fixed. In practice, this variety commonly corresponds to adapting an existing assembly line for a new purpose.

- Type-E: Variable c, variable m.
  This variety deals with the case when both c and m are considered to be decision variables of the problem. Here, the objective is to optimize some measurement of line efficiency.

2.1.2 Setup Assembly Line Balancing and Scheduling Problem

In this document, we are concerned with the SetUp Assembly Line Balancing and Scheduling Problem. This variant differs from the basic SALBP by introducing sequence-dependent setup times between pairs of tasks within a station's workload. This type of consideration was first made in the literature by Andrés *et al.* (2008) in an attempt to reduce the gap between the existing academic theory and the realities of assembly line problems posed by the industry. In the problem tackled by Andrés *et al.*, they considered a setup time to exist between a pair of consecutive tasks inside a station's sequence. As such, it needed to be added to that station's global processing time and so these setup times directly affected the cycle time of the line. The authors considered heuristic rules and employed a Greedy Randomized Adaptive Search Procedure (GRASP) to tackle the type-1 version of the problem. They also developed a lower-bound which was used to estimate the quality of their solutions found.

Later, some of the same authors continued this work in Martino and Pastor (2010), where over 50 priority-rules were considered for a heuristic procedure designed for the type-1 version. All results found for the problem at this point in the literature indicated that adding these sequencing considerations for the tasks made the problem much more difficult to solve. This is demonstrated by only very small instances of the problem being able to be optimally solved.

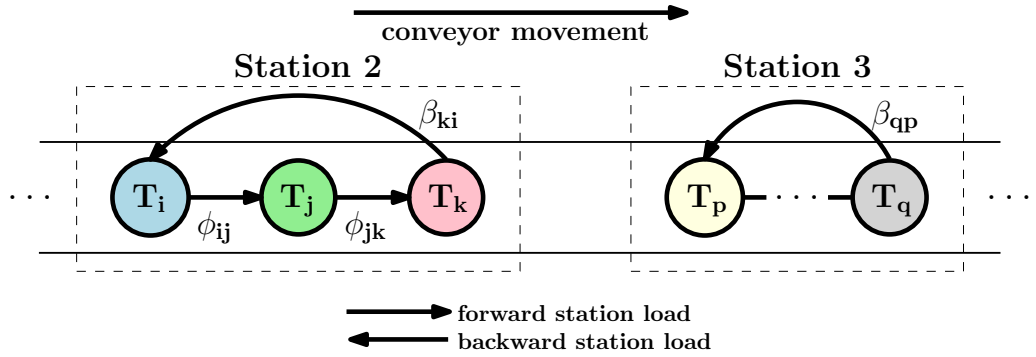
More recently, the work of Scholl *et al.* (2013) extended the previous work on this problem and formally named it the SUALBSP. The SUALBSP differs slightly from the problem tackled previously by introducing a distinguishment between *forward* and *backward* setup times. The motivation for this differentiation is so the theoretical problem can more accurately capture the physical aspects of sequencing tasks. A forward setup time occurs between two consecutive tasks assigned the same station, if each task is processing the same item. A backward setup time occurs between two consecutive tasks assigned the same station, if each task is operating on different items on the assembly line. Physically, a backward setup time captures the setup cost between the last task on the current product and the first task on the next product. We demonstrate this idea using a possible task sequence within two stations in Figure 2.1. Forward setup times are denoted by ϕ and backward setup times by β.

Scholl *et al.* (2013) proposed an exact solution approach by a mathematical formulation of the type-1 problem as a mixed integer program. Using the scheme of Boysen *et al.* (2007), the SUALBSP-1 can be formally classified as,

$$\text{SUALBSP-1: } [\Delta t_{dir}| - |m].$$

Further to their mathematical model, a range of tools were proposed which could be composed in a number of ways to create new heuristic approaches for the problem.

Figure 2.1: Setup times of a station's sequence



Esmaeilbeigi *et al.* (2016) give the most recent contribution of the literature to the study of the SUALBSP. In their paper, three new mixed-integer formulations are presented of the SUALBSP-1 which are based on different ways of representing the decisions. Their first two formulations are station-based and the last is a purely scheduling-based formulation which has abstracted the assignment decisions away from the problem. Each of the three models presented are also modified to be suited to the type-2 version of the SUALBSP. Although these models were presented, the type-2 problem was not the authors primary concern and as such the models were not tested.

In the literature there have been few attempts to tackle the type-2 variety of the SUALBSP. The first was by Yolmeh and Kianfar (2012) who use a Genetic Algorithm which is applicable to either the type-1 case or type-2 case. The only other approaches to the type-2 problem which have been explored are the three mixed integer programs mentioned earlier from Esmaeilbeigi *et al.* (2016).

2.2 Mixed Integer Programming

A Mixed Integer Program is a common extension of the fundamental Linear Program (LP) of the Operations Research discipline. MIPs present the additional constraint on some of their decision variables that restrict their domain to exclusively integer values. Problems which require integer valued solutions occur in many real-world scenarios. Example 3 presents a simple but quite useful application of a problem with integer variables.

> **Example 3.** Consider assigning staff members to complete a given set of tasks. For a solution of this problem to be feasible we require that no fraction of any staff member is split across multiple tasks. Using continuous variables to model this problem will lead to invalid solutions and thus integer-valued variables are required. □

The flexibility of the mixed integer programming approach to modelling problems

has allowed MIPs to model a wide-range of optimization problems. A typical solution methodology when approaching a MIP is to consider its linear relaxation. This presents a much simpler problem to solve, but recovering a feasible integer solution from this relaxed problem can prove challenging.

2.2.1 Branch and Bound

The Branch and Bound (B&B) algorithm is a general tool which can be applied to solve optimization problems which was first proposed by Land and Doig (1960). The structure of the method is general enough that it can be used as a design paradigm for more involved ways of exploring the solution space. As a result, it has spawned a variety of adaptations in the field of operations research since its inception. Some of these include branch-and-cut (Padberg and Rinaldi 1991), branch-and-price (Savelsbergh 1997) and branch-and-infer (Bockmayr and Kasper 1998). A procedure based on the B&B algorithm is the typical way of approaching MIPs and so we now give an overview of how this method operates.

The B&B method constructs a rooted tree used to explore the set of all feasible solutions. At a node of the tree, a decision variable is chosen as the branching variable and the domain of possible values for that variable is divided between two child nodes. The solution space of the current node is split into two smaller disjoint subsets in these child nodes. At the root of the tree, the solution space corresponds to all possible feasible solutions to the original LP relaxation of the problem. For each new node created by branching, the corresponding relaxed LP is solved to determine the optimal value at each node. When each node is processed, a procedure called *fathoming*, its optimal value is checked for both integrality and feasibility. If the node's solution is optimal and integer then it is compared to the current best integer solution. In a minimization (maximization) problem, if the new solution is less (more) than the current best, then the current best is replaced with the new optimal integer solution. This new solution is called the incumbent. Branching on the feasible solution space and solving each relaxed sub-problems continues until all branches have been either explored or are deemed unnecessary to explore, *i.e.*, *pruned*. At termination, the current incumbent solution is the optimal solution to the original non-relaxed MIP, and thus the original problem is solved. We provide some pseudo-code of the B&B algorithm for a minimization MIP in Algorithm 1 in the Appendices.

2.2.2 MIP Solver: Gurobi

Gurobi is a commercial solver designed for mathematical optimization problems including LPs, quadratic programs (QPs) and MIPs. Significant advances have been made to Gurobi's capabilities and other MIP solvers during the last few decades due to a number of improvements in the theory and computational hardware available [13]. Some of the most important improvements in this area are detailed in Lodi

(2010). These factors have given MIP solvers the ability to solve highly complex optimization problems while providing a relatively easy to use interface for doing so.

To solve a MIP, Gurobi uses the branch-and-cut version of the B&B algorithm. Preprocessing is a common procedure implemented by solvers where the problem's input to the solver is altered to remove obviously sub-optimal solutions from the solution space without removing any possibly optimal solution. Further to this pre-solving step, there are numerous improvements made to the procedure of the B&B also. Additional constraints called *cutting planes*, or simply *cuts*, are added to the model which do not remove any feasible solution but result in the linear relaxation being closer to the non-relaxed problem's feasible solution space. The feasible solution space of the original problem is called the *convex hull*.

Advances in the hardware over the past few decades has been the other main source of improvement to solvers such as Gurobi. Utilizing an algorithm which depends on branching the solution space into disjoint subsets, has naturally favoured solvers such as Gurobi by allowing them to take full advantage of parallel computing. Currently, Gurobi can concurrently solve 1024 branches of a B&B tree. Taking advantage of this parallelization can provide more than just a 1024-fold increase is computational power due to the communication of solutions between branches. For example, if one branch finds a high-quality integer feasible solution then all other branches can immediately reap the benefits by getting access to more effective pruning.

2.3 Constraint Programming

Constraint Programming (CP) is a framework for modelling and solving optimization problems. This framework has its roots in the field of Artificial Intelligence and the initial conception of the idea is due to Jaffar and Lassez (1987), who pointed out that the language PROLOG II is a special case of a more general scheme: Constraint Logic Programming. CP uses a declarative descriptive of the problem by defining a set of decision variables each with a set of possible values, *i.e.*, domains; and a set of constraints which restrict the possible combinations of values for the decision variables.

Over the past two decades, CP has been used extensively to tackle a variety of combinatorial optimization problems. The success of constraint programming can generally be attributed to two of its unique features

1. *heterogeneous global constraints* which allow the user to write a high level model using the global constraints to capture important combinatorial structures. In most cases these constraints are formulated in such a way to be abstracted away from the specific properties of any one problem.

2. *user-defined search* which allows the user to specify a structured search procedure for exploring the solution space, using their familiarity with the problem.

CP has proven itself to be an effective solving technology for many scheduling problems such as project [12], train [49] and employee scheduling [21] as well as other types of combinatorial problems such as bin packing [47].

We now provide short explanations of the features of CP mentioned above and a description of the CP solver we chose to use, `Chuffed`. To communicate with the solver, we chose to write our constraint programs using the solver-independent CP language MiniZinc [42]. MiniZinc provides a vast library of global constraints as well as an adequate search language.

2.3.1 Global Constraints

Global constraints are one of the powerful tools made available to us when formulating constraint programs. They allow us to succinctly encode complicated combinatorial structures that commonly occur among discrete problems. One of the simpler, but highly applicable, global constraints is given in Example 4.

> **Example 4.** `alldifferent` is a global constraint that takes an array of integer variables, each with a possibly different domain of possible values. This constraint simply requires all given variables to take different values in the solution. □

One drawback of global constraints is that each one must be individually implemented in the solver. When posed by a problem with inherent combinatorial structures, your choice of solver can be restricted as some solvers may only partially support, or not support at all, the needed global constraints. However the benefits of this type of constraint are not to be understated. The fact that global constraints are formulated to be problem-agnostic can provide users with much easier access to advances in the state-of-the-art solving technology. To illustrate this point we consider the global constraint `cumulative`.

The `cumulative` global constraint was first introduced by Aggoun and Beldiceanu (1993) to efficiently solve and succinctly model a complex scheduling problem. Since then, the `cumulative` constraint has been widely used in a variety of problems due to the fact that resources which are cumulatively restricted occur in many real-world problems. A cumulatively restricted resource could represent a limited work force needed to execute a set of tasks or a machine with a restriction on the number of tasks it can run in parallel [57].

The `cumulative` constraint has commonly been applied to the Resource Constrained Project Scheduling Problem (RCPSP), which has many variants that are well-disposed to utilize the benefits of this global constraint. Since its introduction, the `cumulative` constraint has had a number of improvements proposed, some of

these were by Caseau and Laburthe (1994), Nuijten (1994), Baptiste and Pape (2000) and Carlier and Pinson (2004). Each time an improvement is made to `cumulative` — either by strengthening its propagation or by allowing it to capture more varied and complex structures — the results of all problems where `cumulative` has previously been applied are improved as well.

By encoding the combinatorial nature of a problem using the global constraints made available to us by CP, we are able to not only take advantage of all the previous theoretical improvements to a global constraint, but all future advances as well.

2.3.2 User-Defined Search Procedures

MiniZinc is a powerful modelling language for constraint programming which is capable of concisely specifying complex models. One of the reasons it was the language we chose to write our constraint programs with was its ability to define relatively structured search procedures with ease [55, 56, 65].

The current search facilities provided by MiniZinc can be separated into two types: basic searches and compositional searches. To demonstrate the features of a basic search we present a simplification of MiniZinc code here in Listing 2.1. This basic search specifies a search procedure over an integer variable, namely `int_search`,

```
int_search(array[int] of var int, ann, ann);
```

Listing 2.1: Basic integer search procedure

The declaration of the basic integer search, requires 3 inputs to fully define how it divides the set of feasible solutions and chooses which variables to branch on. Let the declaration be defined by input $(x, varselect, varsplit)$. The decisions variables which need to have their values fixed are given by x. The annotation $varselect$ specifies the order in which to select the variables depending on some selection criteria. Lastly, $varsplit$ defines how to branch on the chosen variable's possible values. During this branching process, variables which have their domains reduced to a singleton value are fixed and removed from later consideration.

There are many variable selection strategies which MiniZinc's search language offers. The simplest selection criterion is `input_order` which selects the variables in the order given. Some of the possible selection criteria which are offered by MiniZinc are presented in Table 2.1.

MiniZinc also offers a multitude of value splitting methods to allow the user to specify application tailored branching strategies. The branching of any search procedure is characterized by creating one branch satisfying the specified splitting property and another branch satisfying its negation. For example, using the splitting method `indomain_min` will create one branch in the search tree where the chosen variable's domain is reduced to its minimum value — *i.e.*, this variable is fixed — and another branch where the minimum value of the variable is removed from the domain. Some of the possible branching methods MiniZinc has implemented are presented in Table 2.2. Example 5 presents a simple use of the `int_search`

Table 2.1: Examples of MiniZinc's variable selection criteria

Selection Criteria	Definition
`input_order`	Choose variables in the order given
`smallest`	Choose the variables with the smallest possible value in their domain
`smallest_largest`	Choose the variables with the smallest largest possible value in their domain
`first_fail`	Choose the variables with the smallest domain

Table 2.2: Examples of MiniZinc's variable splitting methods

Variable Splitting	Definition
`indomain_min`	Branch on the minimum value of the chosen variable
`indomain_max`	Branch on the maximum value of the chosen variable
`indomain_random`	Branch on a random value in the chosen variable's domain
`indomain_split`	Branch on the chosen variable such that the domain is split in half

procedure on an array of integer variables.

Example 5. For example, consider a search annotation

$$\texttt{int\_search}([\texttt{x}, \texttt{y}, \texttt{z}],\ \texttt{smallest},\ \texttt{indomain\_min}),$$

with current domains of the variables $x \in \{5\}$, $y \in \{7, \ldots, 12\}$, $z \in \{2, \ldots, 9\}$. As we have the `smallest` variable selection strategy, the variable z will be chosen as the branching variable. Using `indomain_min` will result in one branch were z's value is fixed to 2 and another branch where its domain is reduced to $\{3, \ldots, 9\}$. □

Further to the search strategy for arrays of integer variables, MiniZinc also has similar search structures available for arrays of boolean variables (`bool_search`) and arrays of float variables (`float_search`).

The most important feature in MiniZinc's search language is its ability to compose primitive searches to create complex application-tailored search procedures. This is done through the use of search combinators developed in [56] and [55]. We briefly detail here how MiniZinc's sequential search combinator, `seq_search`, can be used. MiniZinc's sequential search definition is given in Listing 2.2 and we show how it can be used in Example 6.

```
seq_search(array[int] of ann);
```

Listing 2.2: Sequential search procedure

Example 6. Consider the sequential search annotation in Listing 2.3, which first will decide all boolean variables $b1, \ldots, b3$, by branching on a random variable in each one's domain. After all boolean variables are fixed, the procedure will search over possible values for the x variables choosing to branch on the one with the smallest value in its domain. □

```
seq_search( [bool_search([b1,b2,b3], input_order, indomain_random),
             int_search( [x1,x2,x3], smallest,    indomain_min)    ] );
```

Listing 2.3: Example of a sequential search annotation

Later in Section 4.4.2.3, we formulate specific search strategies to effectively tackle our problem. To achieve this, we will provide a more in-depth look into some of the recent developments in MiniZinc's search language and how we have employed them.

2.3.3 CP Solver: `Chuffed`

To solve the constraint programs that arise in our solution methodology, we chose to use the CP solver `Chuffed` which was initially developed in the Ph.D thesis of Chu (2011). In this section we will note the advantages that this solver provides and also the solver's drawbacks.

What sets `Chuffed` apart from other competing CP solvers are its two main features: lazy clause generation (LCG) and boolean satisfiability solving (SAT). Since the introduction of LCG to solving technology, CP solvers which have integrated it into their procedures have become very well suited to scheduling problems [45]. Their ability to learn nogood clauses has allowed this type of solver to effectively prune the search space, using conflict-driven search to guide the branching strategy [58, 59, 61, 62].

An important feature of CP solvers is their Finite Domain (FD) propagation. A propagator can be thought of as an 'explanation' of a truth. Generally this can be represented by $a_1 \wedge a_2 \wedge \ldots \wedge a_n \rightarrow d$, where each a_i represents an existing domain restriction and d represents the implied domain restriction. This is demonstrated by Example 7. CP solvers which have access to SAT solving technology extend their ability to use FD propagation on clauses of literals, rather than being restricted to propagation of variable domains.

Example 7. Suppose our problem has three decision variables with domains $x \in \{3, 10\}$, $y \in \{2, 10\}$ and $z \in \{0, 10\}$. Given the constraint $z \geq x + y$, we can use finite domain propagation to add the following propagator to our model $(x \geq 3) \wedge (y \geq 2) \rightarrow (z \geq 5)$. □

Later in Section 4.4.2.2, we make use of the global constraint `cumulative` to encode some of the combinatorial nature of our problem. Recent additions in the propagation qualities of `cumulative`, such as time-table filtering [57] and time-table-edge-finding filtering [60], are all readily available in `Chuffed`.

The final advantage of `Chuffed` is that all search capabilities offered by the language MiniZinc have been fully implemented. Perhaps most importantly, `Chuffed` is the only available solver that supports a recent addition to MiniZinc's search language, which will be detailed further in Section 4.4.2.4. This together with our familiarity with `Chuffed` and it being a state-of-the-art solver for numerous scheduling problems, which are closely related to the constraint programs we encountered, made it the obvious choice of solver.

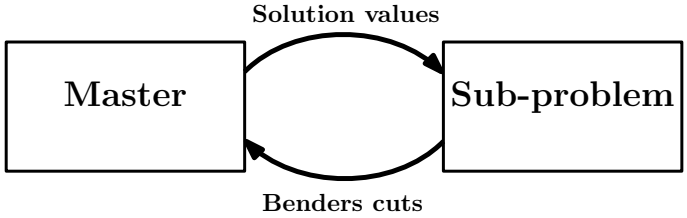
One significant disadvantage of `Chuffed` which must be mentioned, is its inability to be run in parallel. The most common way to parallelize CP solvers is by splitting the search space into disjoint subspaces and running independently on each branch. Although `Chuffed` being a LCG solver with nogood learning has presented some unique difficulties in achieving an effective parallelization. Communicating learnt clauses between parallel solvers is advantageous to reduce the amount of repeated work, but the ability to communicate complex clauses is quite limited. Recent work by Ehlers and Stuckey (2016) has shown that it is possible to achieve an effective parallelization of `Chuffed`, but unfortunately this technology is still in development and we were not able to make use of it for our computational tests.

2.4 Benders Decomposition

Benders decomposition is a method for solving optimization problems which was original proposed in the seminal paper by J. F. Benders in 1962. The method is loosely based on the notion of "learning from one's mistakes"; which is achieved through intelligently partitioning the problem and delayed constraint generation. This approach is well-suited to problems where the decision variables can be divided into a set of primary and secondary decisions. When the secondary variables, also called *complicating* variables, are fixed the original problem becomes substantially simpler to solve. Fixing the complicating variables either partitions the problem into two easier problems, namely the *master* and the *sub-problem*, or results in such a structure where the solution is straightforward. This idea is demonstrated in the following linear program,

$$\begin{aligned} \text{Min} \quad & x_1 + x_2 + x_3 + y \\ \text{subject to} \quad & 2x_1 + x_2 + 3y \geq 8 \\ & x_2 - 2x_3 - 2y \geq 3 \\ & x_3 + 7y \geq 2. \end{aligned}$$

Figure 2.2: High level structure of the Benders decomposition method



We have a set of complicating variables, y, occurring in all constraints and a set of primary variables, x, whose solution would be straightforward if y were fixed.

Once the problem is divided into the master and sub-problems, the Benders algorithm iterates between them; solving each one in turn and uses these solutions to gradually learn where the most important infeasibilities of the problem exist. A top-level view of how the Benders decomposition procedure iterates is given in Figure 2.2. In the case of minimization, the solution to the relaxed master problem, or RMP, provides a lower bound on the optimal value. This trial solution of the RMP is then passed to the sub-problem and, if possible, the sub-problem is solved to optimality. If the solution to the sub-problem is infeasible or sub-optimal, we find out why and then use this information to design a constraint to pass back to the master, which will rule out the current trial solution. This constraint is called a *Benders cut*. It is preferable to formulate these cuts such that not only the current trial solution is removed from the relaxed master's feasible region, but also a large class of other solutions which are infeasible/sub-optimal for similar reasons.

Since its introduction by Benders, this approach has been generalized to be applicable to a wider range of problem structures. We now provide a review of the classical version of the method as well as some of its recent adaptations.

2.4.1 Classical Benders

The initial decomposition method proposed by Benders was used to find the solutions of MIPs via partitioning the problem into two simpler linear programs. We refer the interested reader to the surveys by Costa (2005) and Rahmaniani *et al.* (2017). We now present how this decomposition could be applied to a general minimization problem, based on the work of Saharidis and Ierapetritou (2010).

Consider a problem where the set of decision variables can be divided into two sets, x and y, where y is a set of complicating variables. The problem can be modelled by the following program

$$\text{Min} \quad c^T x + d^T y \tag{2.1}$$

$$\text{s.t.} \quad Ax + By \leq b \tag{2.2}$$

$$Fy \leq p \tag{2.3}$$

By applying Benders decomposition here we receive two simpler problems: a master deciding the y variables with all constraints involving y, and a sub-problem deciding the x variables with all constraints involving x. So by considering a trial solution, $y = \bar{y}$, of the complicating variables, we can pass this information to the sub-problem which is formulated as follow,

$$\text{Min} \quad c^T x + d^T \bar{y} \tag{2.4}$$

$$\text{s.t.} \quad Ax \leq b - B\bar{y} \tag{2.5}$$

Using the duality properties of LPs we can instead consider the dual variables u and represent this sub-problem by its dual as follows,

$$\text{Max} \quad (b - B\bar{y})u \tag{2.6}$$

$$\text{s.t.} \quad A^T u \leq c \tag{2.7}$$

This is the sub-problem which is solved to optimality or proven infeasible. The objective function of the sub-problem varies with y but the feasible set of solutions is independent of these complicating variables.

If the solution of the master problem is feasible then the dual of the sub-problem will find optimality at an extreme point, while if the master's solution was infeasible, the sub-problem is unbounded indicating the existence of an extreme ray in its solution space. In either case, we can formulate a Benders cut to add back into the master problem to remove the current solution from the feasible set.

When the sub-problem's solution is bounded and optimal we can add what are called *optimality* cuts to the master problem

$$z \geq (b - By)\bar{u}_i^T + d^T y, \quad i = 1, 2, \ldots \tag{2.8}$$

where each u_i corresponds to an extreme point. When the solution is unbounded we can create *feasibility* cuts for each extreme ray

$$(b - By)\bar{v}_j^T \leq 0, \quad j = 1, 2, \ldots \tag{2.9}$$

where each v_j corresponds to an extreme ray. After these cuts are created we add them into the master problem and re-solve to find a new trial solution for y. The new RMP is formulated as follows

$$\text{Min} \quad z \tag{2.10}$$

$$\text{s.t.} \quad Fy \leq p \tag{2.11}$$

$$z \geq (b - By)\bar{u}_i^T + d^T y \quad i = 1, 2, \ldots \tag{2.12}$$

$$(b - By)\bar{v}_c^T \leq 0 \qquad j = 1, 2, \ldots \tag{2.13}$$

Geoffrion (1972) generalized the classical approach to a broader class of optimization problems which meant the sub-problems no longer needed to be formulated as a linear program but could also be nonlinear. Further to this improvement, Magnanti and Wong (1981) investigated ways to generate Pareto-optimal Benders cuts to produce a tighter relaxation in the master problem. Other possible enhancement methods for Benders decomposition exist including two- and three-phase heuristic approaches devised by Cordeau *et al.* (2001); trust regions to significantly reduce the runtime considered by Santoso *et al.* (2005) and Maher *et al.* (2014); and parallelized solving of the decomposed problem studied by Dempster and Thompson (1998) and Nielsen and Zenios (1997).

2.4.2 Logic-Based Benders

Hooker and Ottosson's seminal paper from 2003 propose another generalization of Benders decomposition to a much wider context while maintaining the strategy of "learning from ones mistakes". One of the central ingredients in Benders decomposition is the Benders cuts which allow information to be communicated between the partitions of the problem. In the classical case, the formulation of the sub-problems is restricted to linear and non-linear programs. However, this requirement hinders the ability of Benders decomposition to be applied to problems which are highly combinatorial in nature, such as scheduling, as it can be impractical to formulate combinatorial problems using a linear or non-linear model.

Hooker and Ottosson note that "the key to generalizing Benders decomposition is to extend the class of problems for which a suitable dual can be formulated". To this end, they introduced the concept of an *inference dual* of any optimization problem to the literature. We note that this dual is different to the traditional LP dual that the reader my be familiar with. The inference dual is a proof of optimality which allows us to formulate Benders cuts purely through logical reasoning. This adaptation of the classical method proposed in [10] is named *logic-based* Benders decomposition.

A formal description of logic-based Benders decomposition applied to a minimization problem, based on the work of Hooker (2007), is presented as follows.

Logic-based Benders decomposition applies to problems of the form

$$\text{Min} \quad f(x, y) \tag{2.14}$$

$$\text{s.t.} \quad C(x, y) \tag{2.15}$$

$$x \in D_x \tag{2.16}$$

$$y \in D_y \tag{2.17}$$

where $C(x, y)$ is a set of constraints depending on both the x and y variables. General domains of x and y are denoted by D_x and D_y respectively. If we consider fixing

the value of y to $\bar{y} \in D_y$, then the following sub-problem arises

$$\text{Min} \quad f(x, \bar{y}) \tag{2.18}$$

$$\text{s.t.} \quad C(x, \bar{y}) \tag{2.19}$$

$$x \in D_x \tag{2.20}$$

where $C(x, \bar{y})$ is the set of constraints that result by fixing $y = \bar{y}$.

The inference dual of the program (2.18–2.20) is the problem of inferring the tightest possible lower bound on $f(x, \bar{y})$ from $C(x, \bar{y})$. Using the symbol "$\Rightarrow$" to denote logical implication, the inference dual can be represented as follows

$$\text{Max} \quad v \tag{2.21}$$

$$\text{s.t.} \quad C(x, \bar{y}) \overset{P}{\Rightarrow} f(x, \bar{y}) \geq v \tag{2.22}$$

$$v \in \mathbb{R} \tag{2.23}$$

$$P \in \mathcal{P} \tag{2.24}$$

where $A \overset{P}{\Rightarrow} B$ means that B can be deduced from A via proof P where $\mathcal{P}$ is a family of proofs.

The solution found by optimizing the inference dual can be interpreted as a proof of the tightest possible bound, $\hat{v}$, on $f(x, y)$ when y is fixed to $\bar{y}$. From this bound we derive a bounding function $B_{\bar{y}}(y)$ for other values of y. The bounding function must satisfy two properties:

B1: $B_{\bar{y}}(y)$ provides a valid lower bound on $f(x, y)$ for any given $y \in D_y$. That is, $f(x, y) \geq B_{\bar{y}}(y)$ for any feasible (x, y) in the program (2.14–2.17).

B2: In particular, $B_{\bar{y}}(\bar{y}) = \hat{v}$.

If we denote the objective function value of the program (2.14–2.17) by z then the valid inequality $z \geq B_{\bar{y}}(y)$ is a Benders cut whose validity is proved by the inference dual.

At the μ^{th} iteration of the Benders algorithm, we solve the master problem with constraints defined as the cuts generated so far

$$\text{Min} \quad z \tag{2.25}$$

$$\text{s.t.} \quad z \geq B_{y^h}(y) \quad h = 1, 2, \ldots, \mu - 1 \tag{2.26}$$

$$z \in \mathbb{R} \tag{2.27}$$

$$y \in D_y \tag{2.28}$$

where $y^1, \ldots, y^{\mu-1}$ are the previous solutions of the master problem. The solution $\bar{y}$ of the μ^{th} master defines the next sub-problem to be solved.

We denote $v_1^*, \ldots, v_{\mu-1}^*$ as the optimal values of the previous $\mu - 1$ sub-problems. The algorithm continues to iterate between the master and sub-problem until the optimal value z_μ^* of the master equals $v^* = \min\{v_1^*, \ldots, v_{\mu-1}^*\}$.

This new decomposition method allows the general strategy of Benders decomposition to be applied to a much wider class of problems, however using the logic-based version means that for each problem a new method of generating Benders cuts needs to be devised. Further to this, each new Benders cut must be rigorously proven to be valid, *i.e.*, not remove any possibly optimal solution from the solution space. In the classical case, this formal proof was not expressly required as the validity of any cut was confirmed by the duality theory of linear programming.

2.4.3 Hybrid-Benders

To illustrate the benefits that employing logic-based Benders decomposition can provide, we detail one of its a special cases which has demonstrated success in the literature: hybrid Benders decomposition. What distinguishes the hybrid variant is the use of multiple kinds of solving technology to tackle a single problem within a logic-based Benders decomposition framework.

Here we will examine examples from the literature where MIP solvers were effectively combined with CP solvers to take advantage of their complimentary strengths. The benefits of employing such an approach are clear when the structure of a problem allows a natural decomposition, and the resulting partitions can each be efficiently solved by different kinds of solvers. MIP solvers have been successfully applied to solve a wide range of problems including network synthesis, crew scheduling, planning and capital budgeting. While the FD propagation of CP solvers has been effectively applied to solve combinatorial discrete optimization problems and feasibility problems for resource allocation and scheduling.

Jain and Grossmann (2001) considered a parallel machine sequencing problem and devised a Benders decomposition of the original problem into a set of assignment decisions and a set of sequencing decisions. A MIP model was devised to tackle the assignment problem, which could take advantage of the LP relaxation and B&B search offered by MIP solvers. Testing the feasibility of the solutions of the relaxed problem could be achieved efficiently by making use of the propagation engines offered by CP solvers. The feasibility sub-problems were linked to the assignment master problem via a set of feasibility Benders cuts. In their case, multiple nogood cuts were generated at each iteration of the Benders algorithm.

The complimentary strengths between MIP and CP solvers have been noted by Timpe (2002), Benoist *et al.* (2002) and Hooker (2005). The problems tackled by these papers demonstrate the wide applicability of this framework. Specifically these authors approached the following problems: polypropylene batch scheduling, call centre scheduling and multi-machine scheduling respectively.

Li and Womer (2009) use the hybrid Benders decomposition framework to model

a variant of the well-studied Resource Constrained Project Scheduling Problem (RCPSP). The RCPSP is a highly combinatorial optimization problem with a number of extensions in the literature [62, 64]. This scheduling problem is naturally suited to be modelled by the global constraints `cumulative` and `disjunctive` offered by CP. Due to this, a number of advances have been made in theory of CP solvers to improve their ability to solve the RCPSP. Some of these advances were detailed previously in Section 2.3.

The variant considered by Li and Womer introduces an additional layer of complexity to the problem by allowing each of the resources to have more than just a singular capability, as is the case in the basic RCPSP. This addition results in a set of assignment decisions between the resources and the tasks of the project. As the problem now had two distinct sets of decisions — assignment and scheduling — they chose to employ a Benders decomposition. The consequence was a relaxed master problem, modelled as a MIP to handle the assignment decisions and a feasibility sub-problem encoded as a constraint program. As a result of this decomposition, Li and Womer were still able to utilize the global constraints of CP to effectively encode the scheduling aspect of the problem. The authors' hybrid Benders decomposition is notable due to the variety of both optimality and feasibility Benders cuts they were able to formulate. In their further work [34, 36, 37] a number of problems involving assignment and scheduling decisions are tackled again using this hybrid approach to achieve an effective decomposition.

Tran and Beck (2012) consider another variant of the RCPSP, closely related to the assembly line balancing problem we are concerned with in this thesis. Specifically, their focus was on the resource constrained scheduling problem with unrelated machines and sequence-dependent setup times between tasks. They applied a logic-based Benders decomposition and formulated a hybridized solution methodology. Their decomposition of the problem gave them the ability to formulate each sub-problem as an asymmetric TSP. The TSP is a combinatorial optimization problem which is of particular interest to the academic community and as such, dedicated solvers which are tailored to this problem are available [6]. Having the ability to take advantage of the wealth of research into the inherent combinatorial sub-structure of their problem, the TSP, allowed their method to solve problems six orders of magnitude faster than what was previously possible by MIP solvers, while also allowing the optimal solution to be found for instances twice as large as previously able.

2.4.4 Summary

We hope these applications of logic-based Benders decomposition to a range of difficult problems has provided some insight into this framework's applicability. When problems can naturally be decomposed into their more basic combinatorial components, it is possible to achieve superior results by combining solution approaches which can more readily exploit the inherent structures of the problem.

3

Mixed-Integer Programming Formulation

This chapter presents three possible mixed-integer programs to model the SUALBSP-2. We adapt the work of Esmaeilbeigi *et al.* (2016), in which three similar formulations were presented. These programs will provide a benchmark against which to test our solution methodology presented in the following chapter. Esmaeilbeigi *et al.* gave possible valid inequalities which could be added to these formulations, however these additional constraints are as yet untested by the literature.

One of the research aims we have for our project is to test and compare our solution methodology against those previously presented in the literature. However, due to the limited research that has been done into the type-2 variant of the SUALBSP, there is little work to compare our approach against. The only computational results available which are related to the SUALBSP-2 are by Yolmeh and Kianfar (2012). For reasons detailed in Section 5.1, we chose to test on a different set of data to that of Yolmeh and Kianfar, which meant that there is currently no computational results that can be used to make a direct comparison. It is possible to roughly judge the quality our results against those of Yolmeh and Kianfar by comparing how our solution procedure performs on instances of similar size. However, to gain an exact measurement of the quality of different approaches, we were motivated to implement and test the mixed-integer programs presented in Esmaeilbeigi *et al.*.

In order to capture a richer structure of the SUALBSP, Table 3.1 expands upon the notation that was defined earlier in Section 1.3. The work of Esmaeilbeigi *et al.* provides the basis of the notation we formulated for our central thesis. Our choice of big-M value listed in Table 3.1 is calculated as the maximum possible cycle time value, which is detailed later in Equation 3.24.

Table 3.1: Notation Summary

Notation	Definition
n	The number of tasks
m	The number of stations
V	Set of tasks, indexed by i, j and v
K	Set of stations, indexed by k
$E(E^*)$	Set of direct (all) (i,j) precedence relations
t_i	Processing time of task i
t_{sum}	Sum of all processing times, $\sum_{i \in V} t_i$
$P_i(P_i^*)$	Set of direct (all) predecessors of task $i \in V$
$F_i(F_i^*)$	Set of direct (all) successors of task $i \in V$
ϕ_{ij}	Setup time of task $j \in V$ when performed immediately after task $i \in V$ in the forward station load
β_{ij}	Setup time of task $j \in V$ when performed immediately after task $i \in V$ in the backward station load
F_i^{ϕ}	Set of tasks which may directly follow task $i \in V$ in the forward station load, $F_i^{\phi} = \{\, j \in V - (F_i^* - F_i) - P_i^* - \{i\} \mid FS_i \cap FS_j \neq \emptyset \,\}$
F_i^{β}	Set of tasks which may directly follow task $i \in V$ in the backward station load, $F_i^{\beta} = \{\, j \in V - F_i^* \mid FS_i \cap FS_j \neq \emptyset \,\}$.
P_i^{ϕ}	Set of tasks which may directly precede task $i \in V$ in the forward station load, $P_i^{\phi} = \{\, j \in V \mid i \in F_j^{\phi} \,\}$
P_i^{β}	Set of tasks which may directly precede task $i \in V$ in the backward station load, $P_i^{\beta} = \{\, j \in V \mid i \in F_j^{\beta} \,\}$
FS_i	Set of stations which task $i \in V$ can be feasibly assigned
FT_k	Set of tasks which can be feasibly assigned to station $k \in K$
$\bar{c}(\underline{c})$	Upper (lower) bound on the cycle time
M	Sufficiently large "big-M" value, $M = t_{sum} + \beta_{n,1} + \sum_{i=1}^{n-1} \phi_{i,i+1}$

To illustrate how this notation can define the input properties of the SUALBSP-2, we return to the simple instance of the problem in Example 8 originally presented in Chapter 1.

Example 8. Again, consider the instance of the SUALBSP-2 as defined previously in Example 2, with setup times defined in Tables 1.1 and 1.2 (see Chap. 1). This simple example has input parameters defined as follows,

$$V = \{1, 2, 3, 4 = n\}, \qquad K = \{1, 2, 3 = m\},$$
$$E = \big\{(1,2), (2,3), (2,4)\big\}, \qquad t = (6, 5, 2, 9),$$

Table 3.2: Decision variables of FSBF-2

Variable	Definition
c	cycle time of the assembly line, continuous
s_i	start time of task $i \in V$, continuous
x_{ik}	1 iff task $i \in V$ is assigned to station $k \in FS_i$
y_{ij}	1 iff task $i \in V$ is followed directly by task $j \in F_i^\phi$ in the forward station load
z_{ij}	1 iff task $i \in V$ is the last task completed and task $j \in F_i^\beta$ is the first in the same station
o_{ik}	1 iff task $i \in V$ is the last task processed on station $k \in FS_i$
w_i	encodes the number of the station task $i \in V$ is assigned, continuous

$$F_i^\phi = \big(\{2\},\{3,4\},\{4\},\{3\}\big), \quad F_i^\beta = \big(\{1\},\{1,2\},\{1,2,3,4\},\{1,2,3,4\}\big),$$
$$P_i^\phi = \big(\emptyset,\{1\},\{2,4\},\{2,3\}\big), \quad P_i^\beta = \big(\{1,2,3,4\},\{2,3,4\},\{3,4\},\{3,4\}\big),$$
$$FS_i = \big(K \mid i \in V\big), \qquad FT_k = \big(V \mid k \in K\big).$$

Note: for brevity we have omitted the parameters defined by the transitive closure operator, *i.e.*, E^*, P_i^* and F_i^*. The big-M value is calculated by

$$M = t_{sum} + \beta_{n,1} + \sum_{i=1}^{n-1} \phi_{i,i+1} = 22 + 3 + (0 + 3 + 3) = 31. \quad \square$$

3.1 First Station-Based Formulation

The first mixed integer program of the SUALBSP-2 that we present is a station-based formulation, and for brevity we denote this formulation by FSBF-2. The decisions of a station-based formulation are related to the stations along the assembly line.

The definitions of the decision variables in this model have been presented in Table 3.2. The primary decision is denoted by c, which is a continuous variable indicating the cycle time of the assembly line. The start times, s_i, of each task denote when a task begins execution relative to the launch time of the station where task i is performed at. The assignment decisions between the tasks and stations are encoded by variables x_{ik}. Binary variables y_{ij} and z_{ij} are true if task j follows task i in the forward and backward station load respectively. The continuous variable w_i denotes the station number of task i.

FSBF-2: Min c (3.1)

s.t.
$$\sum_{k \in FS_i} x_{ik} = 1 \qquad \forall i \in V \tag{3.2}$$
$$\sum_{k \in FS_i} k \cdot x_{ik} = w_i \qquad \forall i \in V \tag{3.3}$$
$$\sum_{j \in F_i^\phi} y_{ij} + \sum_{j \in F_i^\beta} z_{ij} = 1 \qquad \forall i \in V \tag{3.4}$$
$$\sum_{i \in P_j^\phi} y_{ij} + \sum_{i \in P_j^\beta} z_{ij} = 1 \qquad \forall j \in V \tag{3.5}$$
$$o_{ik} \leq x_{ik} \qquad \forall i \in V,\ k \in FS_i \tag{3.6}$$
$$\sum_{j \in F_i^\beta} z_{ij} \leq \sum_{k \in FS_i} o_{ik} \qquad \forall i \in V \tag{3.7}$$
$$\sum_{i \in FT_k} o_{ik} \leq 1 \qquad \forall k \in K \tag{3.8}$$
$$w_j - w_i \leq M \cdot (1 - y_{ij}) \text{ and } w_i - w_j \leq M \cdot (1 - y_{ij}) \qquad \forall i \in V,\ j \in F_i^\phi \tag{3.9}$$
$$w_j - w_i \leq M \cdot (1 - z_{ij}) \text{ and } w_i - w_j \leq M \cdot (1 - z_{ij}) \qquad \forall i \in V,\ j \in F_i^\beta \tag{3.10}$$
$$\sum_{i \in FT_k} t_i \cdot x_{ik} \leq c \qquad \forall k \in K \tag{3.11}$$
$$s_i + t_i + \sum_{j \in F_i^\beta} \beta_{ij} \cdot z_{ij} \leq c \qquad \forall i \in V \tag{3.12}$$
$$\sum_{i \in V} \sum_{j \in F_i^\beta} z_{ij} \geq m \tag{3.13}$$
$$s_i + c \cdot (w_i - w_j) + t_i + \phi_{ij} \cdot y_{ij} \leq s_j \qquad \forall (i,j) \in E \tag{3.14}$$
$$s_i + (t_i + \phi_{ij}) + (c + \phi_{ij}) \cdot (y_{ij} - 1) \leq s_j \qquad \forall i \in V,\ j \in F_i^\phi \setminus F_i \tag{3.15}$$
$$\underline{c} \leq c \leq \bar{c} \tag{3.16}$$
$$s_i \geq 0 \qquad \forall i \in V \tag{3.17}$$
$$x_{ik} \in \{0,1\} \qquad \forall i \in V,\ k \in FS_i \tag{3.18}$$
$$y_{ij} \in \{0,1\} \qquad \forall i \in V,\ j \in F_i^\phi \tag{3.19}$$
$$z_{ij} \in \{0,1\} \qquad \forall i \in V,\ j \in F_i^\beta \tag{3.20}$$

$$o_{ik} \geq 0 \qquad \forall i \in V,\ k \in FS_i \tag{3.21}$$

$$w_i \in \{1, 2, \ldots, m\} \qquad \forall i \in V \tag{3.22}$$

The objective function (3.1) minimizes the cycle time (as with all programs presented in this chapter), (3.2–3.15) define the main constraints on this formulation and the last constraints (3.16–3.22) are non-functional defining the domains of our decision variables. We now detail the purpose of each of the main constraints.

Constraint (3.2) ensures that each task is assigned exactly one station, (3.3) encode the station numbers of the variables w_i, constraints (3.4) and (3.5) enforce that each task has exactly one follower and one predecessor respectively (in either load direction), (3.6) requires that a task can only be the last task of a station if it has been assigned to that station, constraint (3.7) ensures that the number of last tasks is at least the number of backward setups, (3.8) enforces that each station has at maximum one last task. The four inequalities of constraints (3.9) and (3.10) encode a disjunction using a big-M value to ensure that only tasks in the the same station can be a part of the same sequence. Knapsack constraint (3.11) strengthens the formulation by requiring all tasks be completed by the cycle time, (3.12) enforces the backward setup times to bound the cycle time below, (3.13) ensures that there are at least m backward setup times, constraint (3.14) requires that the precedence relations are satisfied within stations and between stations using the cycle time as a big-M value and finally (3.15) enforces that the forward setup times are satisfied in the forward station load of each station.

Both a lower and upper bound on the cycle time is required as input for this model, so these are calculated as follows,

$$\underline{c} = \max \left\{ t'_{max},\ \left\lceil \frac{t_{sum}}{m} \right\rceil \right\} \tag{3.23}$$

$$\bar{c} = t_{sum} + \beta_{n,1} + \sum_{i=1}^{n-1} \phi_{i,i+1}, \tag{3.24}$$

where $t'_{max} = \max_{i \in V}(t_i + \beta_{ii})$. The theoretical lower bound given by (3.23) is the larger value between the maximum possible processing of any singular task in a station and the equal fractional distribution of all processing times across all m stations.

We note that the upper bound on the cycle time, given by (3.24), need only be the cycle time of some feasible solution. The precedence graph is numbered topologically, thus a precedence-feasible sequencing of all tasks within a single station is 1,2,...,n. The upper bound on the cycle time is trivially the sum of all processing times together with the setup times resulting from the above sequencing. As this assignment and sequence is valid, the upper bound is feasible.

The number of variables and constraints are both bounded by $\mathcal{O}(n^2)$. We note

Table 3.3: Decision variables of SSBF-2

Variable	Definition
g_{ijk}	1 iff $i \in V$ is followed directly by task $j \in F_i^{\phi}$ in the forward station load on station $k \in FS_i$
h_{ijk}	1 iff $i \in V$ is the last task of station $k \in FS_i$ and $j \in F_i^{\beta}$ is the first
r_i	ordering variable which represents the priority of task $i \in V$ in a station's sequence of tasks

here that this formulation of the SUALBSP-2 requires a number of disjunctive constraints, which can have an undesirable impact on the ability of MIP solvers to handle the problem. This is due to MIPs needing to formulate disjunctive constraints with a big-M value which leads to weak linear relaxations. Constraints (3.9), (3.10), (3.14) and (3.15) all encode disjunctions.

3.2 Second Station-Based Formulation

The next formulation is abbreviated as SSBF and in this case the number of variables is bounded by $\mathcal{O}(n^3)$. The decision variables in this model are similar to FSBF-2 except for a few differences which are presented in Table 3.2.

SSBF-2: Min c (3.25)

s.t. (3.4), (3.5), (3.16), (3.18), (3.22)

$$\sum_{j \in FT_k \cap F_i^{\phi}} g_{ijk} + \sum_{j \in FT_k \cap F_i^{\beta}} h_{ijk} = x_{ik} \qquad \forall i \in V,\ k \in FS_i \tag{3.26}$$

$$\sum_{i \in FT_k \cap P_j^{\phi}} g_{ijk} + \sum_{i \in FT_k \cap P_j^{\beta}} h_{ijk} = x_{ik} \qquad \forall j \in V,\ k \in FS_j \tag{3.27}$$

$$\sum_{i \in FT_k} \sum_{j \in FT_k \cap F_i^{\beta}} h_{ijk} = 1 \qquad \forall k \in K \tag{3.28}$$

$$(n - |F_i^*| - |P_j^*|) \cdot \left(\sum_{k \in FS_i \cap FS_j} g_{ijk} - 1 \right) + r_i + 1 \leq r_j \qquad \forall i \in V,\ j \in F_i^{\phi} \tag{3.29}$$

$$r_i + 1 \leq r_j \qquad \forall (i,j) \in E \tag{3.30}$$

$$w_i \leq w_j \qquad \forall (i,j) \in E \tag{3.31}$$

$$\sum_{i \in FT_k} t_i \cdot x_{ik} + \sum_{i \in FT_k} \sum_{j \in FT_k \cap F_i^{\phi}} \phi_{ij} g_{ijk} + \sum_{i \in FT_k} \sum_{j \in FT_k \cap F_i^{\beta}} \beta_{ij} h_{ijk} \leq c \qquad \forall k \in K \tag{3.32}$$

$$\sum_{i \in FT_k \setminus \{j\}} x_{ik} \leq (n - m + 1) \cdot (1 - h_{ijk}) \qquad \forall k \in K,\ j \in FT_k \tag{3.33}$$

$$g_{ijk} \in \{0, 1\} \qquad \forall k \in K,\ i \in FT_k,\ j \in FT_k \cap F_i^{\phi} \tag{3.34}$$

$$h_{ijk} \in \{0, 1\} \qquad \forall k \in K,\ i \in FT_k,\ j \in FT_k \cap F_i^{\beta} \tag{3.35}$$

$$|P_i^*| + 1 \leq r_i \leq n - |F_i^*| \qquad \forall i \in V \tag{3.36}$$

Constraints (3.26–3.27) ensure each task has exactly one successor and one predecessor in the cyclic sequence of the assigned station and each cycle is forced to contain exactly one backward setup by constraint (3.28). Constraint set (3.29–3.30) establish the precedence relations between the tasks within each station. We note here that (3.29) is a disjunctive constraint and becomes inactive when tasks i and j are assigned to different stations, *i.e.*, $g_{ijk} = 0$. Due to this, constraint (3.31) is added to ensure the precedence relations are satisfied between stations. A knapsack constraint (3.32) is again included to strengthen the formulation. If a task is its own successor, *i.e.*, $h_{jjk} = 1$, then constraint (3.33) requires that no other tasks are assigned the same station as task j. Constraints (3.34–3.36) are the non-functional constraints defining the new decision variables domains.

This model needs to use two disjunctions to achieve a correct representation of the problem. These disjunctive constraints are (3.29) and (3.33).

3.3 Scheduling-Based Formulation

We now demonstrate that the SUALBSP can be formulated exclusively as a scheduling problem, namely as the SCBF. Taking this approach means we must again choose a new way of encoding the decisions of the problem. This leads us to the new decision variable q_{ij} which takes the value 1 if and only if task $i \in V$ is processed before task $j \in F_i^{\phi}$ along the assembly line. Together with q_{ij} we again use decision variables s_i, y_{ij} and z_{ij} as defined previously. Variable r_i is also reused however now its value represents the position of task i in the full schedule of tasks, rather than within a single cyclic sequence of a station.

SCBF-2: Min c (3.37)

s.t. (3.2), (3.3), (3.16), (3.17), (3.19), (3.20), (3.30), (3.36)

$$q_{ij} + q_{ji} = 1 \qquad \forall i \in V,\ j \in V \setminus (P_i^* \cup F_i^*) \text{ with } i < j \tag{3.38}$$

$$r_i + 1 + (n - |F_i^*| - |P_j^*|) \times (q_{ij} - 1) \leq r_j \qquad \forall i \in V,\ j \in V \setminus (P_i^* \cup F_i^*) \text{ with } i \neq j \tag{3.39}$$

$$r_j - 1 + (n - |F_j^*| - |P_j^*| - 1) \times (y_{ij} - 1) \leq r_i \qquad \forall i \in V,\ j \in F_i^{\phi} \tag{3.40}$$

$$y_{ij} \leq q_{ij} \qquad \forall i \in V,\ j \in F_i^{\phi} \setminus F_i^* \tag{3.41}$$

$$z_{ji} \leq q_{ij} \qquad \forall i \in V,\ j \in P_i^{\beta} \setminus F_i^* \text{ with } i \neq j \tag{3.42}$$

$$s_i + (t_i + \phi_{ij}) + (c + \phi_{ij}) \cdot (y_{ij} - 1) \leq s_j \qquad \forall i \in V,\ j \in F_i^{\phi} \tag{3.43}$$

$$\sum_{i \in V} \sum_{j \in F_i^{\beta}} z_{ij} = m \tag{3.44}$$

$$q_{ij} \in \{0, 1\} \qquad \forall i \in V,\ j \in V \setminus (P_i^* \cup F_i^*) \text{ with } i \neq j \tag{3.45}$$

Constraint (3.38) establishes an ordering between two tasks i and j which are not related by the precedence graph. The transitive relation between the q_{ij} variables is enforced by (3.39) together with (3.30) and (3.38). Explicitly, the combination of these constraints guarantee that for three tasks i, j and v, we have the following,

$$[q_{ij} = 1] \wedge [q_{jv} = 1] \Rightarrow [q_{iv} = 1].$$

Constraints (3.40–3.41) with (3.39) enforce the following implication,

$$[y_{ij}] \Rightarrow [r_j = r_i + 1].$$

Further to this, the conjunction of these constraints also encodes the forward setup times. Constraint (3.42) ensures that backward setups are accounted for in the schedule defined by the q_{ij} variables.

This formulation requires disjunctive constraints (3.39), (3.40) and (3.43).

3.4 Valid Inequalities

In Esmaeilbeigi *et al.* (2016) some valid inequalities are proposed to improve the formulations of the type-2 version of SUALBSP. As we mentioned previously, none of these valid inequalities were tested in their paper as their concern was with the

type-1 version of the problem.

The following two inequalities can be added to both FSBF-2 and SCBF-2

$$y_{ij} + y_{ji} \leq 1 \qquad \forall i \in V, j \in (F_i^{\phi} \cap P_i^{\phi}) \tag{3.46}$$

$$w_{ii} + \sum_{j \in F_i^{\beta}} w_{ij} + \sum_{j \in P_i^{\beta}} w_{ji} \leq 1 \quad \forall i \in V \tag{3.47}$$

Inequality (3.46) states task i cannot be before and after another task j in the forward station load. Equivalently, inequality (3.47) does the same for the backward station load. Esmaeilbeigi *et al.* note that (3.46) dominates (3.47) and thus only the first inequality was tested in the relevant formulations.

Let us denote the total setup time that occurs throughout all stations by S. The idle time of a station is the difference between the line's cycle time and the station's workload; let the total idle time across all stations be denoted by I. So the total amount of time used by the assembly line can be calculated by summing all processing times with the total setup time and the total idle time. This results in the following relation $t_{sum} + S + I = m \cdot c$. This can be formulated as a valid inequality to tighten the linear relaxation of the MIPs as follows

$$t_{sum} + \sum_{i \in V} \sum_{j \in F_i^{\phi}} \phi_{ij} \cdot y_{ij} + \sum_{i \in V} \sum_{j \in F_i^{\beta}} \beta_{ij} \cdot z_{ij} \leq m \cdot c. \tag{3.48}$$

Here we interpret the left-hand side of inequality (3.48) as a lower bound on the line capacity and the right-hand side as the tightest possible upper-bound. Once again, this inequality cannot be added to SSBF-2 due to (3.48) requiring the decision variables y_{ij} and z_{ij}.

3.5 Summary

When testing the viability of a Benders decomposition approach to a problem, it is common practice to compare the computational results against the best known mixed-integer programming formulation of the full problem. Although, due to the SUALBSP-2 being relatively under-studied, no computational results of an exact solution approach are available. As such, we will test the MIPs presented here in order to receive benchmark results. These results will allow us to judge the quality of our Benders decomposition method, which will be detailed in the next chapter.

—*Divide et impera*
(Divide and conquer)

Philip II of Macedon

4 Benders Decomposition

In this chapter we detail the Benders decomposition of the SUALBSP-2. The formulation of our logic-based decomposition can be separated into three primary pieces. The first is the formulation of the relaxed master problem and the approach used to solve it, which is detailed in Section 4.3. Similar to the first, we secondly detail our formulation of the sub-problems. The combinatorial structure that arises in the sub-problems allows multiple possible formulations and solving procedures, which are overviewed in Section 4.4. Sections 4.4.1, 4.4.2 and 4.4.3 each detail a possible approach to formulating the sub-problems.

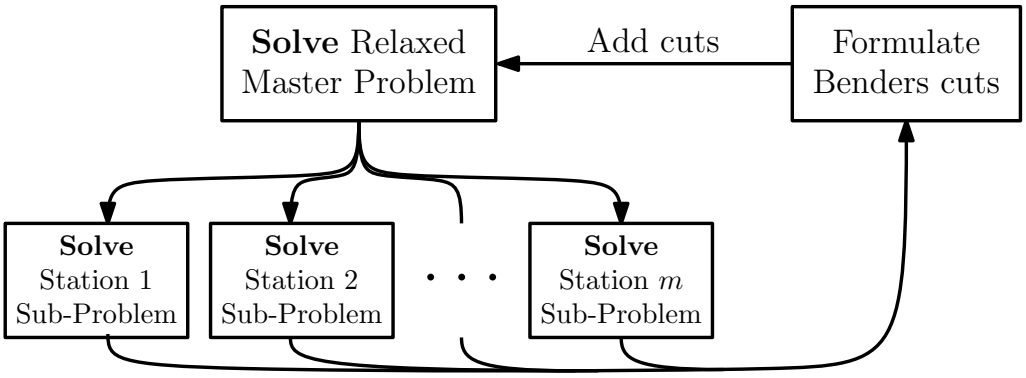
The last ingredient needed for a full specification of a logic-based Benders decomposition is how the relaxed master and sub-problems communicate during the iteration of the algorithm. As such, we present the Benders cuts which we studied and note each one's possible strengths and weaknesses depending on the solution procedure. The Benders cuts we formulated are devised and proven to be valid in Section 4.5. These cuts, and how they can improve the practical effectiveness and efficiency of this approach, is our primary contribution to the literature. We close this chapter in Section 4.6, by summarising the algorithm used to iterate through the Benders decomposition and our cut-generation scheme.

4.1 The Decomposition

We begin this chapter with an overview of how the SUALBSP-2 can be decomposed. There are two distinct types of decisions in the problem and as a result this provides the basis of our decomposition approach.

We viewed the primary decisions to be the assignment of tasks to stations and

Figure 4.1: Framework of our logic-based Benders decomposition



the secondary decisions as the execution times for each task within a station. As such, the relaxed master problem is similar to the classical SALBP-2, where the only concern is allocating the tasks to stations. The assignment found by the master should still be a precedence-feasible solution, however the setup times will be relaxed. As such, the cycle time estimated by the RMP will be the maximum load among the stations with setups assumed to be zero.

Once the master problem is solved to optimality, m sub-problems arise — one corresponding to each station — which take the subset of assigned tasks as input. The objective of each sub-problem is to find the optimal schedule of the tasks such that the precedence relations are respected and the station's workload is minimized. As mentioned earlier in Chapter 1, a sub-problem can be viewed as similar to the asymmetric TSP. After solving each sub-problem to optimality or detecting infeasibility, we can pass this information back to the master in the form of Benders cuts. If a station's minimal possible workload could not respect the master's relaxed cycle time value, then we need to devise a Benders cut to reflect this.

In Figure 4.1 we present the framework of how our Benders decomposition iterates between the partitions of the problem.

4.2 Types of Sub-Problems

When using logic-based Benders decomposition, there are two general types of sub-problems which can be implemented: optimality sub-problems and feasibility sub-problems. We now detail the unique qualities of each type and discuss their possible advantages and disadvantages when applied to the SUALBSP-2.

4.2.1 Feasibility Sub-Problems

Using feasibility sub-problems means that we are only required to check if the relaxed master's current solution is feasible. If the sub-problems are found to be infeasible

given the master's cycle time, then a cut-set is sent back to the master to remove this assignment and similar assignments from the master's feasible solution space. Testing for infeasibility can be a fast procedure as only one sub-problem needs to report that its assignment cannot satisfy the master's cycle time. Although the trade-off here is that feasibility sub-problems provide a comparatively small amount of information when compared to optimality sub-problems. This could mean that the speed we gain from only testing for feasibility could be dwarfed by the reduction in cut quality. In fact, low quality Benders cuts can lead to more iterations of the algorithm being required; if the primary bottleneck in computational runtime is due to the master then simpler sub-problems may not provide an overall runtime reduction. Even so, it is not necessarily the case that feasibility sub-problems are strictly worse than optimality sub-problems.

In the case of the SUALBSP-2 we have m sub-problems at each iteration of the decomposition. If we consider feasibility sub-problems, then there are two simple but quite distinct ways which the cutting procedure could possibly be formulated.

- *Solve all sub-problems.* By testing the feasibility of each station's assignment, we find out all stations which are infeasible and all which can satisfy the cycle time. This allows k Benders cuts to be made where k is the number of sub-problems where infeasibility is detected ($k \leq m$). Each sub-problem can be run in parallel which can vastly reduce the solution time of the sub-problems, however parallel computation isn't strictly necessary in this case.
- *Solve until infeasibility.* Run all sub-problems in parallel until one detects infeasibility, then terminate all other problems. This procedure will be much faster than the last however we gain far less information from the sub-problems. In this case we can make only two Benders cuts. The sub-problem which found infeasibility results in a Benders cut that removes its assignment of tasks from the master's feasible region. We can also make a weaker cut that removes the full assignment of tasks from occurring in future iterations.

In both cases of feasibility sub-problems, the information gained is limited and so the range of Benders cuts we can formulate is quite restricted. The feasibility cuts we considered are presented later in Section 4.5.1.

4.2.2 Optimality Sub-Problems

Programming the scheduling problem within each station as an optimality sub-problem means that the optimal sequence of tasks within each station will be found; irrelevant of whether a station's load can respect the master's cycle time estimate. The disadvantage of this when compared to just testing for infeasibility, is that solving each combinatorial scheduling problem to optimality may be far more time consuming. The primary advantage is that at each iteration of the Benders algorithm, we receive richer information from the sub-problems allowing more flexibility in the Benders cuts we devise.

We predominantly explored the use of optimality cuts in our Benders decomposition and in the coming sections (see 4.5.2–4.5.6) we will provide their formulation.

4.3 Relaxed Master Problem

The relaxed master problem handles the assignment of tasks to stations. As mixed-integer programming is an effective solving technology for allocation problems (refer to Section 2.4.3) we formulated the RMP as a MIP.

Let the current iteration of the Benders algorithm be indexed by μ and a previous iteration by ν. We now present the formulation of the RMP at the μ^{th} iteration, which we will denote by RMP(μ). The cycle time variable, c^μ, and the assignment variables, x^μ_{ik}, have equivalent definitions as in the previous chapter but are now also indexed by the Benders iteration counter. The complications presented by the sequencing problems are omitted from the RMP and as a result the total setup cost of each station is taken as zero.

$$\text{RMP}(\mu)\text{: Min } c^\mu \tag{4.1}$$

$$\text{s.t.} \quad \sum_{k \in FS_i} x^\mu_{ik} = 1 \qquad \forall i \in V \tag{4.2}$$

$$\sum_{k \in FS_i} k \cdot x^\mu_{ik} \leq \sum_{k \in FS_j} k \cdot x^\mu_{jk} \quad \forall (i,j) \in E \tag{4.3}$$

$$\sum_{i \in V} t_i \cdot x^\mu_{ik} \leq c^\mu \qquad \forall k \in K \tag{4.4}$$

$$\mathcal{C}^\nu \qquad \nu \in \{1, 2, \ldots, \mu - 1\} \tag{4.5}$$

$$\underline{c}^\mu \leq c^\mu \leq \bar{c}^\mu \tag{4.6}$$

$$x^\mu_{ik} \in \{0, 1\} \qquad \forall i \in V,\ k \in FS_i \tag{4.7}$$

The objective is to minimize the cycle time estimate in (4.1). Constraint (4.2) ensures each task is assigned exactly one station. We ensure the precedence relations are satisfied between stations with constraint (4.3). Each station's total workload, *i.e.*, sum of task processing times, is required to respect the cycle time through constraint (4.4). Finally, the set of Benders cuts found in the previous iterations are included in the master in constraint (4.5). The remaining constraints define the domains of each decision variable.

We note here that unlike the MIP formulations presented in the previous chapter, this relaxed formulation of just the assignment portion has not required any disjunctive constraints. Hence, the linear relaxation used throughout the B&B algorithm of the Gurobi solver will likely be far tighter.

Table 4.1: Decision variables of sub-problem k at iteration μ

Variable	Definition
l_k^μ	Total load of the current station k at iteration μ, k is a parameter for a given sub-problem
s_i	start time of task $i \in V_k^\mu$, continuous
y_{ij}	1 iff task $i \in V_k^\mu$ is followed directly by task $j \in F_i^\phi$ in the forward station load
z_{ij}	1 iff task $i \in V_k^\mu$ is the last task completed and task $j \in F_i^\beta$ is the first in the same station

4.4 Sub-Problems

Each of the sub-problems has a similar structure to the asymmetric Travelling Salesperson Problem with some forbidden paths. We can view the tasks as the cities and distance between any pair of tasks as the setup time. The asymmetry is due to ϕ_{ij} (β_{ij}) not necessarily being equal to ϕ_{ji} (β_{ji}).

As the structure of each scheduling problem has a highly combinatorial nature we consider three approaches: mixed-integer programming, constraint programming and fully formulating the sub-problem as a TSP. When the sub-problem is formulated as a constraint program or a TSP, then the logic-based Benders decomposition we propose can be seen as a hybrid solution methodology.

We now define new notation for formulating a sub-problem. The set of tasks assigned to station k at iteration μ is defined as $V_k^\mu \subseteq V$. The set of precedence relations is given by

$$E_k^\mu = \{\, (i,j) \mid i,j \in V_k^\mu \,\} \subseteq E,$$

The decision variables y_{ij}, z_{ij} and s_i all have equivalent definitions as in the previous chapter. The total workload of a station is decided by the variable l_k^μ. Table 4.1 provides a summary of a sub-problem's decision variables.

4.4.1 MIP Sub-Problem Formulation

Let SP-MIP(μ) denote the mixed-integer programming formulation of the sub-problem at iteration μ. Given the set V_k^μ of tasks that need to be sequenced, we can formulate the MIP as follows

$$\text{SP-MIP}(\mu)\text{: Min } l_k^\mu \tag{4.8}$$

$$\text{s.t.} \quad \sum_{j \in F_i^\phi} y_{ij} + \sum_{j \in F_i^\beta} z_{ij} = 1 \qquad \forall i \in V_k^\mu \tag{4.9}$$

$$\sum_{i \in P_j^{\phi}} y_{ij} + \sum_{i \in P_j^{\beta}} z_{ij} = 1 \qquad \forall j \in V_k^{\mu} \tag{4.10}$$

$$\sum_{i \in V_k^{\mu}} \sum_{j \in F_i^{\beta}} z_{ij} = 1 \tag{4.11}$$

$$s_i + t_i + \phi_{ij} \cdot y_{ij} \leq s_j \qquad \forall (i,j) \in E_k^{\mu} \tag{4.12}$$

$$s_i + t_i + \phi_{ij} \leq s_j + M(1 - y_{ij}) \qquad \forall i \in V_k^{\mu},\ j \in F_i^{\phi} \tag{4.13}$$

$$s_i + t_i + \sum_{j \in F_i^{\beta}} \beta_{ij} \cdot z_{ij} \leq l_k^{\mu} \qquad \forall i \in V_k^{\mu} \tag{4.14}$$

$$l_k^{\mu} \geq 0 \tag{4.15}$$

$$s_i \geq 0 \qquad \forall i \in V_k^{\mu} \tag{4.16}$$

$$y_{ij}^{\mu} \in \{0,1\} \qquad \forall i \in V_k^{\mu},\ j \in F_i^{\phi} \tag{4.17}$$

$$z_{ij}^{\mu} \in \{0,1\} \qquad \forall i \in V_k^{\mu},\ j \in F_i^{\beta} \tag{4.18}$$

Together constraints (4.9) and (4.10) ensure that each task in the station's cyclic sequence has exactly one successor and one predecessor respectively. Only one backward setup cost exists in the sequence due to (4.11) and the precedence relations are enforced by (4.12). The disjunctive constraint (4.13) requires that if tasks j directly follows task i in the sequence, then the setup cost is observed. Finally, constraint (4.14) bounds the total station load below by the completion time of the last task in the station.

Decomposing the full MIP formulation into the assignment and scheduling components has significantly reduced the number of disjunctive constraints required. Although, to correctly encode the forward setup time we still require one disjunctive constraint in SP-MIP.

4.4.2 CP Sub-Problem Formulation

Let SP-CP(μ) denote the CP formulation of the sub-problem at iteration μ. We considered a constraint programming formulation of the sub-problem as CP is a powerful solution technology available for scheduling problems (see Section 2.3).

Here we will detail the CP model we formulated and the various advantages that this approach provides. Note, that we will give simplified snippets of the MiniZinc code of our CP model.

4.4.2.1 Basic CP Model

The majority of the CP model is very similar to the MIP. To give the reader an idea of how a constraint from a MIP can instead be represented in a constraint program,

we now provide the MiniZinc code which equivalently encapsulates linear constraint (4.9).

```
constraint forall ( i in TASK ) (
      sum( j in followForward[i]  )( y[i,j] )
    + sum( j in followBackward[i] )( z[i,j] ) == 1
);
```

The set `TASK` is equivalent to V_k^{μ} in the MIP, while `followForward[i]` and `followBackward[i]` correspond to F_i^{ϕ} and F_i^{β} respectively.

All other constraints from the MIP translate in similar ways and for brevity we will only note the main differences between the two models. For the full specification of the CP program, we refer the reader to the MiniZinc code included in the Appendices A.1. The expressive nature of CP allows us to more effectively encode the disjunctive constraint from (4.13). This can be done as follows,

```
constraint forall ( i in TASK, j in followForward[i] ) (
    y[i,j] <-> ( s[i] + dur[i] + forwardSetup[i,j] == s[j] )
);
```

Where `dur[i]` corresponds to t_i and `forwardSetup` stores the two-dimensional array of forward setup times, ϕ_{ij}. Using the bijection operator, `<->`, offered by MiniZinc, we are able to succinctly create two conditional constraints in the constraint model. Also, notice that previously this constraint was encoded as an inequality, however now we are able to fix the starting time of task j if we know that task i is its direct predecessor. The flexibility and expressiveness of the CP language MiniZinc has allowed us to remove many of the negative qualities of this disjunctive constraint.

4.4.2.2 Global Scheduling Constraints

Many scheduling problems have been effectively solved by CP due to the powerful propagation qualities of the global constraints. We will now briefly detail some of the global constraints that are well-suited to our problem and how they can be utilized. Note that the constraints presented here have been somewhat simplified and some of their more complex functionality is not regarded.

Since within each station no two tasks can be executed concurrently, the `disjunctive` constraint can model this non-overlap property. This global constraint takes as input two arrays related to a task: variable start times, and fixed durations.

```
disjunctive( array of var int: start,
             array of int: dur );
```

In our problem we have start time variables s_i and fixed processing durations t_i for each task, which suggests `disjunctive` could be suited to a sub-problem. However, the actual duration between any two tasks i and j is not just the processing time of task i, but also the setup cost ϕ_{ij}. Since this setup cost depends on the task ordering, the duration cannot be fixed. As such, `disjunctive` may not provide as effective propagation as another global constraint.

The global constraint `cumulative` — previously described in Section 2.3.1 — is similar to `disjunctive` but requires two further pieces of input: the quantity of resources that each task requires and also the limit of total available resources.

```
cumulative( array of var int: start,
            array of int: dur,
            array of var int: resourceRequirement,
            int: resourceLimit );
```

In our case only one task can be processed within a station, so we define the resource limit to be 1. To overcome the problem encountered with the `disjunctive` constraint we introduce a new set of decision variables that are added to SP-CP. For each pair of tasks in V_k^μ we add the continuous variable σ_{ij} defined as follows

$$\sigma_{ij} = \begin{cases} s_i & \text{if } y_{ij} = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Now that we have a variable defining the start time of a task i which also depends on task i's place in the sequence, we can use the `cumulative` constraint to more accurately encode the non-overlapping quality of the tasks.

```
constraint cumulative(
    [ sigma[i,j]                  | i in TASK, j in TASK ],
    [ dur[i] + forwardSetup[i,j] | i in TASK, j in TASK ],
    [ y[i,j]                      | i in TASK, j in TASK ],
    1
);
```

The start time variables given to cumulative are the σ_{ij} variables. This requires the `cumulative` constraint to handle the scheduling of n^2 variables rather than just n. But we hope that the intelligent propagation it will provide will outweigh the cost. Each σ_{ij} variable has only a single corresponding ϕ_{ij} and so the durations sent to `cumulative` are now fixed values. Notably, the resource requirement of each task has been given to the constraint as the variable y_{ij}. So if task j does not directly follow task i in the task sequence then y_{ij} will take value 0 and the resource requirement of the corresponding task in the `cumulative` constraint will also be 0, *i.e.*, it will not be considered in the schedule.

By achieving an encoding of the non-overlapping quality between the tasks of a station using a global constraint, we have allowed the CP sub-problems to gain access to the well-studied propagation properties of `cumulative`.

4.4.2.3 Search Procedure

MiniZinc provides a search language containing a range of basic search procedures, such as `int_search` and `bool_search`, as well as a compositional search procedure, `seq_search`, which allows the user to define problem-specific search strategies. This allows the user utilize their familiarity with the problem to direct the search tree of the CP solver toward areas which will likely provide good feasible solutions. We

note here that including the auxiliary variable σ_{ij} in the CP model allowed us to formulate a wider range of potential search strategies.

We considered a range of basic and compositional search procedures. A simplified representation of the basic procedures are as follows

```
ann: start_io = int_search( s, input_order,      indomain_min );
ann: start_sm = int_search( s, smallest,         indomain_min );
ann: start_sl = int_search( s, smallest_largest, indomain_min );
ann: start_ff = int_search( s, first_fail,       indomain_min );

ann: sigma_ff = int_search( [ sigma[i,j] | i in TASK, j in TASK ],
                            first_fail, indomain_max );
```

To briefly illustrate how these procedures operate, we consider the `start_sm` search. In this case, the CP solver will choose to branch on the start time variables of the tasks first. The first start time variable that will be chosen, say s_i, will be the one with the smallest value in its domain; governed by the `smallest` variable selection. Once chosen, two new nodes will be created: one with the value of s_i fixed to its smallest value, and one where this smallest value is removed from s_i's domain.

We compose these five search annotations in a number of ways using the `seq_search` procedure as follows

```
ann: start_io_Then_sigma = seq_search([ start_io, sigma_ff ]);
ann: start_sm_Then_sigma = seq_search([ start_sm, sigma_ff ]);
ann: start_sl_Then_sigma = seq_search([ start_sl, sigma_ff ]);
ann: start_ff_Then_sigma = seq_search([ start_ff, sigma_ff ]);
```

Take for example, `start_sm_Then_sigma`. In this case the start time variables s_i are fully decided before moving to the σ_{ij} variables. Although having a full solution to s_i will necessarily fix all values the σ_{ij} variables, since the full sequence of tasks has been found. Hence, the compositional procedure has in fact provided little benefit for us here.

In our computational experiments we compare the results of some of these basic and compositional procedures. Further to this, we compare these results against the default search of the CP solver `Chuffed` to check whether guiding the branching of the search tree provides a benefit for this problem.

4.4.2.4 Priority Search

A recent advancement to the search functionality of MiniZinc by Feydy *et al.* (2017) has provided a new search combinator: `priority_search`. We give a brief summary of this compositional procedure and detail how we have used it to formulate new search strategies.

The `priority_search` procedure is defined as

```
ann: priority_search( ann, ann, array[int] of ann )
```

where the first and second input annotations can be defined as *selvars* (selection variables) and *varsel* (selection criteria). Similar to the sequential search combinator, this search takes as input an array of other search procedures. However

the extra functionality that `priority_search` offers is the ability to dynamically change how the array of search annotation inputs are ordered. This is unlike the `seq_search` procedure which required a fixed list that could not be changed as the search progressed. `priority_search` offers the user more flexibility than was previously available due to the ability to nest priority search strategies.

Here we provide one example of a `priority_search` that we considered and detail how three other similar priority search strategies can be formulated.

```
ann: priority_ff = priority_search( s, first_fail,
    [seq_search([
                int_search([ s[i] ],                    first_fail, indomain_min)
                int_search([ sigma[i,j] | j in TASK ], first_fail, indomain_min) ])
    | i in TASK ]
);
```

Listing 4.1: Priority search used for CP sub-problems

This search is also formulated in three similar ways by using the `input_order`, `smallest` and `smallest_largest` variable selection strategies instead of `first_fail`.

As noted earlier, the compositional search procedure we considered gained little upon the basic searches they combined. The priority search, given in Listing 4.1, branches first on the start time of the task with the fewest possible values in its domain, say task i. Once s_i is fixed, we then search over the start times of variables that can follow task i, *i.e.*, the variables σ_{ij}. Priority Search allows us to quickly remove possible followers from consideration and guide the branching of the search procedure.

4.4.3 TSP Sub-Problem Formulation

Let SP-TSP(μ) denote the TSP formulation of the sub-problem at iteration μ. Since the sequencing problem within each station is similar to the asymmetric version of the TSP, we assumed that employing a dedicated TSP solver, such as Concorde [6], would be a desirable approach to the sub-problems.

The TSP solver Concorde has been shown to be exceptionally effective at tackling the symmetric case of the TSP; solving instances of up to 85,900 cities. Although the input to the solver is restricted to just the symmetric version, if we are able to represent a station's sub-problem as a symmetric TSP then the computational runtime reduction may be worthwhile.

Converting an asymmetric TSP to a symmetric version is a straightforward procedure that results in the number of cities being doubled. So overcoming that obstacle is possible, however the differentiation between forward and backward setup times poses another problem. This difference means that the city where the travelling salesperson begins and the final city they visit has an impact on the total tour length (due to the backward setup). This complication results in a sub-problem being distinctly different (but closely related) to the TSP. This difference proved difficult to overcome early in our exploration of this formulation. As a result, we focused our attention on the other possible sub-problems and did not pursue a TSP

Table 4.2: Notation for proofs

Notation	Definition
$\mathcal{C}^\nu$	The set of all cuts added to the RMP at iteration ν of the Benders algorithm
$\mathcal{C}^\nu_x$	The set of type-x cuts added to the RMP at iteration ν, where $x \in \{ ng, i, ii, iii, ub, l \}$, *e.g.*, $\mathcal{C}^\nu_{ng}$ denotes the set of nogood cuts
$\mathcal{F}^\nu$	The feasible solution space of RMP(ν)
S^ν ($S^{\nu*}$)	A feasible (optimal) solution to RMP(ν), $S^\nu \in \mathcal{F}^\nu$
$[x^\nu_{ik}]$ $([x^{\nu*}_{ik}])$	A feasible (optimal) solution to the assignment decisions of RMP(ν)

formulation further. However, we do encourage the enthusiastic reader to try their hand at such a formulation.

4.5 Cuts

The quality of the Benders cuts formulated from the solution of the sub-problems directly impact the speed with which the Benders decomposition will approach the optimal solution. As such, we considered a range of cuts which can be added to the master problem, after all sub-problems have been resolved.

For this section we introduce new notation detailed in Table 4.2. Using this notation we represent a solution to the master by its assignment and cycle time variables as $S^\nu = \{[x^\nu_{ik}], c^\nu\}$, and the optimal solution is represented as $S^{\nu*} = \{[x^{\nu*}_{ik}], c^{\nu*}\}$.

4.5.1 Nogood Cut

A *nogood* is a partial solution to a problem which prevents a full feasible solution from being obtained. In our case, we considered a partial solution to the full problem to be an assignment of tasks to a single station. If we consider sub-problem k at iteration ν then when $l^\nu_k \not\leq c^\nu$ a nogood cut can be added to the master problem to remove this partial solution from the feasible solution space of RMP(μ), where $\mu = \nu + 1, \nu + 2, \ldots$.

We now present the formulation of the nogood cut added to the RMP at iteration ν and considered for all iterations $\mu > \nu$.

$$\mathcal{C}^\nu_{ng}: \quad c^\nu + 1 - \sum_{k \in K} \sum_{i \in V^\nu_k} (1 - x^\mu_{ik}) \leq c^\mu. \tag{4.19}$$

For this to be a valid cut, we require that the current solution, $S^{\nu*}$, is no longer

possible in all future master problems, *i.e.*,

$$S^{\nu *} \notin \mathcal{F}^{\mu},\ \forall \mu \in \{\ \nu + 1,\ \nu + 2, \dots\},$$

and also that the cut does not remove any globally optimal solutions. As we only add the cut when a sub-problem cannot respect the master's tentative cycle time value, no globally optimal solutions can be removed.

Theorem 1. The proposed nogood cut (4.19) is valid.

Proof. There are two cases to consider: the solution to the RMP(μ) has the same task assignment and a different task assignment to RMP(ν).

a) $[x_{ik}^{\nu *}] = [x_{ik}^{\mu *}]$. In this case the nogood cut (4.19) reduces to

$$c^{\nu} + 1 \leq c^{\mu} \ \Rightarrow \ c^{\nu} < c^{\mu}.$$

Thus, when the assignment is equivalent, the cycle times must be different.

b) $[x_{ik}^{\nu *}] \neq [x_{ik}^{\mu *}]$. Adding $\mathcal{C}_{ng}^{\nu}$ removed at least $S^{\nu *}$ from the feasible solution space of RMP(μ).

In case a), the solution differs by cycle time value and in case b), the solution differs by task assignment, so $S^{\nu *} \notin \mathcal{F}^{\mu}$. Hence, (4.19) is a valid cut. □

To show that the Benders decomposition algorithm will find the global optimal solution, we require Lemma 1.

Lemma 1. The Benders decomposition algorithm terminates in a finite number of iterations when adding a valid cut at each iteration.

Proof. After iteration μ of the algorithm, either all sub-problems can satisfy the current cycle time, or at least one station cannot satisfy c^{μ}. Cases:

a) $\forall k \in K\ l_k^{\mu} \leq c^{\mu}$. The solution to RMP($\mu$) is optimal and we terminate.

b) $\exists k \in K$ such that $l_k^{\mu} \not\leq c^{\mu}$. Add the corresponding cut to all future RMPs.

In case b), since the cut added is valid, at least one feasible solution is removed from the master problem. We also know that the RMP has a finite number of feasible integer solutions. Hence, the Benders decomposition algorithm will terminate in a finite number of iterations. □

4.5.2 First Infer Cut

The *infer* cuts we devised are examples of optimality cuts, where each sequencing sub-problem is solved to optimality at each iteration of the Benders algorithm. By comparing the current optimal solution of the RMP to the optimal solution of a station, we can make an inference about the set of tasks assigned this station. The first kind of inference we make is the simplest of those we considered and can be described as follows.

At iteration ν after all sub-problems have been solved to optimality, assume that $\exists k \in K$ such that $l_k^\nu > c^\nu$, *i.e.*, station k's optimal sequence of tasks cannot respect the master's cycle time. At a later iteration, μ, if we have the same assignment of tasks to station k, then we can infer that the master's cycle time must be at least the optimal load of station k found previously at iteration ν, l_k^ν. This inference can be formulated as the following implication:

$$\left([x_{ik}^{\nu *}] = [x_{ik}^{\mu *}]\right) \;\Rightarrow\; (l_k^\nu \leq c^\mu).$$

We can model this implication as a disjunctive constraint in the RMP. The infer cut found at iteration ν for a given station k and added to RMP(μ) where $\mu > \nu$, is formulated as follows,

$$\mathcal{C}_i^\nu : \quad l_k^\nu - M \sum_{i \in V_k^\nu} (1 - x_{ik}^\mu) \leq c^\mu. \tag{4.20}$$

We note that to capture this disjunction in the RMP, a big-M value was required. As the master problem is solved using a MIP solver, this cut could have undesirable effects on the linear relaxation.

We now prove that the infer cut (4.20) is valid. The proof proceeds similarly to the proof of the nogood cut above.

Theorem 2. The proposed infer cut (4.20) is valid.

Proof. Let k be the station which caused the infer cut to be added in iteration ν, *i.e.*, $l_k^\nu > c^\nu$. There are two cases to consider: when RMP(μ) has the same task assignment and a different task assignment to RMP(ν).

a) $[x_{ik}^{\nu *}] = [x_{ik}^{\mu *}]$. The infer cut (4.20) reduces to

$$l_k^\nu \leq c^\mu.$$

In this case we know the optimal possible sequencing of the tasks assigned station k. Thus we receive a lower bound on c^μ which must respect l_k^ν. The infer cut was added due to $c^\nu < l_k^\nu$ so we know that $c^\nu \neq c^\mu$ in this case.

b) $[x_{ik}^{\nu *}] \neq [x_{ik}^{\mu *}]$. The assignment to station k is different so we can make no inference in this case.

In both cases $S^{\nu *}$ is not the optimal solution to RMP(μ). Hence, $S^{\nu *} \notin F^{\mu}$ and the cut (4.20) is valid. □

4.5.3 Second Infer Cut

This inference cut is similar to the last as we again compare the optimal load values of each station against the current master's cycle time. Any station loads which exceed the current cycle time value cause a cut to be generated. For this infer cut, further reasoning is made about the tasks assigned to a given station, which remove the need of a big-M value.

We introduce the concept of the total *burden* on the station's workload that a given task contributes. Let B^{ν}_{ik} denote an upper bound of the total burden on station k's workload, resulting from assigning task $i \in V$ to k in iteration ν. The burden of a task is a total of its processing time and all setup costs which it creates in the task sequence. To get an upper bound on the burden incurred from assigning task i to station k at iteration ν, we need to calculate the following two values: 1) an upper bound on the setup time of task i in the task sequence of k (denoted Γ^{ν}_{ik}) and 2) a lower bound on the setup time that would occur between the tasks of station k if task i was not assigned to k at iteration ν (denoted γ^{ν}_{ik}). The burden of a task is calculated by

$$B^{\nu}_{ik} = t_i + \Gamma^{\nu}_{ik} - \gamma^{\nu}_{ik},$$

where the upper and lower bounds on the setup times are calculated as follows

$$\begin{aligned} \Gamma^{\nu}_{ik} = \quad & \max \Big\{ \{ \phi_{ji} \mid j \in P^{\phi}_i \} \cup \{ \beta_{ji} \mid j \in P^{\beta}_i \} \Big\} \\ & + \max \Big\{ \{ \phi_{ij} \mid j \in F^{\phi}_i \} \cup \{ \beta_{ij} \mid j \in F^{\beta}_i \} \Big\} \end{aligned} \tag{4.21}$$

$$\gamma^{\nu}_{ik} = \min \Big\{ \{ \phi_{i'j} \mid j \in F^{\phi}_{i'} \} \cup \{ \beta_{i'j} \mid j \in F^{\beta}_{i'} \} \;\Big|\; i' \in V^{\nu}_k \setminus \{i\} \Big\}. \tag{4.22}$$

To account for both setup costs which task i contributes to, the upper bound, Γ^{ν}_{ik}, is the sum of the setup time occurring directly before task i's execution and the setup time directly after. As for the lower bound γ^{ν}_{ik}, if we consider removing task i from the task sequence of station k, then only one setup cost is needed to ensure the task sequence remains unbroken.

Using this upper bound on the burden incurred from task i we can formulate the following infer cut

$$\mathcal{C}^{\nu}_{ii} : \quad l^{\nu}_k - \sum_{i \in V^{\nu}_k} B^{\nu}_{ik}(1 - x^{\mu}_{ik}) \leq c^{\mu}. \tag{4.23}$$

We can interpret this as follows. Suppose that a cut of the form (4.23) was added after iteration ν due to the load of station k begin greater than c^{ν}. Later in iteration μ, for each task $i \in V^{\nu}_k \setminus V^{\mu}_k$ we subtract an upper bound on the total burden, B^{ν}_{ik},

incurred from assigning i to k. So for each task no longer assigned to station k, the bound on the cycle time c^μ is relaxed.

We prove the correctness of this upper bound on the burden in Lemma 2. We note here that the insight of this proof has been adapted from work by Tran and Beck (2012).

Lemma 2. B_{ik}^ν is a true upper bound on the burden incurred from assigning task i to station k in iteration ν.

Proof. To show that this lower bound on c^μ, where $\mu > \nu$, does not remove any globally optimal solution from $\mathcal{F}^\mu$, we prove that B_{ik}^ν is a true upper bound on the contribution of a task to an optimal schedule. We proceed by contradiction, assuming there is a schedule that violates the lower bound on the cycle time.

Given a set of tasks V_k^ν assigned to station k in the current iteration, let l_k^ν be the optimal station workload. Define two disjoint subsets given by $\bar{V}_k^\mu \cup \hat{V}_k^\mu = V_k^\nu$. Assume that in a subsequent iteration μ, the set of tasks $\bar{V}_k^\mu$ are assigned station k. The minimal workload for station k is now $\bar{l}_k^\mu$. Thus, contrary to our theorem, we have that

$$\bar{l}_k^\mu < l_k^\nu - \sum_{j \in \hat{V}_k^\nu} B_{ik}^\nu. \tag{4.24}$$

Let $Seq(\bar{V}_k^\mu)$ denote the sequence of tasks in station k corresponding to $\bar{V}_k^\mu$. It is possible to construct a sequence containing all tasks of V_k^ν by placing all tasks in $\hat{V}_k^\mu$, one-by-one into $Seq(\bar{V}_k^\mu)$.

Let σ_i be the setup time required to insert tasks $i \in \hat{V}_k^\mu$ into $Seq(\bar{V}_k^\mu)$. We know the station load of this constructed schedule to be

$$\bar{l}_k^\mu + \sum_{i \in \hat{V}_k^\mu} t_i + \sigma_i.$$

This is a sequence of all tasks $i \in V_k^\nu$ and must have a total workload greater than or equal to l_k^ν. However, $t_i + \sigma_i \leq B_{ik}^\nu$, so we know the following is true

$$\sum_{i \in \hat{V}_k^\mu} t_i + \sigma_i \leq \sum_{i \in \hat{V}_k^\mu} B_{ik}^\nu$$

Thus we receive

$$l_k^\nu \leq \bar{l}_k^\nu + \sum_{j \in \hat{V}_k^\mu} B_{ik}^\nu$$

which contradicts inequality (4.24). Hence, B_{ik}^ν is a true upper bound. □

We now prove the validity of this infer cut; again the proof splits into two cases.

Theorem 3. The proposed infer cut (4.23) is valid.

Proof. Let k be the station that caused the cut (4.23) to be added to the RMP after iteration ν. There are two cases to consider: when RMP(μ) and RMP(ν) have the same task assignment and a different task assignment to station k.

a) $\forall i \in V\ x_{ik}^{\nu} = x_{ik}^{\mu}$. The infer cut (4.23) reduces to

$$l_k^{\nu} \leq c^{\mu}.$$

In this case we know the optimal possible sequencing of the tasks assigned station k. Thus we receive a lower bound on c^{μ} which must respect l_k^{ν}. The infer cut was added due to $c^{\nu} < l_k^{\nu}$ so we know that $c^{\nu} \neq c^{\mu}$ in this case.

b) $\exists i \in V$ such that $x_{ik}^{\nu} \neq x_{ik}^{\mu}$. The bound we infer on the cycle time c^{μ} is relaxed by B_{ik}^{ν}. From Lemma 2, we know that B_{ik}^{ν} is a true upper bound on the burden of task i. Thus, this bound does not remove any globally optimal solution from $\mathcal{F}^{\mu}$.

In the first case the cycle time must be different while in the second the assignment is different, so $S^{\nu *} \notin \mathcal{F}^{\mu}$. Hence, the cut (4.23) is valid. □

4.5.4 Third Infer Cut

The final type of inference cut we explored is again similar to the previous two. The second infer cut took into account the effect of removing a task from a station's task sequence. Whereas this Benders cut reasons about the effect of inserting an additional task into the task sequence of a station.

When considering adding a task to a station's task sequence, we again aim to quantify the total workload burden that this additional task brings. However, to get a correct lower bound on the cycle time, we instead seek a lower bound on the burden incurred from an additional task. Let b_{ik}^{ν} denote a lower bound on the burden incurred from assigning task $i \in V \setminus V_k^{\nu}$ to station k in iteration ν. Note that the burden of an additional task is only defined for tasks not already assigned to station k. The lower bound on the burden is defined as follows,

$$b_{ik}^{\nu} = t_i + \bar{\gamma}_{ik}^{\nu}$$

The lower bound on the setup time is calculated slightly differently here compared to equation (4.22); this altered calculation is as follows

$$\begin{aligned} \bar{\gamma}_{ik}^{\nu} = \quad & \min \left\{ \left\{ \phi_{ji} \mid j \in P_i^{\phi} \right\} \cup \left\{ \beta_{ji} \mid j \in P_i^{\beta} \right\} \right\} \\ & + \min \left\{ \left\{ \phi_{ij} \mid j \in F_i^{\phi} \right\} \cup \left\{ \beta_{ij} \mid j \in F_i^{\beta} \right\} \right\}. \end{aligned} \tag{4.25}$$

This alteration is due to the fact that when inserting a new task into a station's task sequence, the newly inserted task becomes part of two setup costs. We must take the minimum of both.

Further to this alteration, the upper bound on the burden of a task being removed from a station's task sequence is changed. We now have the following

$$\bar{B}^{\nu}_{ik} = t_i + \Gamma^{\nu}_{ik}.$$

For this infer cut, we no longer take into account the setup cost required to ensure the task sequence remains unbroken. Note, this will result is a strictly weaker lower bound being inferred on the cycle time, as $\bar{B}^{\nu}_{ik}$ is strictly weaker than B^{ν}_{ik}.

Our final inference cut is formulated as follows

$$\mathcal{C}^{\nu}_{iii}: \quad l^{\nu}_{k} - \sum_{i \in V^{\nu}_{k}} \bar{B}^{\nu}_{ik}(1 - x^{\mu}_{ik}) + \sum_{i \in V \setminus V^{\nu}_{k}} b^{\nu}_{ik} \cdot x^{\mu}_{ik} \leq c^{\mu}. \tag{4.26}$$

Given that $\bar{B}^{\nu}_{ik}$ is a weaker upper bound than B^{ν}_{ik}, it must be a true upper bound. The proof that b^{ν}_{ik} is a true lower bound is roughly equivalent to the proof of Lemma 2 and is omitted. We now provide a sketch of the proof for this cut's validity, due to it being similar to the previous proofs.

Theorem 4. The proposed infer cut (4.26) is valid.

Proof. Let k be the station that caused the cut (4.26) to be added to the RMP after iteration ν.

Similar to the second inference cut, the proof separates into two cases: one where in some subsequent iteration μ, station k has the same assignment and one where it differs by at least one task. Along the same lines as before, the former case results in a new cycle time value, while the latter case results in a new assignment.

So in both cases we have that $S^{\nu *} \notin \mathcal{F}^{\mu}$. Hence, the cut (4.26) is valid. □

4.5.5 Global Bound

We now devise a global upper bound on the cycle time of the problem. Given that it is generated in a similar way to the cuts and also allows us to devise a logic cut in the following section, we include its explanation here. As with the inference cuts, this bound requires the optimal solution of all sub-problems to be found.

After all sub-problems of iteration ν have been solved to optimality we can compare the values of l^{ν}_{k}, $\forall k \in K$. Suppose that at least one station's optimal task sequence cannot satisfy the cycle time of RMP(ν). Although the master's tentative cycle time cannot be respected, the full assignment $[x^{\nu}_{ik}]$ together with each feasible task sequence found by the sub-problems constructs a globally feasible solution to the SUALBSP-2. We can calculate the cycle time of this feasible solution in the

usual way by taking the maximum load among all stations. The cycle time value of this feasible solution gives us a global upper bound on the cycle time of any future solution.

We can formulate this global bound found at iteration ν, and applicable to any subsequent iteration $\mu > \nu$, as follows

$$\mathcal{C}_{ub}^{\nu} : \quad c^{\mu} \leq \max\{\, l_k^{\nu} \mid k \in K \,\}. \tag{4.27}$$

This upper bound is calculated using the objective value of a globally feasible solution, and thus must be a true upper bound on the cycle time of SUALBSP-2.

This bound can not only be added to all future master problems, but also to any sub-problem as an upper bound on the station's total workload. Notably, this may lead to a sub-problem returning infeasibility if the optimal task sequence cannot respect this global bound. This slightly complicates how we can generate Benders cuts so we provide some brief reasoning about this now.

Let Ω^{ν} denote the current best upper bound on the cycle time found using (4.27) at iteration ν. When constraining sub-problem k's objective value by this global bound at iteration μ, there are three possible cases:

1. $l_k^{\mu} \not\leq \Omega^{\nu}$. The sub-problem cannot respect the upper bound and infeasibility is returned. In this case, we are only able to generate feasibility cuts.

2. $c^{\mu} < l_k^{\mu} \leq \Omega^{\nu}$. The station's load can respect the upper bound but not the current master's solution. Here we can generate optimality cuts as the sub-problem's optimal solution is known.

3. $l_k^{\mu} \leq c_{\mu}$. The sub-problem satisfies the current master's cycle time and so no cuts are generated.

In the following section we devise a new type of feasibility cut which can take advantage of the new information that this global upper bound provides.

4.5.6 Logic Cut

Generally, a *logic* cut employs reasoning to remove a set of similar solutions from the feasible solution space of the relaxed master. Any infer cut could also be classified as a logic cut, however the Benders cut we present in this section is not formulated as an inference. As such we label it with a more general classification.

The global bound devised in the previous section resulted in sub-problems possibly returning infeasibility. We can use this information to generate a feasibility cut and add that back to the subsequent master problems. The nogood cuts we devised earlier could be used in this case, although we are able to reason further about the information this global bound provides and in turn receive a more powerful feasibility cut; which we call a logic cut.

Given an upper bound Ω^ν on the cycle time generated by (4.27) in iteration ν, suppose that at a later iteration μ we constrain the load of all sub-problems by this bound. Now suppose that the station load of sub-problem k cannot respect this bound, so we have $l_k^\mu \not\leq \Omega^\nu$. Since this is a globally feasible upper bound on the cycle time, we know that the set of tasks assigned to station k, V_k^μ, necessarily leads to infeasibility. Thus, we can generate an infeasibility cut in the master problem which removes this assignment from all future solutions.

Given a global upper bound found by (4.27) we formulate a logic cut, as follows

$$\mathcal{C}_l^\nu : \quad \sum_{i \in V_k^\nu} (1 - x_{ik}^\mu) \geq 1. \tag{4.28}$$

This constraint ensures that at a later iteration μ, the assignment to station k differs by at least one task from V_k^ν.

We now prove the validity of this logic cut.

Theorem 5. The proposed logic cut (4.28) is valid.

Proof. Suppose that at iteration ν, a cut of the form (4.28) was added due to sub-problem k being unable to satisfy the current best global upper bound, denoted by Ω. Now consider the RMP at a later iteration μ.

Once more, there are two cases to consider: the assignment solution to station k of RMP(μ) is equivalent to that of RMP(ν) and different.

a) $V_k^\nu = V_k^\mu$. Proceed by contradiction. In this case we have

$$\sum_{i \in V_k^\nu} (1 - x_{ik}^\mu) = 0 \not\geq 1.$$

Thus, we have a contradiction due to the logic cut (4.28), so this case does not need to be considered further.

b) $V_k^\nu \neq V_k^\mu$. As the assignment to station k is different than in the optimal solution of RMP(ν), the solution $S^{\nu *}$ is no longer feasible for RMP(μ).

In the only valid case we have that $S^{\nu *} \notin \mathcal{F}^\mu$. Hence, the cut (4.28) is valid. □

4.6 The Algorithm

We now briefly describe how the algorithm used to iterate between the relaxed master problem and the m sub-problems. A flowchart of the overall Benders decomposition is provided in Figure 4.2, and in Figure 4.3 we give a flowchart of how an individual optimality sub-problem is processed.

4.6.1 The Relaxed Master Problem

The algorithm begins by initializing the current input instance of the SUALBSP-2. For this input, RMP(1) is constructed as a MIP of the form presented in Section 4.3 with no Benders cuts, *i.e.*, $\mathcal{C} = \emptyset$. This problem is solved using the Gurobi optimizer and if possible the optimal solution, S^{1*}, is found. If the relaxed master is infeasible, then the current instance is infeasible and so we terminate. Otherwise we calculate the current Benders gap value using c^1 and check if it is within a predefined tolerance, ε. Unless stated otherwise, we took $\varepsilon = 0$ and so did not terminate until the globally optimal solution to the instance was found.

The Benders gap can be interpreted as a measure of the disparity between the solutions to the sub-problems and the current master. The Benders gap is defined as follows

$$\text{Benders gap} = 100 \cdot \frac{\Omega - c^{\mu}}{c^{\mu}}, \tag{4.29}$$

where Ω is the current best upper bound on the cycle time and μ is the current iteration, *i.e.*, here $\mu = 1$. The value of Ω is initialized as the big-M value presented in Table 3.1 from the previous Chapter. We note that for the Benders gap to ever have the possibility of approaching 0%, the upper bound Ω must be calculated in a more intelligent way than just using big-M. Thus, we only check if the Benders gap ≈ 0 when the global upper bound devised in Section 4.5.5 is considered.

Only in the trivial case when the optimal solution is to assign all tasks to a single station, will the Benders gap value be 0 after the first iteration. So this test is skipped if $\mu = 1$.

After all sub-problems have been solved, we test if their load values can all satisfy the current cycle time. If the cycle time is not violated then the master problem has found a globally optimal solution and we terminate. Otherwise, at least one station sub-problem caused a Benders cut to be generated. When applying the global upper bound from Section 4.5.5 to the load of each sub-problem, we must check if any of the Benders cuts added were a feasibility cut. If at least one feasibility cut was generated then the current feasible assignment must result in a strictly worse cycle time value then the current Ω and thus we do not update the global upper bound in this case. When only optimality cuts are created, we test if the current globally feasible assignment leads to a lower cycle time than Ω and update it if necessary.

The generated cuts are added to the RMP and we re-solve. This iterative process continues until a globally optimally solution is found.

4.6.2 The Sub-Problem

Once the assignment of tasks to the m stations is found, all sub-problems are solved. Unless stated otherwise, we will only consider optimality sub-problems for the description given here. We now refer the reader to Figure 4.3 for an overview of how

Figure 4.2: Flowchart of our Benders decomposition algorithm

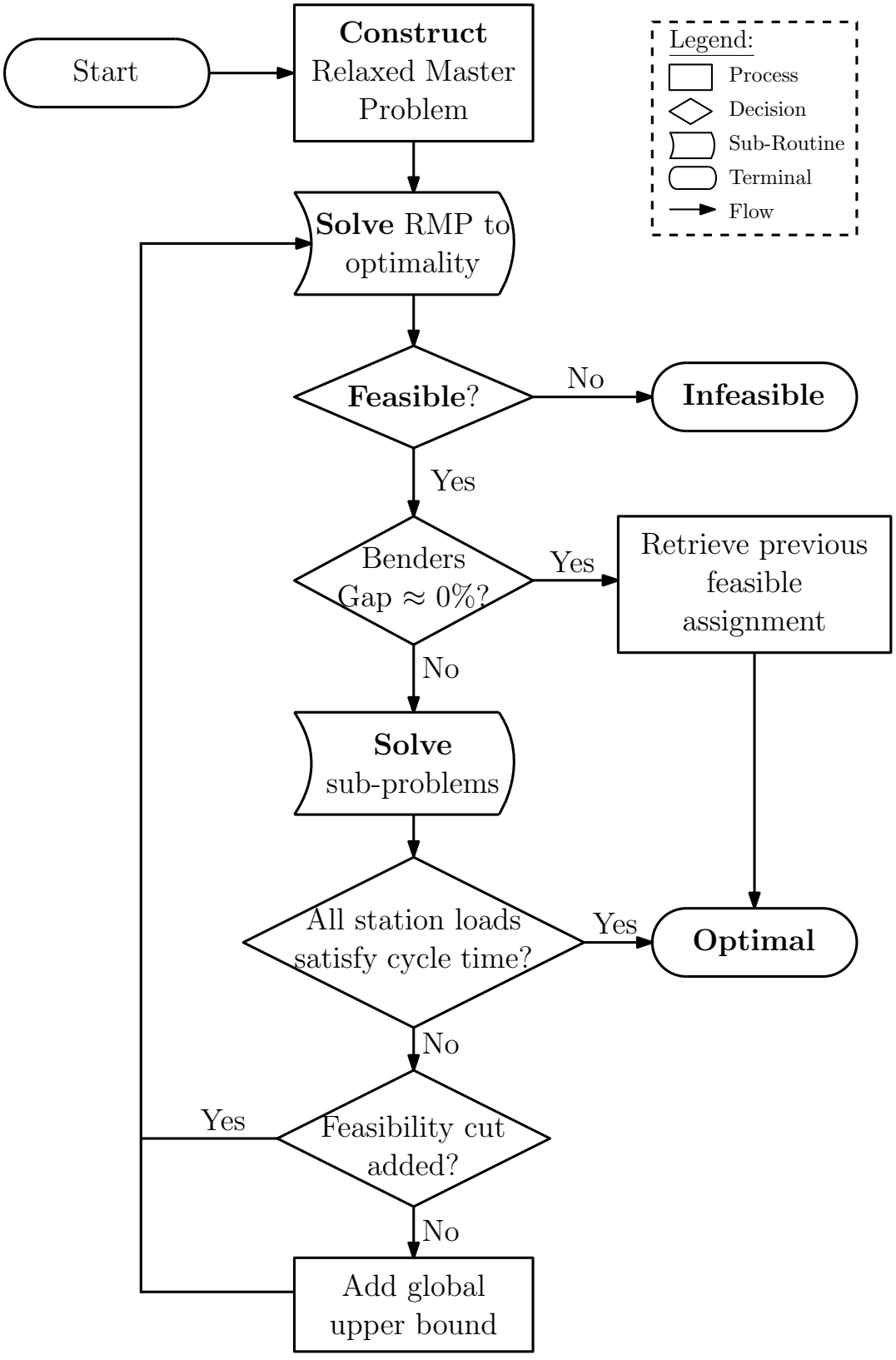
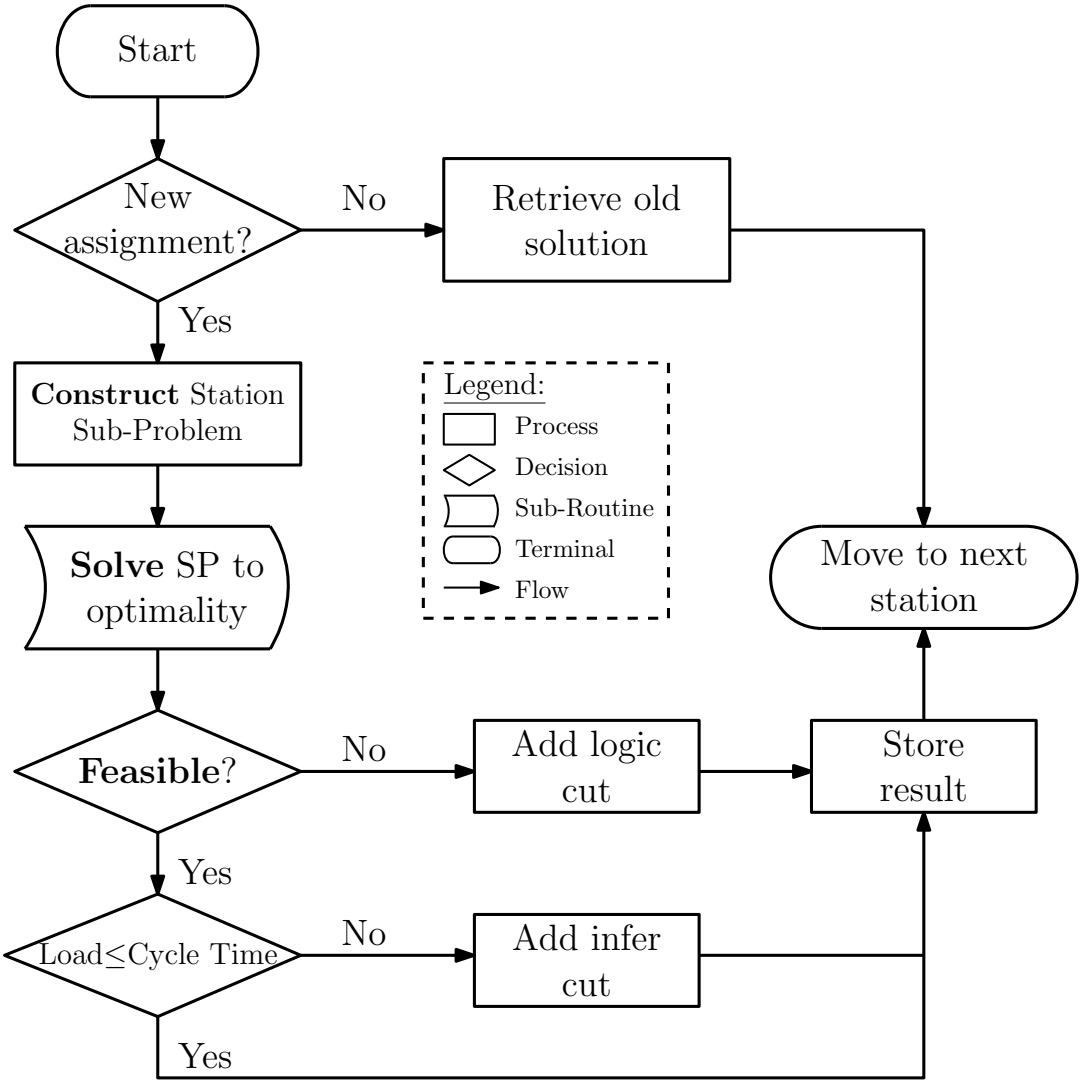


Figure 4.3: Flowchart of an optimality sub-problem



a single sub-problem is processed. We first check if the current assignment to the station has been solved previously or not. All previous solutions are stored together with their optimal station load values, so if we find an assignment which has already been processed, then we retrieve the previous solution and move to the next sub-problem. In the case of a new assignment of tasks, the sub-problem is constructed and solved to optimality.

When a sub-problem is found to be infeasible a logic cut is added; when the load does not satisfy the current cycle time an infer cut is added; and if the cycle time is satisfied then no cut is generated. In all three cases, the resulting solution is stored with the current assignment for future referral. The process continues on to the next sub-problem until none remain and then returns to the master.

4.6.3 Heuristic Approach

We conclude this chapter by mentioning a minor modification that can be made to the above procedure. We only considered a tolerance of 0% on the Bender gap, however we could instead take $\varepsilon > 0$. This would result in our exact solution methodology becoming a heuristic method able to guarantee a pre-defined quality on the assignment and schedule found. For example, if we set $\varepsilon = 0.05$, then the algorithm will terminate once a globally feasible solution is found which is within 5% of the optimal. The modification of this parameter offers this logic-based Benders decomposition a higher degree of flexibility. It is possible that reaching a gap of 5% can be done very efficiently but finding the global optimal is intractable. In practice, the industry representative may only require feasible solutions of this quality as the sub-optimal solution could still be superior to their current assembly line design.

5 Computational Experiments

All experiments were run on a personal computer with an Intel i5 5575R CPU (four cores each with 2.8GHz clock speed) and 16GB of main memory. The scripts for the MIPs and the primary script used to iterate through the Benders decomposition algorithm was written in Python 3.4. We employed Gurobi 7.0.2 as the solver used for all MIP formulations. As mentioned previously, `Chuffed` was the solver we chose to solve all constraint programs with. MiniZinc version 2.1.2 was used to compile the models into the `Chuffed` FlatZinc format. Unless stated otherwise, all tests were run with a time limit of 30 minutes (1800 seconds).

We chose to only test FSBF-2 and SCBF-2 for the computational experiments of the previous work. This is a result of our encoding of the SSBF-2 model returning infeasible solutions in our preliminary tests. Due to time constraints we were unable to discern whether the problem was our encoding of the model or if the SSBF-2 is ill-formed.

We now provide a brief outline of the computational results reported in this chapter. First a description of the data used for testing is given in Section 5.1. As the MIPs presented in Esmaeilbeigi *et al.* (2016) have been untested, we investigate the best combination of valid inequalities to include for these models in Section 5.2. In Section 5.3 we report on the best combination of Benders cuts to use for our decomposition. We provide a detailed investigation into the best configuration of the CP sub-problem in Section 5.4. Once the superior sub-problem formulation is determined, we close in Section 5.5 of this chapter by reporting on the results of our best solution methodology on a set of benchmark instances.

5.1 Data

The primary dataset available for the SUALBSP is the SBF generated by Scholl *et al.* However, this dataset was created for the type-1 case in particular and so is not immediately applicable to the SUALBSP-2. The SBF is divided into two halves, the SBF1 and SBF2, where each half consists of a total of 1076 instances. We first provide a brief explanation of how SBF1 was generated.

The basis of the SBF1 are a set of 269 instances for the SALBP-1 containing between 4 and 297 tasks. When creating instances for the SUALBSP-1, the cycle time, task times and precedence relations are all transferred from the original instances. We also require values of the forward and backward setup times *a priori*, so to create these the authors consider the average processing time among all tasks, denoted t_{av}. In practice the setup times are smaller than the execution time of a task, so a parameter α is used when randomly generating the setups. An upper bound on the possible setup is defined as $\alpha \cdot t_{av}$ where $\alpha \in \{ 0.25, 0.50, 0.75, 1.00 \}$. For example, when $\alpha = 1.00$, the largest possible setup time is 50% of the average task time. The 269 input instances are duplicated for each α value and so 1076 instances of SUALBSP-1 are generated for SBF1.

The SBF2 was generated in almost exactly the same way as SBF1. However in this case, care was taken when randomly generating the setup times to ensure that at least one optimal solution to the original SALBP-1 instance is still feasible in the new SUALBSP-1 instance. This solution will necessarily be the optimal solution of the SUALBSP-2 as noted by Scholl *et al.* When storing the instance data, the authors also recorded the optimal number of stations for the original SALBP-1 instance.

The data we chose to run all our tests on was adapted from the SBF2 dataset. As the optimal solutions were recorded, we could take the optimal number of stations as input and then solve for the cycle time. This simple change allowed 1076 instances for testing our formulations. Esmaeilbeigi *et al.* (2016) considered a subset of the SBF2 instances which they divided into 3 classes. We considered the same subset of instances for the type-2 problem which are detailed in Table 5.1. In this table we give the number of tasks, stations, precedence relations and order strength of the precedence graph. In total 396 instances were considered in our tests.

The order strength of an instance is a measure of its complexity where higher values correspond to more difficult instances. However, we note that in the extreme, when $OS = 1$, there is only one precedence-feasible task sequence and thus the problem becomes trivial.

5.2 MIP Formulations

We first report on how the MIP formulations presented in Chapter 3 performed. In Table 5.2 and 5.3 we give the results of FSBF-2 and SCBF-2 respectively. Each is tested on each combination of the valid inequalities from Section 3.4 against

Table 5.1: Dataset summary

Class	Creator	#Inst.	n	m	$\|E\|$	Order Strength (OS)
1		108	7-21	2-8	6-27	
	`mertens`	6	7	2-6	6	medium (52.40)
	`bowman8`	1	8	5	8	high (75.00)
	`jaeschke`	5	9	3-8	11	high (83.33)
	`jackson`	6	11	3-8	13	medium (58.18)
	`mansoor`	3	11	2-4	11	medium (60.00)
	`mitchell`	6	21	3-8	27	high (70.95)
2		112	25-30	3-14	32-40	
	`roszieg`	6	25	4-10	32	high (71.67)
	`heskia`	6	28	3-8	40	low (22.49)
	`buxey`	7	29	7-13	36	medium (50.74)
	`sawyer30`	9	30	5-14	32	low (44.83)
3		176	32-58	3-31	38-82	
	`lutz1`	6	32	6-11	38	high (83.47)
	`gunther`	7	35	7-14	43	medium (59.50)
	`kilbrid`	10	45	3-10	62	low (44.60)
	`hahn`	5	53	4-8	82	high (83.82)
	`warnecke`	16	58	14-31	70	medium (59.10)
Overall		396	7-58	2-31	6-82	

all 3 classes of the subset of instances from SBF2. The following datapoints are summarized for each test: the number of nodes explored by Gurobi's B&B tree, the average gap for all instances, the number of instances where no feasible solution was found, the fraction of instances found to optimality out of the whole class, the percentage of optimal instances and finally the average runtime taken by Gurobi.

By inspecting the data we find that using the valid inequality (3.46) without (3.48) is the best formulation of FSBF-2. As for SCBF-2, we found that the best formulation was when neither of the two valid inequalities were included in the program.

For both the MIPs, optimality was proven for almost none of the instances of class 2 and 3. As a result, all the average runtime values are close to the time limit of 30 minutes. In the case of SCBF-2, no feasible solution was found for 66.5% of the instances in class 3. Even for the smallest class of data, where there is no more than 21 tasks, neither MIP was able to prove optimality for all instances. We note that when calculating the gap value, we only average the instances where a feasible solution was found. With a 30 minute time limit, both MIPs could not achieve an average gap value less than 10% for class 2 and 3.

Table 5.2: FSBF-2 formulations tested on classes 1,2 and 3

Class	Ineqs.	#Nodes	%Gap	#No.s.	#Opt.	%Opt.	Rt.(s)
1	–	40,181	0.56	0	98/108	90.74	172.90
	(3.46)	43,437	0.16	0	**101/108**	93.52	166.57
	(3.48)	50,987	1.14	0	84/108	77.78	628.47
	(3.46),(3.48)	51,734	1.12	0	85/108	78.73	645.02
2	–	171,847	14.81	0	0/112	0.00	1801.08
	(3.46)	168,899	13.76	0	0/112	0.00	1800.66
	(3.48)	205,752	18.04	0	0/112	0.00	1801.52
	(3.46),(3.48)	202,980	17.90	0	0/112	0.00	1801.27
3	–	48,927	19.75	9	7/176	3.98	1763.45
	(3.46)	46,060	19.56	9	7/176	3.98	1759.67
	(3.48)	62,845	25.64	16	0/176	0.00	1800.23
	(3.46),(3.48)	63,002	24.72	15	0/176	0.00	1801.12

Here we tested the full mixed-integer programming approach on the smallest subset of instances of the SBF2. The poor results we have found make it clear that these programs are unsatisfactory exact solution methods for tackling the SUALBSP-2.

5.3 Cuts

The cutting strategy used in the iteration of the Benders algorithm can have a large impact on its overall performance. Even when either the master or sub-problem is exceptionally difficult to solve, if the Benders cuts added to the master allow us to find optimality in very few iterations then the problem can still be tractable.

5.3.1 Nogood Cuts

In the preliminary testing phase, we considered feasibility sub-problems using the nogood cuts presented in Section 4.5.1 of the previous chapter. These results are briefly summarized in Table 5.4. Even on the smallest class 1, the Benders algorithm failed to prove optimality for 66 of the instances tested. Due to this, we felt that further investigation into the use of nogood cuts was not the most fruitful research direction. Note that we did not make use of the possible parallel computing capabilities detailed earlier in Section 4.2.1, where the feasibility sub-problems could be run concurrently. This potential avenue was not explored as the percentage of computation time spent solving the sub-problems was dwarfed by the time spent solving the master problem; as we will see in the coming results.

For all remaining experiments optimality sub-problems were used unless stated

Table 5.3: SCBF-2 formulations tested on classes 1,2 and 3

Class	Ineqs.	#Nodes	%Gap	#No.s.	#Opt.	%Opt.	Rt.(s)
1	–	221,842	1.62	0	**92/108**	86.11	374.88
	(3.46)	244,811	1.97	0	84/108	77.78	403.17
	(3.48)	238,515	1.75	0	86/108	76.93	377.05
	(3.46),(3.48)	215,866	1.75	0	86/108	76.93	380.82
2	–	651,880	18.01	28	0/112	0.00	1800.33
	(3.46)	701,617	19.75	31	0/112	0.00	1800.47
	(3.48)	686,566	18.98	30	0/112	0.00	1801.03
	(3.46),(3.48)	612,442	18.57	31	0/112	0.00	1800.99
3	–	301,055	10.09	109	0/176	0.00	1800.88
	(3.46)	326,284	12.20	117	0/176	0.00	1801.00
	(3.48)	324,878	12.15	112	0/176	0.00	1801.21
	(3.46),(3.48)	319,908	12.06	117	0/176	0.00	1801.01

Table 5.4: Benders algorithm tested on class 1 with nogood cuts

Cuts	Class	#No sol.	#Opt.	%Opt.	Runtime(s)
$\mathcal{C}_{ng}$	1	66	42/108	38.89	1456.57

otherwise.

5.3.2 Cutting Procedure

In Tables 5.5 and 5.6 we provide the primary comparison of the possible cutting strategies that we considered. The first table compares the 9 possible cut combinations on the smallest class of data (class 1). The variation in the results of Table 5.5, makes it difficult to discern a clear best cutting procedure. So we further tested 6 cut combinations on class 2 in the Table 5.6. The choice of solving technology for the sub-problems will only effect the time taken to solve all scheduling problems but not the cutting procedure. Thus we arbitrarily chose to test all combinations of cuts using MIP sub-problems.

The two tables comparing the cut combinations contain the following data: average number of nodes explored by Gurobi's B&B tree (total combined between the RMP and sub-problems), average number iterations of the Benders algorithm, average number of cuts generated, average Benders gap, total number of instances where no feasible solution was found, percentage of the instances proved optimal and the average runtime of all instances.

By inspecting the results of Table 5.5 it is difficult to reason about which cutting

Table 5.5: Benders cutting strategies compared on class 1

Cuts and Bound										
$\mathcal{C}_i$	$\mathcal{C}_{ii}$	$\mathcal{C}_{iii}$	$\mathcal{C}_l$	$\mathcal{C}_{gb}$	#Nodes	#Its.	$\lvert\mathcal{C}\rvert$	%Gap	#No.s.	%Opt.	Rt.(s)
✓					4,718	18	35	0.00	0	100.00	5.8
	✓				2,845	15	28	0.00	0	100.00	4.1
		✓			170	4	8	0.00	0	100.00	1.1
✓				✓	3,790	16	32	0.00	0	100.00	2.9
	✓			✓	2,064	12	26	0.00	0	100.00	1.9
		✓		✓	128	3	9	0.00	0	100.00	0.4
✓			✓	✓	766	17	34	0.00	0	100.00	2.6
	✓		✓	✓	896	17	34	0.00	0	100.00	2.8
		✓	✓	✓	56	4	12	0.00	0	100.00	0.5

strategy is superior as all instances were proved optimal for every combination. These results immediately tell us that the Benders decomposition we present here has outperformed the pure MIP formulations previously presented by the literature. We note that when the global bound in not enforced after a globally feasible solution is found, the Benders gap remains substantially large and so it is difficult to measure the convergence of the Benders algorithm. Thus for the remaining experiments we run, the global bound will always be used unless stated otherwise. The 3$^{\text{rd}}$ version of the inference cut ($\mathcal{C}_{iii}$) appears to be a key ingredient in the Benders algorithm terminating efficiently, but to confirm this suspicion we test our solution methodology on class 2.

In Table 5.6 we can see that class 2 proved more challenging for the Benders decomposition to solve the problem as in many cases optimality was proven for only $\sim 30\%$ of the instances. These results make it clear that $\mathcal{C}_{iii}$ is the strongest infer cut as it is able to solve almost twice as many instances to optimality that the other 2 infer cuts. Further to this, it needed half as many iterations and similarly half as many cuts in total. For the instances where the optimal solution was not found, $\mathcal{C}_{iii}$ provided the lowest Benders gap values as well.

We must also examine whether including the logic cut, $\mathcal{C}_l$, provided a benefit. When considering the logic cut, a global bound on the cycle time is instituted as an upper bound on the load of each station's sub-problem, possibly leading to feasibility sub-problems, rather than optimality. The logic cut was formulated to provide the RMP with detailed information about the assignment variables whilst also reducing the time needed to solve a sub-problem; however it also has a notable drawback. When a logic cut is generated by a sub-problem, that sub-problem has not been solved to optimality. This results in us not being able to generate an optimality cut for this sub-problem, such as $\mathcal{C}_{iii}$. The results of Table 5.6 indicate that on average, not being able to perform an infer cut has not outweighed the beneficial information

Table 5.6: Benders cutting strategies compared on class 2

Cuts and Bound										
$\mathcal{C}_i$	$\mathcal{C}_{ii}$	$\mathcal{C}_{iii}$	$\mathcal{C}_l$	$\mathcal{C}_{gb}$	#Nodes	#Its.	$\lvert\mathcal{C}\rvert$	%Gap	#No.s.	%Opt.	Rt.(s)
✓				✓	1,522k	99	396	11.63	1	31.25	1339.7
	✓			✓	1,430k	99	399	10.79	1	32.14	1348.5
		✓		✓	1,024k	47	196	8.42	1	**60.71**	959.9
✓			✓	✓	918k	112	439	11.38	1	33.04	1318.4
	✓		✓	✓	964k	114	443	11.33	1	31.25	1342.3
		✓	✓	✓	898k	68	225	9.03	1	58.04	956.8

provided by the logic cut.

Hence, for all remaining experiments we used the infer cut $\mathcal{C}_{iii}$ in combination with the global bound $\mathcal{C}_{gb}$ as the cutting procedure, unless stated otherwise.

We mention here that combining different types of infer cuts together in a single cutting procedure was not explored. This is because every inference made by $\mathcal{C}_i$ is contained within $\mathcal{C}_{ii}$ and every inference made by $\mathcal{C}_{ii}$ is similarly contained within $\mathcal{C}_{iii}$. Thus combining these infer cuts will not lead to a substantial reduction in runtime.

5.4 CP Sub-Problem

The CP solver `Chuffed` provides the user with numerous input parameters to tailor the propagation and search procedure to the problem at hand. In this section, we report on the comparisons between the model formulation and search strategies in order to find the best suited configuration for the SUALBSP-2 instances.

As we are only concerned with optimizing the sub-problems' solving time, the runtime values we present here will only average the time used by the CP model, *i.e.*, excluding the runtime of the master problem. Similarly, this also applies to the average number of nodes explored by the CP sub-problems..

5.4.1 Model Formulation

Two main types of CP models were considered, so we first check which of these is superior before moving on to test other input parameters. CP_1 represents the model detailed in Section 4.4.2.1 from the previous chapter. CP_2 represents the same model but with the additional decision variable σ_{ij} and the `cumulative` global constraint.

In Table 5.7, we compare these two models against each other on the class 1

Table 5.7: CP formulations compared on classes 1 and 2 when $\alpha = 1.00$

Class	Model	#SP Nodes	%Gap	%Opt.	SP Runtime(s)
1	CP_1	32,755	0.00	100.00	3.83
	CP_2	29,225	0.00	100.00	**2.84**
2	CP_1	2,768,871	15.45	25.00	386.71
	CP_2	2,503,183	14.03	25.00	**329.60**

and class 2 instances where α is fixed to 1.00. The search strategy used for these tests was simply the default search of `Chuffed`. These tests indicate that there is a reduction in sub-problem runtime when the `cumulative` constraint is used in the CP formulation. For all remaining experiments where CP sub-problems were used, CP_2 was the formulation utilized unless stated otherwise.

5.4.2 Search Strategy

Earlier in Section 4.4.2 of the previous Chapter, we detailed a range of possible search procedures which could be employed when solving our constraint program. In Table 5.8 we present the results of testing the CP sub-problems on a selection of these procedures. The names of the searches refer to the following procedures: `def` is the default search, `start` is the basic search on start time variables, `start_ff_Then_sigma` is a sequential search of start then σ variables. Four priority searches were considered which only differ by the variable selection strategy used to choose the next sequential procedure.

From the results of Table 5.8, we can immediately conclude that the default search is inferior to the search strategies we have tailored for the SUALBSP-2. Although, among the search procedures we devised, there is very little variations in the results. Each procedure was able to find the optimal solution for the same number of instances, while the runtime was very similar. The basic search `start` needed to explore the fewest number of nodes in the search tree, however the priority search `priority_ff` provided the best average runtime. For all remaining tests where CP sub-problems were utilized, the priority based search strategy using the `first_fail` variable selection was used.

5.5 Benchmark Experiments

We close this chapter by comparing the best configuration of our logic-based Benders decomposition against the results of the MIP formulations for all benchmark instances from the subset of SBF2 we have used. Before presenting this final result,

Table 5.8: CP search strategies compared on class 2 when $\alpha = 1.00$

Class	Search	#SP Nodes	%Gap	%Opt.	SP Runtime(s)
2	`default`	2,503k	14.03	25.00	329.60
	`start`	1,884k	8.97	42.86	246.57
	`start_ff_Then_sigma`	2,391k	9.69	42.86	254.23
	`priority_io`	2,448k	9.12	42.86	253.70
	`priority_sm`	2,408k	9.06	42.86	252.91
	`priority_sml`	2,470k	9.04	42.86	252.63
	`priority_ff`	2,426k	8.90	42.86	**245.74**

Table 5.9: MIP and CP sub-problem formulations compared on class 2

Class	Sub-Problem	#SP Nodes	%Gap	%Opt.	SP Runtime(s)
2	SP-MIP	630k	8.42	60.71	173.05
	SP-CP	2,489k	8.21	61.61	**165.26**

we must first decide on the best sub-problem formulation for our Benders decomposition.

Earlier in Section 2.3.3, we gave a brief description of the CP solver, `Chuffed`, we chose to utilize when tackling the constraint programs we formulated. As mentioned in that section, `Chuffed` does not currently have the ability to be run in parallel. This handicap has hampered its ability to effectively compete with Gurobi, which can easily run in parallel on all four cores of the computer we used for testing.

In Table 5.9 we present a comparison of our Benders decomposition algorithm tested on the class 2 instances when using MIP and CP sub-problems. There are many more nodes explored when using CP sub-problems due to the different search procedure of a CP solver when compared to a MIP solver. The results of the two formulations are similar, but we can conclude that our best configuration of the CP model has provided a reduction in runtime when compared to the MIP formulation.

Now using the best formulation of our Benders decomposition we present the results of our final benchmark experiments on all 3 classes from the subset of the SBF2 instances. These results can be found in Table 5.10 where we provide the average number of iterations, cuts, average gap, number optimal and runtime breakdown. The instances from classes 2 and 3 proved to be more challenging than the smaller instances from class 1. Most notably, the total runtime taken by the Benders algorithm is almost entirely due to the complexity of the RMP.

More cuts are to be expected when the difficultly of the problem is increased, but we also mention that for the instances of class 3 there is also a larger number of stations. This will lead to more cuts being made for every iteration of the algorithm.

Table 5.10: Benchmark results of our final Benders decomposition

Class	#Its.	#Cuts.	%Gap	#Opt.	RMP.Runtime(s)	SP.Runtime(s)
1	3	9	0.00	108/108	0.36 (87.95%)	0.04 (10.98%)
2	47	196	8.21	69/112	781.29 (82.33%)	165.26 (17.42%)
3	92	468	15.76	27/176	1462.01 (88.00%)	173.63 (10.45%)

Table 5.11: Benders decomposition compared to pure MIP formulations

	Benders decomposition				FSBF-2		SCBF-2	
Class	%Gap	#Opt.	%Opt.	Rt.(s)	%Gap	%Opt.	%Gap	%Opt.
1	0.00	108/108	100.00	0.41	0.16	93.52	1.62	77.78
2	8.21	69/112	61.61	948.93	13.76	0.00	18.01	0.00
3	15.76	27/176	15.34	1661.93	19.56	3.98	10.09	0.00

Table 5.11 compares how our final results perform when compared to the pure MIP formulations of the SUALBSP-2. For each class of instances our solution methodology beats the best formulation of the MIPs. In the case of class 2 where the Benders decomposition could not prove optimality for all instances, the average Benders gap at termination is superior to the best average gap found by Gurobi when solving either FSBF-2 and SCBF-2. For class 3, our algorithm could only find the optimal solution for 15.3% of the 176 instances. Also the SCBF-2 model returned superior average gap values for class 3, even though it was unable to find any optimal solutions. Although the gap value of SCBF-2 for the class 3 instances may be misleading, as it only calculated from the 59 of 176 instances where a feasible solution was found.

Our results indicate that the SUALBSP-2 is a difficult problem to solve when the instance size becomes large. However, we can see from these results that decomposing the problem using a logic-based Benders decomposition framework provides the most optimal solutions when compared to the other available exact solution procedures.

6 Conclusion

In this thesis we devised a logic-based Benders decomposition of the SUALBSP-2. The natural division between the assignment and scheduling variables led us to believe that this approach would be well-suited to the problem. Our formulation resulted in a classical SALBP-2 being solved by the master problem while a sub-problem for each station was created. We considered multiple ways of formulating and solving the sequencing problems that arose in each station.

Logic-based Benders decomposition allowed us to formulate a sub-problem as either a MIP or a constraint program and using its inference dual to generate Bender cuts. Employing CP solving technology resulted in marginal improvements to the runtime and solution quality when compared to MIP sub-problems. However the CP solver `Chuffed` was only able to operate on one core while the MIP solver Gurobi was able to utilize all four cores. The inclusion of the global constraint `cumulative` in the CP model and investigating the possible search strategies of the CP solver, both contributed to the CP model outperforming the MIP sub-problem formulation.

Both feasibility and optimality Benders cuts were devised. Solving each sub-problem to optimality allowed a much wider range of cuts to be generated at each iteration of the Benders algorithm. We provided a detailed analysis into the advantages and disadvantages of each possible cutting procedure and after performing experiments, we were able to conclude that combining the global upper bound, $\mathcal{C}_{gb}$, with our strongest inference cut, $\mathcal{C}_{iii}$, resulted in the most effective cutting planes added to the relaxed master problem.

In summary, our decomposition approach was able to solve 100% of the smallest class of instances in an average runtime of 0.41 seconds. For the second class, which contained small-to-medium sized instances, our method was able to solve 61.6% of

the instances, while also finding superior gap values compared to pure MIP models solved with Gurobi. In the case of largest class we considered, only 15.3% of optimal solutions were proven in the 30 minute time limit. Of the 396 instances we tested which had up to 58 tasks and 31 stations, the Benders decomposition was able to solve 51.5% to optimality.

6.1 Future Work

The work and results we have reported on in this thesis stand as only the very beginning of the research which could be done into decomposition approaches to the SUALBSP. The logic-based Benders decomposition we devised can be viewed simply as a framework for a solution procedure to this problem. As such this framework presents a range of possible future research directions; some of which we will now detail.

Firstly, applying our Benders decomposition to the type-1 problem of SUALBSP is a straight forward extension. This will result in m sub-problems needing to be solved where m is a variable, while each station has an upper bound on its load. We can again view these sequencing problems as asymmetric TSPs, however now not only is the number of cities within each TSP unknown, but the number of TSPs to be solved is also a decision. As each sub-problem has a pre-defined upper bound, feasibility cuts may be better suited in the type-1 case.

The final results we presented in the previous chapter tell us that the vast majority of solve time in our decomposition is due to the relaxed master problem. We only considered a full mixed integer programming approach for the master however this was not exclusively necessary. Any procedure which can return an assignment of tasks to the stations and provides a lower bound on the cycle time would be applicable. Thus, developing heuristic procedures to replace our current formulation of the RMP may be a fruitful direction of investigation. In fact, it is very possible that implementing such a procedure — even if it is relatively simple — could significantly improve the results found here.

In this thesis, we considered few ways of formulating the MIP version of the sub-problems. Further investigation into valid inequalities of this program or simply a different set of decision variables could lead to better results. Regarding the CP sub-problems, the lack of parallelization of the `Chuffed` solver hindered its capabilities somewhat. So when this feature becomes available, re-running the experiments to check the reduction in runtime would provide a useful comparison between different solution technologies.

Finally, we emphasise the limited amount of research which has been done into the SUALBSP. This variant of the ALBP presents a non-trivial layer of complexity with a set of scheduling decisions, however this consideration is applicable to a wide variety of assembly lines. Thus, including setup costs in other well-known variants of the ALBP would lead to a range of possible research directions.

A

Appendix

Algorithm 1 Branch-and-Bound algorithm for a minimization problem

$Incumbent = \infty$
Initialize the tree with root node, n_0, and solve the relaxed LP
$Nodes \leftarrow \{n_0\}$ ▷ *The set of nodes to be fathomed*

while $Nodes \neq \emptyset$ **do**
    Select a node $n \in Nodes$
    Choose a relaxed variable, x_i, in the LP solution
    Branch on x_i, splitting its domain into two subsets, creating nodes n_a, n_b
    Solve the two relaxed LPs of each node.
    for $n_{new} \in \{n_a, n_b\}$ **do**
        if $LPsol$ is integer **then**
            if $LPsol < Incumbent$ **then**
                $Incumbent \leftarrow LPsol$ ▷ *Found new best solution*
            else if $LPsol \geq Incumbent$ **then**
                $Nodes \leftarrow Nodes \cup \{n_{new}\}$
        else if $LPsol$ is not integer **then**
            if $LPsol \leq BestBound$ **then**
                $Nodes \leftarrow Nodes \cup \{n_{new}\}$
            else if $LPsol > BestBound$ **then**
                $Nodes \leftarrow Nodes \setminus \big(n_{new} \cup Children(n_{new})\big)$ ▷ *Pruning*
                Continue to next node
        if $LPsol < BestBound$ **then**
            $BestBound \leftarrow LPsol$

return $Incumbent$

```
% Constraint Programming model for a station's sub-problem
include "cumulative.mzn";
include "disjunctive.mzn";
include "redefinitions.mzn";
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% INSTANCE INITIALISATION
int: nTasks;
int: nPrecs;
int: maxLoad;  % maximum makespan
set of int: TASK;
set of int: PREC = 1..nPrecs;
set of int: TIME = 0..maxLoad;
array[TASK] of int: dur; % duration
array[TASK] of set of TASK: suc; % set of successors
array[TASK,TASK] of int: forwSU; % forward setup times
```

```
array[TASK,TASK] of int: backSU; % backward setup times
array[TASK] of set of TASK: followForw; % allowed followers in forward load
array[TASK] of set of TASK: followBack; % allowed followers in backward load
array[TASK] of set of TASK: precedeForw; % allowed preceders in forward load
array[TASK] of set of TASK: precedeBack; % allowed preceders in backward load
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% DECISION VARIABLES
array[TASK] of var TIME: s; % start time
array[TASK,TASK] of var TIME: spair; % start time pairings
array[TASK,TASK] of var bool: y; % forward direction following
array[TASK,TASK] of var bool: z; % backward direction following
var TIME: load;  % load
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% CONSTRAINTS
% Only one follower in either station load direction
constraint
  forall (
    i in TASK
  )(
      sum( j in followForw[i] )( y[i,j] )
    + sum( j in followBack[i] )( z[i,j] )
    == 1
  );
% Only one preceder in either station load direction
constraint
  forall (
    j in TASK
  )(
      sum( i in precedeForw[j] )( y[i,j] )
    + sum( i in precedeBack[j] )( z[i,j] )
    == 1
  );
% Exactly one backward setup
constraint
  sum(
    i in TASK, j in followBack[i]
  )(
    z[i,j]
  ) == 1
  ;
% Precedence constraints
constraint
  forall (
    i in TASK, j in suc[i]
  )(
    s[i] + dur[i] + forwSU[i,j]*y[i,j] <= s[j]
  );
% Forward station load respects setup times
constraint
  forall (
    i in TASK, j in followForw[i]
  )(
    y[i,j] <-> ( s[i] + dur[i] + forwSU[i,j] == s[j] )
  );
% Backward station load respects station load
constraint
  forall (
    i in TASK
  )(
      s[i] + dur[i]
    + sum(
      j in followBack[i]
      )(
        backSU[i,j]*z[i,j]
      )
    <= load
  );
```

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% REDUNDANT CONSTRAINTS
% Cumulative Global
constraint
  cumulative(
    [ spair[i,j]       | i in TASK, j in TASK ],
    [ dur[i] + forwSU[i,j]  | i in TASK, j in TASK ],
    [ y[i,j]          | i in TASK, j in TASK ],
    1
  );
constraint
  forall(
    i in TASK, j in TASK
  )(
    s[i] == spair[i,j]
  );
% Fix some ordering variables to zero
constraint
  forall (
    i in TASK, j in TASK
  where
    not( j in followForw[i] )
  )(
    y[i,j] == 0
  );
constraint
  forall (
    i in TASK, j in TASK
  where
    not( j in followBack[i] )
  )(
    z[i,j] == 0
  );
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% OBJECTIVE
ann: my_search;
% Solve
solve :: my_search
minimize load;
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% OUTPUT
output
if full_output == 0 then
  ["load = " ++ show(load) ++ "\n"]
elseif full_output == 1 then
  ["load = " ++ show(load) ++ "\n"] ++
  ["start = " ++ show(s) ++ "\n"]
else
  [""]
endif;
```

Listing A.1: Full MiniZinc model for CP sub-problems

Table A.1: Breakdown of all 1076 adapted SBF2 instances

Class	Creator	#Inst.	n	m	$\|E\|$	OS
1		108	7-21	2-8	6-27	
	mertens	6	7	2-6	6	52.40
	bowman8	1	8	5	8	75.00
	jaeschke	5	9	3-8	11	83.33
	jackson	6	11	3-8	13	58.18
	mansoor	3	11	2-4	11	60.00
	mitchell	6	21	3-8	27	70.95
2		112	25-30	3-14	32-40	
	roszieg	6	25	4-10	32	71.67
	heskia	6	28	3-8	40	22.49
	buxey	7	29	7-13	36	50.74
	sawyer30	9	30	5-14	32	44.83
3		176	32-58	3-31	38-82	
	lutz1	6	32	6-11	38	83.47
	gunther	7	35	7-14	43	59.50
	kilbrid	10	45	3-10	62	44.60
	hahn	5	53	4-8	82	83.82
	warnecke	16	58	14-31	70	59.10
4		224	70-83	7-63	86-112	
	tonge	16	70	7-23	86	59.04
	wee-mag	24	75	31-63	87	22.70
	arc83	16	83	8-21	112	59.10
5		144	89-94	12-49	116-181	
	lutz2	11	89	24-49	116	77.60
	lutz3	12	89	12-23	116	77.60
	mukherje	13	94	13-25	181	44.80
6		208	111-148	7-51	175-176	
	arc111	17	111	9-27	176	40.40
	barthold	8	148	7-14	175	25.80
	barthol2	27	148	25-51	175	25.80
7		104	297-297	25-50	423-423	
	scholl	26	297	25-50	423	58.20
Overall		1076	7-297	2-31	6-82	

Table A.2: Full results of FSBF-2 with (3.46) on classes 1,2 and 3

Class	Alpha	Creator	#Nodes	%Gap	#No solution	#Optimal	%Optimal	Runtime(s)
1	1.00	mertens	117	0.00	0	6/6	100.00	0.46
		bowman8	0	0.00	0	1/1	100.00	0.40
		jaeschke	249	0.00	0	5/5	100.00	0.60
		jackson	3,798	0.00	0	6/6	100.00	6.31
		mansoor	5,237	0.00	0	3/3	100.00	3.95
		mitchell	234,143	0.79	0	5/6	83.33	682.81
	0.75	mertens	80	0.00	0	6/6	100.00	0.30
		bowman8	0	0.00	0	1/1	100.00	0.25
		jaeschke	106	0.00	0	5/5	100.00	0.40
		jackson	4,456	0.00	0	6/6	100.00	8.91
		mansoor	4,010	0.00	0	3/3	100.00	2.52
		mitchell	188,922	1.02	0	4/6	66.67	799.00
	0.50	mertens	80	0.00	0	6/6	100.00	0.32
		bowman8	0	0.00	0	1/1	100.00	0.10
		jaeschke	143	0.00	0	5/5	100.00	0.47
		jackson	4,456	0.00	0	6/6	100.00	8.81
		mansoor	3,902	0.00	0	3/3	100.00	2.70
		mitchell	166,621	1.02	0	4/6	66.67	737.33
	0.25	mertens	80	0.00	0	6/6	100.00	0.37
		bowman8	0	0.00	0	1/1	100.00	0.06
		jaeschke	107	0.00	0	5/5	100.00	0.42
		jackson	4,456	0.00	0	6/6	100.00	8.76
		mansoor	1,394	0.00	0	3/3	100.00	1.21
		mitchell	166,876	1.02	0	4/6	66.67	738.00
Overall			43,437	0.16	0	101/108	93.52	166.57
2	1.00	roszieg	269,369	12.92	0	0/6	0.00	1800.47
		heskia	401,337	12.40	0	0/6	0.00	1800.89
		buxey	55,003	28.79	0	0/7	0.00	1800.70
		sawyer30	30,589	28.26	0	0/9	0.00	1800.84
	0.75	roszieg	211,653	8.15	0	0/6	0.00	1800.44
		heskia	375,894	8.60	0	0/6	0.00	1800.88
		buxey	26,939	25.80	0	0/7	0.00	1800.70
		sawyer30	23,766	25.86	0	0/9	0.00	1800.86
	0.50	roszieg	364,062	5.29	0	0/6	0.00	1800.41
		heskia	354,638	6.84	0	0/6	0.00	1800.68
		buxey	38,177	16.03	0	0/7	0.00	1800.57
		sawyer30	34,790	16.35	0	0/9	0.00	1800.67
	0.25	roszieg	364,552	5.29	0	0/6	0.00	1800.41
		heskia	330,465	3.81	0	0/6	0.00	1800.71
		buxey	85,544	8.13	0	0/7	0.00	1800.64
		sawyer30	71,434	7.64	0	0/9	0.00	1800.59
Overall			168,899	13.76	0	0/112	0.00	1800.66
3	1.00	lutz1	52,738	10.98	0	1/6	16.67	1733.60
		gunther	37,277	31.30	0	0/7	0.00	1800.91
		kilbrid	45,651	24.87	1	0/10	0.00	1801.75
		hahn	74,365	14.70	0	0/5	0.00	1801.53
		warnecke	4,813	58.44	0	0/16	0.00	1802.23
	0.75	lutz1	49,364	10.60	0	0/6	0.00	1800.53
		gunther	39,351	22.45	0	0/7	0.00	1800.90
		kilbrid	85,124	18.87	2	0/10	0.00	1801.61
		hahn	112,021	7.18	0	0/5	0.00	1801.44
		warnecke	4,580	48.45	1	0/16	0.00	1802.40
	0.50	lutz1	115,017	2.81	0	2/6	33.33	1706.07
		gunther	26,716	18.32	0	0/7	0.00	1800.84
		kilbrid	49,411	17.51	1	0/10	0.00	1801.55
		hahn	108,078	7.62	0	0/5	0.00	1801.18
		warnecke	7,121	41.94	1	0/16	0.00	1802.23
	0.25	lutz1	63,440	1.76	0	4/6	66.67	733.06
		gunther	31,915	11.11	0	0/7	0.00	1800.79
		kilbrid	100,912	5.77	3	0/10	0.00	1801.41
		hahn	161,258	4.92	0	0/5	0.00	1801.18
		warnecke	7,657	31.58	0	0/16	0.00	1802.41
Overall			46,060	19.56	9	7/176	3.98	1759.67

Table A.3: Full results of SCBF-2 with (3.46) on classes 1,2 and 3

Class	Alpha	Creator	#Nodes	%Gap	#No solution	#Optimal	%Optimal	Runtime(s)
1	1.00	mertens	623	0.00	0	6/6	100.00	0.31
		bowman8	568	0.00	0	1/1	100.00	0.36
		jaeschke	739	0.00	0	5/5	100.00	0.43
		jackson	11,149	0.00	0	6/6	100.00	9.00
		mansoor	10,954	0.00	0	3/3	100.00	5.38
		mitchell	1,123,442	11.10	0	0/6	0.00	1800.25
	0.75	mertens	500	0.00	0	6/6	100.00	0.30
		bowman8	142	0.00	0	1/1	100.00	0.21
		jaeschke	596	0.00	0	5/5	100.00	0.28
		jackson	18,033	0.00	0	6/6	100.00	11.11
		mansoor	9,303	0.00	0	3/3	100.00	5.04
		mitchell	957,732	11.81	0	0/6	0.00	1800.24
	0.50	mertens	500	0.00	0	6/6	100.00	0.27
		bowman8	34	0.00	0	1/1	100.00	0.10
		jaeschke	665	0.00	0	5/5	100.00	0.26
		jackson	18,033	0.00	0	6/6	100.00	10.85
		mansoor	11,228	0.00	0	3/3	100.00	6.38
		mitchell	1,116,662	12.23	0	0/6	0.00	1800.23
	0.25	mertens	500	0.00	0	6/6	100.00	0.29
		bowman8	49	0.00	0	1/1	100.00	0.10
		jaeschke	545	0.00	0	5/5	100.00	0.28
		jackson	18,033	0.00	0	6/6	100.00	10.73
		mansoor	12,718	0.00	0	3/3	100.00	7.20
		mitchell	1,117,030	12.23	0	0/6	0.00	1800.24
Overall			244,811	1.97	0	84/108	77.78	403.17
2	1.00	roszieg	991,067	20.31	0	0/6	0.00	1800.30
		heskia	726,379	11.69	2	0/6	0.00	1800.60
		buxey	685,017	16.08	4	0/7	0.00	1800.50
		sawyer30	530,560	27.54	4	0/9	0.00	1800.57
	0.75	roszieg	734,178	18.44	0	0/6	0.00	1800.24
		heskia	756,204	17.72	2	0/6	0.00	1800.56
		buxey	558,135	24.44	3	0/7	0.00	1800.59
		sawyer30	497,528	29.45	4	0/9	0.00	1800.49
	0.50	roszieg	1,195,487	16.73	0	0/6	0.00	1800.31
		heskia	711,764	13.75	2	0/6	0.00	1800.52
		buxey	609,114	22.60	2	0/7	0.00	1800.50
		sawyer30	556,436	19.70	4	0/9	0.00	1800.53
	0.25	roszieg	1,193,838	16.73	0	0/6	0.00	1800.28
		heskia	652,072	11.64	2	0/6	0.00	1800.49
		buxey	580,369	23.58	0	0/7	0.00	1800.48
		sawyer30	614,005	25.58	2	0/9	0.00	1800.47
Overall			701,617	19.75	31	0/112	0.00	1800.47
3	1.00	lutz1	716,824	18.94	0	0/6	0.00	1800.37
		gunther	482,696	10.68	5	0/7	0.00	1800.65
		kilbrid	318,856	–	10	0/10	0.00	1801.17
		hahn	450,264	21.67	0	0/5	0.00	1800.85
		warnecke	170,586	–	16	0/16	0.00	1801.47
	0.75	lutz1	644,760	16.40	0	0/6	0.00	1800.38
		gunther	525,585	21.13	4	0/7	0.00	1800.62
		kilbrid	320,037	–	10	0/10	0.00	1801.08
		hahn	414,795	27.77	0	0/5	0.00	1800.86
		warnecke	167,942	–	16	0/16	0.00	1801.36
	0.50	lutz1	473,510	15.23	0	0/6	0.00	1800.42
		gunther	463,846	28.02	2	0/7	0.00	1800.60
		kilbrid	323,799	–	10	0/10	0.00	1800.96
		hahn	363,883	18.67	0	0/5	0.00	1800.92
		warnecke	135,786	–	16	0/16	0.00	1801.48
	0.25	lutz1	475,795	16.00	0	0/6	0.00	1800.33
		gunther	427,559	23.37	2	0/7	0.00	1800.50
		kilbrid	356,573	–	10	0/10	0.00	1800.97
		hahn	310,303	26.20	0	0/5	0.00	1800.70
		warnecke	111,564	–	16	0/16	0.00	1801.41
Overall			326,284	12.20	117	0/176	0.00	1801.00

References

[1] Abderrahmane Aggoun and Nicolas Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57 – 73, 1993. ISSN 0895-7177. doi: http://dx.doi.org/10.1016/0895-7177(93)90068-A. URL http://www.sciencedirect.com/science/article/pii/089571779390068A.

[2] Ali Allahverdi. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2): 345 – 378, 2015. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2015.04.004. URL http://www.sciencedirect.com/science/article/pii/S0377221715002763.

[3] Ali Allahverdi and H.M. Soroush. The significance of reducing setup times/setup costs. *European Journal of Operational Research*, 187(3): 978 – 984, 2008. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2006.09.010. URL http://www.sciencedirect.com/science/article/pii/S0377221706008162.

[4] Ali Allahverdi, C.T. Ng, T.C.E. Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985 – 1032, 2008. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2006.06.060. URL http://www.sciencedirect.com/science/article/pii/S0377221706008174.

[5] Carlos Fernández Andrés, Cristóbal Miralles, and Rafael Pastor. Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. *European Journal of Operations Research*, 187(3):1212–1223, 2008.

[6] David Applegate, Robert E. Bixby, Vaek Chvtal, , and William J. Cook. Concorde tsp solver, 2017. URL http://www.math.uwaterloo.ca/tsp/concorde.html.

[7] Philippe Baptiste and Claude Le Pape. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints*, 5(1):119–139, 2000. ISSN 1572-9354. doi: 10.1023/A:1009822502231. URL http://dx.doi.org/10.1023/A:1009822502231.

[8] I Baybars. A survey of exact algorithms for the simple assembly line balancing problem. *Manage. Sci.*, 32(8):900–932, August 1986. ISSN 0025-1909. URL http://dl.acm.org/citation.cfm?id=9579.9580.

[9] Christian Becker and Armin Scholl. A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168(3):694 – 715, 2006. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2004.07.023. URL http://www.sciencedirect.com/science/article/pii/S0377221704004801. Balancing Assembly and Transfer lines.

[10] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numer. Math.*, 4(1):238–252, December 1962. ISSN 0029-599X. doi: 10.1007/BF01386316. URL http://dx.doi.org/10.1007/BF01386316.

[11] Thierry Benoist, Etienne Gaudin, and Benoît Rottembourg. Constraint programming contribution to benders decomposition: A case study. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, CP '02, pages 603–617, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44120-4. URL http://dl.acm.org/citation.cfm?id=647489.727169.

[12] Timo Berthold, Stefan Heinz, Marco E. Lübbecke, Rolf H. Möhring, and Jens Schulz. A constraint integer programming approach for resource-constrained project scheduling. In *Proceedings of the 7th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, CPAIOR'10, pages 313–317, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-13519-6, 978-3-642-13519-4. doi: 10.1007/978-3-642-13520-0_34. URL http://dx.doi.org/10.1007/978-3-642-13520-0_34.

[13] Robert E. Bixby. Solving real-world linear programs: A decade and more of progress. *Oper. Res.*, 50(1):3–15, January 2002. ISSN 0030-364X. doi: 10.1287/opre.50.1.3.17780. URL http://dx.doi.org/10.1287/opre.50.1.3.17780.

[14] Alexander Bockmayr and Thomas Kasper. Branch and infer: A unifying framework for integer and finite domain constraint programming. *INFORMS J. on Computing*, 10(3):287–300, March 1998. ISSN 1526-5528. doi: 10.1287/ijoc.10.3.287. URL http://dx.doi.org/10.1287/ijoc.10.3.287.

[15] Nils Boysen, Malte Fliedner, and Armin Scholl. A classification of assembly line balancing problems. *European Journal of Operational Research*, 183 (2):674 – 693, 2007. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2006.10.010. URL http://www.sciencedirect.com/science/article/pii/S0377221706010435.

[16] Jacques Carlier and Eric Pinson. Jackson's pseudo-preemptive schedule and cumulative scheduling problems. *Discrete Applied Mathematics*, 145

(1):80 – 94, 2004. ISSN 0166-218X. doi: https://doi.org/10.1016/j.dam.2003.09.009. URL http://www.sciencedirect.com/science/article/pii/S0166218X04000678. Graph Optimization {IV}.

[17] Yves Caseau and Franois Laburthe. Cumulative scheduling with task intervals, 1994.

[18] Geoffrey G. Chu. *Improving Combinatorial Optimization*. PhD thesis, The University of Melbourne, 2011. URL http://hdl.handle.net/11343/36679.

[19] Jean-Franois Cordeau, Goran Stojkovi, Franois Soumis, and Jacques Desrosiers. Benders decomposition for simultaneous aircraft routing and crew scheduling. *Transportation Science*, 35(4):375–388, 2001. ISSN 00411655, 15265447. URL http://www.jstor.org/stable/25768970.

[20] Alysson M. Costa. A survey on benders decomposition applied to fixed-charge network design problems. *Comput. Oper. Res.*, 32(6):1429–1450, June 2005. ISSN 0305-0548. doi: 10.1016/j.cor.2003.11.012. URL http://dx.doi.org/10.1016/j.cor.2003.11.012.

[21] Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. *Constraint Programming Based Column Generation for Employee Timetabling*, pages 140–154. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-32264-1. doi: 10.1007/11493853_12. URL http://dx.doi.org/10.1007/11493853_12.

[22] M. A. H. Dempster and R. T. Thompson. Parallelization and aggregation ofnested benders decomposition. *Annals of Operations Research*, 81(0):163–188, 1998. ISSN 1572-9338. doi: 10.1023/A:1018996821817. URL http://dx.doi.org/10.1023/A:1018996821817.

[23] Thorsten Ehlers and Peter J. Stuckey. *Parallelizing Constraint Programming with Learning*, pages 142–158. Springer International Publishing, Cham, 2016. ISBN 978-3-319-33954-2. doi: 10.1007/978-3-319-33954-2_11. URL http://dx.doi.org/10.1007/978-3-319-33954-2_11.

[24] Rasul Esmaeilbeigi, Bahman Naderi, and Parisa Charkhgard. New formulations for the setup assembly line balancing and scheduling problem. *OR Spectrum*, 38(2):493–518, 2016.

[25] Thibaut Feydy, Adrian Goldwaser, Andreas Schutt, Peter J. Stuckey, Kenneth Young, and Thibaut Feydy. Priority search with minizinc, 2017.

[26] Henry Ford and Samuel Crowther. *My Life and Work by Henry Ford*. Garden City, N.Y., Doubleday, Page Co, 1922. URL http://www.gutenberg.org/ebooks/7213.

[27] A. M. Geoffrion. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, 10(4):237–260, 1972. ISSN 1573-2878. doi: 10.1007/BF00934810. URL http://dx.doi.org/10.1007/BF00934810.

[28] J. N. Hooker. A hybrid method for the planning and scheduling. *Constraints*, 10(4):385–401, 2005. ISSN 1572-9354. doi: 10.1007/s10601-005-2812-2. URL http://dx.doi.org/10.1007/s10601-005-2812-2.

[29] J. N. Hooker. Planning and scheduling by logic-based benders decomposition. *Oper. Res.*, 55(3):588–602, May 2007. ISSN 0030-364X. doi: 10.1287/opre.1060.0371. URL http://dx.doi.org/10.1287/opre.1060.0371.

[30] J.N. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003. ISSN 1436-4646. doi: 10.1007/s10107-003-0375-9. URL http://dx.doi.org/10.1007/s10107-003-0375-9.

[31] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '87, pages 111–119, New York, NY, USA, 1987. ACM. ISBN 0-89791-215-2. doi: 10.1145/41625.41635. URL http://doi.acm.org/10.1145/41625.41635.

[32] Vipul Jain and Ignacio E. Grossmann. Algorithms for hybrid milp/cp models for a class of optimization problems. *INFORMS J. on Computing*, 13(4):258–276, September 2001. ISSN 1526-5528. doi: 10.1287/ijoc.13.4.258.9733. URL http://dx.doi.org/10.1287/ijoc.13.4.258.9733.

[33] A.H. Land and A.G. Doig. *An Automatic Method of Solving Discrete Programming Problems*. Reprint series / Research Techniques Unit of the London School of Economics and Political Science). London School of Economics and Political Science, 1960. URL https://books.google.com.au/books?id=V61oMwEACAAJ.

[34] H. Li. *Benders Decomposition Approach for Project Scheduling with Multi-Purpose Resources*, pages 587–601. Springer International Publishing, Switzerland, Cham, 2015. ISBN 978-3-319-05443-8. doi: 10.1007/978-3-319-05443-8_27. URL http://dx.doi.org/10.1007/978-3-319-05443-8_27.

[35] H. Li and K. Womer. Scheduling projects with multi-skilled personnel by a hybrid milp/cp benders decomposition algorithm. *Journal of Scheduling*, 12 (3):281, 2009. ISSN 1099-1425. doi: 10.1007/s10951-008-0079-3. URL http://dx.doi.org/10.1007/s10951-008-0079-3.

[36] Haitao Li and Mehdi Amini. A hybrid optimization approach to configure a supply chain for new product diffusion: a case study of multiple-sourcing strategy. *International Journal of Production Research*, 50(11):3152–3171, 2012.

[37] Haitao Li and Keith Womer. A decomposition approach for shipboard manpower scheduling. *Military Operations Research*, 14(3):1–23, 2009.

[38] Andrea Lodi. *Mixed Integer Programming Computation*, pages 619–645. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-540-68279-0. doi: 10.1007/978-3-540-68279-0_16. URL http://dx.doi.org/10.1007/978-3-540-68279-0_16.

[39] T. L. Magnanti and R. T. Wong. Accelerating benders decomposition: Algorithmic enhancement and model selection criteria. *Operations Research*, 29 (3):464–484, 1981. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/170108.

[40] Stephen J. Maher, Guy Desaulniers, and François Soumis. Recoverable robust single day aircraft maintenance routing problem. *Comput. Oper. Res.*, 51:130–145, November 2014. ISSN 0305-0548. doi: 10.1016/j.cor.2014.03.007. URL http://dx.doi.org/10.1016/j.cor.2014.03.007.

[41] Luigi Martino and Rafael Pastor. Heuristic procedures for solving the general assembly line balancing problem with setups. *International Journal of Production Research*, 48(6):1787–1804, 2010. doi: 10.1080/00207540802577979. URL http://dx.doi.org/10.1080/00207540802577979.

[42] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack. Minizinc: Towards a standard cp modelling language. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007: 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007. Proceedings*, pages 529–543, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-74970-7. doi: 10.1007/978-3-540-74970-7_38.

[43] Soren S. Nielsen and Stavros A. Zenios. Scalable parallel benders decomposition for stochastic linear programming. *Parallel Computing*, 23 (8):1069 – 1088, 1997. ISSN 0167-8191. doi: http://dx.doi.org/10.1016/S0167-8191(97)00044-6. URL http://www.sciencedirect.com/science/article/pii/S0167819197000446.

[44] W.P.M. Nuijten. *Time and resource constrained scheduling : a constraint satisfaction approach*. PhD thesis, Technische Universiteit Eindhoven, 1994. URL http://dx.doi.org/10.6100/IR431902.

[45] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, 2009. ISSN 1572-9354. doi: 10.1007/s10601-008-9064-x. URL http://dx.doi.org/10.1007/s10601-008-9064-x.

[46] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991. doi: 10.1137/1033004. URL http://dx.doi.org/10.1137/1033004.

[47] David Pisinger and Mikkel Sigurd. Using decomposition techniques and constraint programming for solving the two-dimensional bin-packing problem. *INFORMS J. on Computing*, 19(1):36–51, January 2007. ISSN 1526-5528. doi: 10.1287/ijoc.1060.0181. URL http://dx.doi.org/10.1287/ijoc.1060.0181.

[48] R. Rahmaniani, T.G. Crainic, M. Gendreau, and W. Rei. The benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017. doi: 10.1016/j.ejor.2016.12.005. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85009384722&doi=10.1016%2fj.ejor.2016.12.005&partnerID=40&md5=68f888c6abfc502d35488967968de98b. cited By 1.

[49] J. Rodriguez. A constraint programming model for real-time train scheduling at junctions, 2007.

[50] Georgios K. D. Saharidis and Marianthi G. Ierapetritou. Improving benders decomposition using maximum feasible subsystem (mfs) cut generation strategy. *Computers and Chemical Engineering*, 34(8):1237–1245, 2010. URL http://dblp.uni-trier.de/db/journals/cce/cce34.html#SaharidisI10.

[51] Tjendera Santoso, Shabbir Ahmed, Marc Goetschalckx, and Alexander Shapiro. A stochastic programming approach for supply chain network design under uncertainty. *European Journal of Operational Research*, 167(1):96–115, November 2005. URL https://ideas.repec.org/a/eee/ejores/v167y2005i1p96-115.html.

[52] Martin Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841, 1997. URL http://EconPapers.repec.org/RePEc:inm:oropre:v:45:y:1997:i:6:p:831-841.

[53] Armin Scholl and Robert Klein. Balancing assembly lines effectively a computational comparison. *European Journal of Operational Research*, 114(1):50 – 58, 1999. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(98)00173-8. URL http://www.sciencedirect.com/science/article/pii/S0377221798001738.

[54] Armin Scholl, Nils Boysen, and Malte Fliedner. The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics. *OR Spectrum*, 35(1):291–320, 2013.

[55] Tom Schrijvers, Guido Tack, Pieter Wuille, Horst Samulowitz, and Peter J. Stuckey. *An Introduction to Search Combinators*, pages 2–16. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38197-3. doi: 10.1007/978-3-642-38197-3_2. URL http://dx.doi.org/10.1007/978-3-642-38197-3_2.

[56] Tom Schrijvers, Guido Tack, Pieter Wuille, Horst Samulowitz, and Peter J. Stuckey. Search combinators. *Constraints*, 18(2):269–305, 2013. ISSN 1572-9354. doi: 10.1007/s10601-012-9137-8. URL http://dx.doi.org/10.1007/s10601-012-9137-8.

[57] A. Schutt, T. Feydy, P. J. Stuckey, and M. G. Wallace. Explaining the cumulative propagator. *Constraints*, 16(3):250–282, 2011. ISSN 1572-9354. doi: 10.1007/s10601-010-9103-2. URL http://dx.doi.org/10.1007/s10601-010-9103-2.

[58] A. Schutt, P. J. Stuckey, and A. R. Verden. Optimal carpet cutting. In Jimmy Lee, editor, *Proceedings of Principles and Practice of Constraint Programming – CP 2011*, pages 69–84, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-23786-7. doi: 10.1007/978-3-642-23786-7_8. URL http://dx.doi.org/10.1007/978-3-642-23786-7_8.

[59] A. Schutt, T. Feydy, and P. J. Stuckey. Scheduling optional tasks with explanation. In Christian Schulte, editor, *Proceedings of Principles and Practice of Constraint Programming – CP 2013*, pages 628–644, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40627-0. doi: 10.1007/978-3-642-40627-0_47. URL http://dx.doi.org/10.1007/978-3-642-40627-0_47.

[60] Andreas Schutt, Thibaut Feydy, and Peter J. Stuckey. Explaining time-table-edge-finding propagation for the cumulative resource constraint. In Carla P. Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 7874 of *Lecture Notes in Computer Science*, pages 234–250. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38170-6. doi: 10.1007/978-3-642-38171-3_16.

[61] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. A satisfiability solving approach. In Christoph Schwindt and Jürgen Zimmermann, editors, *Handbook on Project Management and Scheduling Vol. 1*, pages 135–160. Springer International Publishing, Cham, 2015. ISBN 978-3-319-05443-8. doi: 10.1007/978-3-319-05443-8_7. URL http://dx.doi.org/10.1007/978-3-319-05443-8_7.

[62] R. Szeredi and A. Schutt. Modelling and solving multi-mode resource-constrained project scheduling. In Michel Rueher, editor, *Proceedings of Principles and Practice of Constraint Programming – CP 2016*, pages 483–492, Switzerland, Cham, 2016. Springer International Publishing. ISBN 978-3-319-44953-1. doi: 10.1007/978-3-319-44953-1_31. URL http://dx.doi.org/10.1007/978-3-319-44953-1_31.

[63] Christian Timpe. Solving planning and scheduling problems with combined integer and constraint programming. *OR Spectrum*, 24(4):431–448, 2002. ISSN

1436-6304. doi: 10.1007/s00291-002-0107-1. URL http://dx.doi.org/10.1007/s00291-002-0107-1.

[64] Tony T. Tran and J. Christopher Beck. Logic-based benders decomposition for alternative resource scheduling with sequence dependent setups. In *Proceedings of the 20th European Conference on Artificial Intelligence*, ECAI'12, pages 774–779, Amsterdam, The Netherlands, The Netherlands, 2012. IOS Press. ISBN 978-1-61499-097-0. doi: 10.3233/978-1-61499-098-7-774. URL https://doi.org/10.3233/978-1-61499-098-7-774.

[65] Pascal Van Hentenryck, Laurent Perron, and Jean-François Puget. Search and strategies in opl. *ACM Trans. Comput. Logic*, 1(2):285–320, October 2000. ISSN 1529-3785. doi: 10.1145/359496.359529. URL http://doi.acm.org/10.1145/359496.359529.

[66] Abdolmajid Yolmeh and Farhad Kianfar. An efficient hybrid genetic algorithm to solve assembly line balancing problem with sequence-dependent setup times. *Comput. Ind. Eng.*, 62(4):936–945, May 2012. ISSN 0360-8352. doi: 10.1016/j.cie.2011.12.017. URL http://dx.doi.org/10.1016/j.cie.2011.12.017.