Solving the Flexible Job Shop Problem with Alternative Process Plans

Evaluating Constraint Programming and Multivalued Decision Diagrams

Master Thesis

Steven Boonstoppel
Computing Science

Supervisors UU:

Dr. Han Hoogeveen
Research Institute of Information and Computing Sciences

Dr. ir. Marjan van den Akker
Research Institute of Information and Computing Sciences

Supervisor TNO:

Dr. Jacques Verriet
TNO-ESI

August 2025

Abstract

This thesis addresses the Flexible Job Shop Scheduling Problem (FJSP) with three extensions: Sequence-Dependent Setup Times (SDST), Blocking tasks and Alternative Process Plans (APPs). The research evaluates the efficacy of two distinct optimization paradigms: Constraint Programming (CP) and Multivalued Decision Diagrams (MDDs). For CP, both the commercial IBM CPLEX CP Optimizer and Google's open-source OR-Tools CP-SAT solver are utilized. The study formalizes the problem, including multifunctional machine routing, SDST, blocking constraints, and APPs, into a unified CP model, demonstrating its robust framework for real-time, make-to-order environments. Both solvers are competitive, with either having a slight advantage in certain aspects. Furthermore, it explores MDDs as a complementary technique, showcasing how restricted and relaxed MDD variants can rapidly generate bounds and decent solutions. Through extensive computational experiments on both established and newly generated benchmark instances (the latter made publicly available), this thesis shows that alternative process plans can directly be implemented to try and minimize setup times and establishes hybrid CP-MDD strategies as a promising direction for large-scale, real-time scheduling implementations in high-mix, low-volume production settings. The findings indicate that while CP offers greater flexibility and gradually improves solutions over longer runtimes, MDDs excel in quickly generating decent schedules, particularly when SDST are involved, making them suitable for scenarios prioritizing rapid schedule creation.

Preface

Starting in September 2024, I knew that writing a thesis was going to be a challenge, not in the least due to the part-time teaching job. I set myself the goal to find out whether I would see myself as a researcher at an institute or company, and therefore wanted to write my thesis outside of the university. I applied at a vacancy at the research institute TNO, where I was welcomed onto the High Tech Campus of Eindhoven. With Eindhoven being quite a distant travel, we agreed that I would be on-site one day a week, work two days from home, with the other two days as a teacher at school. This partition of the week was not always the easiest, especially in the beginning, as I was not always sure what to do or was easily sidetracked by emails or other projects. The scheduling field was new to me, and I was given much space to choose freely.

However, as my subject took shape, it became more clear what I wanted to do and what I was good at. I did not want to create another extremely long abbreviation with a very niche setting in the Job Shop field, but rather contribute something that would hopefully be useful and lead on to new projects. This turned out to be the exploration of Multivalued Decision Diagrams for (Flexible) Job Shop scheduling, in comparison with Constraint Programming. With this subject, I could learn and include both theoretical aspects (coming up with a formulation for several modifications and/or extensions) as well as obtaining numeric results comparing these two paradigms. At first, it looked like MDDs would lose hopelessly against CP, but in the end, the obtained results do show that MDDs are quite promising. This of course is a sweet observation, and, combined with diminishing work load at school due to the approaching holiday, gave me more confidence and helped boosting my concentration to completing my thesis. I am happy with the final result.

I would like to thank Jacques Verriet first and foremost, my daily (weekly) supervisor, who was always happy to discuss and review topics, took me on a few trips to talk to some experienced people in the field, and helped steering me into the right directions where needed. Next, I want to thank Han Hoogeveen, who made sure that my research and paper is actually theoretically sound and that I would not lose the big picture. I am sure that he raised his eyebrows a few times on the initial progress, but that helped me to push extra during the final couple of months. My gratitude also goes out to the people that I had some small chats with or helped me in another way over the course of my thesis, among which Leon regarding constraint programming, Eghonghon-Aye regarding multivalued decision diagrams and Marvin about all sorts of orders, machines and process plans, keeping the theory connected to the application. Also, I am grateful for the access to the server hardware at my job, allowing me to generate all the results for this thesis. Last but not least, I thank my parents and girlfriend for supporting me during all phases of the process, even when they did not really have a clue what I was working on.

Table of Contents

1 Introduction

1.1. Motivation

Manufacturing systems are ubiquitous and sustain almost all our production lines. Many of these systems are simple: there is a fixed operation to be performed by a single machine or human such as packing an item in a box, with the goal of processing as many as possible. However, industrial systems quickly grow quite complex with varying requirements, machines and customer orders.

Examples of such complex manufacturing industries include PCB manufacturing (where circuit boards must be drilled, copper-plated, silkscreened, coated, cut to the right shape and size, using materials of varying thickness), semiconductor manufacturing (where wafers are cleaned, coated, implanted with ions, etched, multiple materials are deposited, while maintaining certain temperature conditions) or the printing industry (where books, cards, leaflets and magazines are printed on one or both paper sides, on different materials, folded or cut to different shapes and sizes, with all sorts of finishings such as embossing, stapling, laminating, etc.). These industries have in common that products are not simply made by a single machine or using a fixed routine, but instead evolved into highly flexible environments capable of producing many different products in many different ways.

With the rise of (online) tools that provide accessibility to these industries even to less-skilled consumers, it has become very easy for individuals or small companies to order e.g. a small quantity of PCBs or a few leaflets. Moreover, competition and innovation in the industry has caused a decrease in price. As a result, with orders of all shapes and sizes, the market sees more High-Mix Low-Volume (HMLV) production rather than Low-Mix High-Volume (LMHV) and transforms into an environment where Make-to-Order is the norm, rather than Make-to-Stock (Gan et al. 2023). In turn, it becomes more and more complex for manufacturing business to generate efficient schedules that maximize their order production.

In this Master's thesis, we hope to improve these schedules by looking at one of the latest developments in this area, namely Alternative Process Plans: different sets of activities that yield the same product. As the setting is similar and results are highly applicable to the Online Printing Shop (OPS) scheduling problem as proposed by Lunardi et al. (2020), we will stick to the domain of the printing industry as a working example throughout this thesis. However, all discussed concepts apply similarly to other industries. Moreover, we will attempt to use a relatively new modelling technique to solve these kinds of scheduling problems, namely Multivalued Decision Diagrams, to try and see whether it is a useful technique.

1.2. Orders: the customer viewpoint

Traditionally, printing shops are used for large-quantity orders, such as the daily newspaper. In this case, printing is done 'analog': an offset is created for each page, and thousands of identical pages can be printed with ease. However, with the rise of HMLV, there has been a shift to digital printing, similar to what we are used to with personal printers. While e.g. some popular books may still be printed with the thousands, an upcoming writer may order print runs of a handful of books at a time. A small business orders a few magazines to be put on display, and a small event orders a few posters and leaflets.

In all of these examples, the customer orders a product with a certain printing design and possibly specifies the material (thickness, size, finish) and the deadline by which they need it produced. As long as those requirements are met, the shop operator is free to choose how that product is made, and at what time it will be produced.

1.3. The production floor: the shop viewpoint

We will set the scene for the production floor from the perspective of the printing industry. A printing business' production floor typically consists of a number of printers and finishers. Generally, a machine has a number of functions. For instance, while some can only print simplex (one-sided), others are able to print simplex and duplex (two-sided); some can print up to paper size B4, while others can handle larger sizes etc. If a shop has all identical machines, they are typically named *parallel* units (Heinz et al. 2022). Otherwise, if they all have their own set of functionalities, they are commonly named *multi-function* units (Hurink et al. 1994).

Obviously, not all machines have equal speed, even if they have the same functionalities. Newer models can be faster, or a specialized model may be very good at a particular task. Consequently, the machines have *flexible processing speeds.* The processing speed may also differ per functionality of the machine: it usually takes longer to print or emboss a larger sheet of paper. This flexible processing speed only applies to the multi-function setting: parallel machines (see previous paragraph) are considered to have identical speed.

This however is not the only thing that should be noted about multi-functional machines. If one machine can, for example, process both A3- and A4-sized paper, parts of the machine must be widened or narrowed when switching the size of paper: these lead to *sequence-dependent setup times.* These setup times depend on the sequence of the schedule: if a subsequent task on the same machine has the same properties, no or smaller setup time is required; if it has different properties, the setup time will be larger. The effect of these setup times becomes significantly more impactful in HMLV compared to LMHV. A business that only prints newspapers may find the setup times insignificant in their schedule, but the effect is much more apparent in a shop that prints all sorts of products. It could be quite inefficient to process orders in a random mix: if there are multiple orders of the same type requiring the same function, setup times can potentially be reduced by scheduling them back to back.

When looking at the production chain in a factory as a whole, another point that arises is the use of buffers: while production is in progress, parts may need to be stored before moving on to the next stage. This happens in two cases: during a production step (e.g. printing a thousand sheets), or when assembling subparts (e.g. book blocks and their covers). Three types of buffers are distinguished in the literature: infinite buffers, finite buffers or no buffers. Infinite buffers make it possible to process a single operation (such as printing one side) at any time, without concern about the storage size. Finite buffers mean that after a certain number of partial products are made, they have to move on to the next process as storage in practice is not infinite. Sometimes, depending on the shop or machines, one stack of output can be stored at the machine until the next machine is available. This results in a blocked machine: if another job were to enter the machine, the output would contain multiple jobs. This scenario is typically described as *blocking.* When there are no buffers, the output must be immediately transported to the next machine: this case is also known as *no-wait* or *tightly-coupled.* This is for instance the case when transport belts connect machines back-to-back.

1.4. Processing orders: the operator viewpoint

With a set of orders (mostly referred to as *jobs* throughout this work) placed during a day, and a set of available machines, the operator goes to make a schedule or requests a program to generate a schedule. The scheduler has some degrees of freedom to generate a schedule.

Firstly, there is the *sequencing* freedom: when do we start processing a certain job (and the specific tasks within that job, such as printing or cutting). Sometimes, this choice is affected e.g. by a reward for finishing a specific job earlier so it can be shipped faster, while in other occasions, it is just a matter of finishing the whole set of jobs as soon as possible.

Then, there is the *assignment* freedom: selecting which machine is going to produce (parts of) a job. For instance, if the business owns multiple printers, and assuming at most a marginal difference in quality between these machines, any of them may be assigned to a task without affecting the final product. If an order consists of multiple sub-parts or processing steps, this freedom is available for any such step.

Lastly, there is the *processing* freedom. While the final product that is shipped to the customer must meet the order requirements, the production process is not fixed. In turn, this means that the shop operator is free to choose (some aspects of) the input material and operations performed on the materials. Sticking to the printing theme, imagine an order for an A4 booklet: while it can be produced by printing on A4 sheets and binding these, it can also be made by printing on A3 sheets and folding them before binding, or printing on A3 sheets and cutting them in halves prior to the binding process. Any of these *production plans* is a valid choice, and thus this selection of process plans can be utilized by the shop operator to create an efficient production schedule.

1.5. The production schedule

Production businesses typically work with production schedules: a timetable that shows for each machine in the shop what it should produce and when this should happen. Usually, these are divided into *shifts* of e.g. 8 hours. Depending on the business and agreements, during or at the end of the shift the orders are shipped out. The most common question in this case is: how many orders can be produced during the shift? If there are a lot of orders, choices have to be made about which ones must be prioritized and which ones can be delayed (maybe at a certain penalty) - typically, this is referred to as minimizing the (weighted) tardiness. If the order deadlines are less pressing, the challenge typically reduces to minimizing the total time required to produce all the orders: the *makespan*. The makespan is the time it takes from the start of the production to the end of the latest completion time.

The schedule is always limited by a bottleneck: the machine(s) that limit(s) the reduction of the makespan, or the operator(s) required to switch machines (corresponding to the sequence-dependent setup times). This is where the alternative process plans might be able to help. Taking the production flexibility into account can reduce the load of the bottleneck, decreasing the overall makespan of the shop. Or, these process plans may reduce the setup times, in turn resulting in less time consumed by the operator.

1.6. Goal

The goal of this research is to be able to produce a schedule for a typical daily shift of a moderately sized shop subject to High-Mix Low-Volume: e.g. a set of up to 100 orders that must be produced before the delivery truck collects these for shipping at the end of the day. The resulting solution would preferably allow an operator to run the scheduler while starting up the factory or getting a cup of coffee, and have a (near-) optimal schedule presented within a few minutes. Considering this goal, the main question arising in this research is: what is the effect of including setup times and alternative process planning into the scheduler on the quality and makespan of the produced schedule?

1.7. Example instance

To demonstrate the impact of some of the points raised in the previous section, we consider an example of a printing shop with four machines. These machines have the following functions:

- M_1: it can print A4-sized paper.
- M_2: it can print A3- and A4-sized paper.
- M_3: it can perform any cut, including halving a sheet A3 into two sheets A4.
- M_4: it applies a finish to any sheet.

M_2 is a multifunctional machine. It is specialized in printing A3-sized paper at a high speed, but can still print A4 at a speed comparable to M_1.

Next, consider five orders. Their specifications are:

- Order 1: 8 A4 sheets, with finish. Note: print on M_1 (due to reproducibility with previous order).
- Order 2: 12 A4 sheets, no finish.
- Order 3: 8 A4 sheets, with finish.
- Order 4: 4 A3 sheets, cut to some unique size, with finish.
- Order 5: 4 A4 sheets, no finish. Note: print on special A4-only paper.

The finish could be per sheet (such as embossing each page) or per stack of sheets (such as binding), but that distinction is not important here.

A basic generation of process plans that starts with the materials matching the order specifications would result in the following set of process plans:

- Plan 1: $[(M_1, 4)] \rightarrow [(M_4, 2)]$
- Plan 2: $[(M_1, 6) \vee (M_2, 6)]$
- Plan 3: $[(M_1, 4) \vee (M_2, 4)] \rightarrow [(M_4, 3)]$
- Plan 4: $[(M_2, 4)] \rightarrow [(M_3, 1)] \rightarrow [(M_4, 1)]$
- Plan 5: $[(M_1, 2) \vee (M_2, 2)]$

Here, (M_i, p) means that machine M_i can be used with a total process time of p time units. $[a \vee b]$ indicates that either a or b can be selected to process this activity. $\alpha \rightarrow \beta$ indicates precedence constraints: activity α must be completed before β can be processed. Processing times are derived from the number of sheets of a task and the speed of machine (e.g. machine 1 can process four A4-sheets per unit of time in this example).

Now consider that orders 2 and 3 may just as well be produced on sheets of A3 paper and subsequently cut into two sheets A4: the resulting product is identical to the customer. Thus, their alternatives process plans would be:

- Plan 2 (alt): $[(M_2, 2)] \rightarrow [(M_3, 1)]$

- Plan 3 (alt): $[(M_2, 1)] \rightarrow [(M_3, 1)] \rightarrow [(M_4, 3)]$

In Figure 1.1, four schedules are presented for this example. The first three sub-figures consider the normal, singular process plans. In Figure 1.1(a), only a few basic rules are present: the precedence constraints must be followed, machines can only process one activity at any time, and an activity can only be processed by one machine at a time. This yields a makespan of 10 time units. In Figure 1.1(b), one of the rules is tightened: activities must start exactly when the preceding activity is completed (we cannot hold items in buffers). Consequently, some activities cannot start immediately, and as a result the makespan increases to 11 time units. In Figure 1.1(c), a new rule is added: sequence-dependent setup times. Machine M_2 requires 2 time units to switch between printing sheets of A3 and A4, and machine M_4 requires 1 time unit between any pair of activities for this example. The makespan increases to 13 time units because of the additional setup times incurred. However, in Figure 1.1(d), the alternative process plans are added into the model. All rules present in Figure 1.1(c) apply, but the increased flexibility means that both the second and third order can now be produced on M_2's A3 function without incurring any setup time on this machine, while orders 1 and 5 can be printed on M_1 without incurring setup time. This means that a shorter makespan of 10 time units can be achieved, and shows the advantage of considering these alternative process plans.

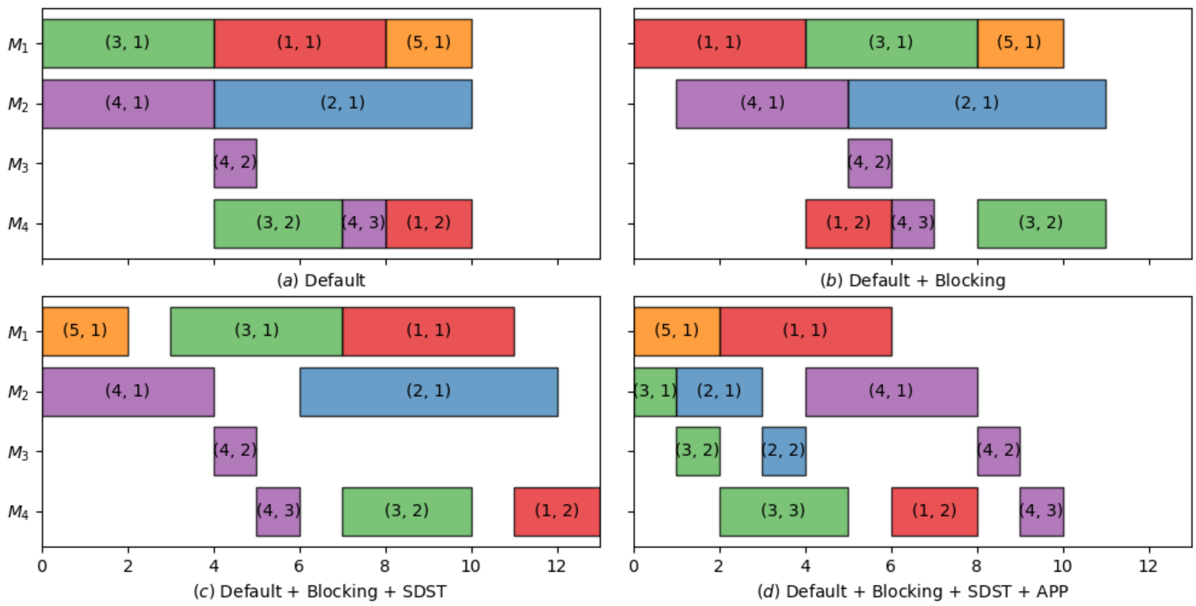

Figure 1.1.: Gantt schedules for example instance with four machines, five jobs.

1.8. Contribution

The primary contribution of this thesis is threefold. First, it formalizes a typical production scheduling problem — including multifunctional machine routing, sequence-dependent setup times, blocking constraints and alternative process plans — into a unified Constraint Programming (CP) model, providing a proven framework for real-time, make-to-order environments. Second, it explores Multivalued Decision Diagrams (MDDs) as a new and complementary paradigm, demonstrating how restricted and relaxed MDD variants can rapidly generate bounds and possibly warm-start solutions for the CP solver. Finally, through extensive computational experiments on benchmark instances, this thesis demonstrates the operational value of alternative process plans and establishes hybrid CP–MDD strategies as a promising direction for large-scale, real-time implementations. Together, these contributions advance both the theory and practice of scheduling in high-mix, low-volume manufacturing systems.

1.9. Outline

In Chapter 2, we discuss the history of (job shop) scheduling as well as the history and categorization of our setting. Then we introduce some notation and formalization in Chapter 3. Next, in Chapter 4 we introduce a Constraint Programming model, and in Chapter 5 an MDD model. In Chapter 6, we collect benchmark results for a variety of configurations to compare the CP and MDD models. Finally, our findings and comparisons are discussed in Chapter 7 with some recommendations for future research.

2 Literature review

2.1. History

The scheduling of jobs has a rich history, with a commonly recognized starting point dating back to a paper by Johnson (1954). Johnson considered a scenario of a shop where n items must be processed in two or three stages with one machine per stage, and was able to optimally solve the problem for two stages, as well as a restricted version of the three-stage problem. According to Xiong et al. (2022), this marked the start of the field of research, that received its name 'Job Shop Scheduling' from Sisson (1959), which is now a widely used term. However, this field of research is at times also called Machine Scheduling, as it emphasizes that this problem is primarily concerned with generating a machine schedule (even though this directly yields a job/task schedule).

A classical Job Shop Scheduling Problem (JSSP) consists of a set of *jobs* $J = \{J_1, J_2, ..., J_n\}$, where each job J_i consists of a set of *tasks* $O_i = \{O_{i1}, O_{i2}, ..., O_{in_i}\}$ defined in topological order, to be processed on a set of machines $M = \{M_1, M_2, ..., M_m\}$. Each task is assigned a machine in M on which it must be processed with a given processing time p_{ij}. In the JSSP, both machines and jobs are unitary: a machine can process one job at a time, and a job can only be processed by one machine at a time. And except for some special research cases, tasks in a job are typically not allowed to overlap. The task is to find a sequence of tasks on the machines that minimizes the latest completion time of all jobs, i.e. the duration in which all jobs are processed, defined as the makespan, denoted by $C_{\max}$.

While our research deals with unitary machines and tasks, we will lend ideas from a related field. In contrast to the unitary machine scheduling, a closely linked field to JSSP is the Resource-Constrained Project Scheduling Problem (RCPSP). In the RCPSP, the same notion of jobs (although usually one job or project), tasks and machines is used. However, neither machines nor tasks are unitary in the RCPSP: a task may (or must) be processed by multiple machines (simultaneously) and a machine may process multiple tasks at the same time.

2.2. Three-field notation

As many applications have different environments, requirements or goals, numerous studies have been carried out on Job Shop scheduling. They each have their own set of constraints, extensions and objectives. To categorize these problems, scheduling literature commonly uses the $\alpha|\beta|\gamma$ three-field notation proposed by Graham et al. (1979).

In this notation, α denotes the machine environment. Some popular examples are:

1: there is a single machine in this shop/problem.

P**,** Pm**:** the shop consists of a set of parallel, identical machines such that the processing time p_j is equal for all machines. If m is specified, this is a fixed size. Otherwise, m is part of the input.

Q**,** Qm**:** the shop consists of a set of parallel machines with different speeds s_i for machine M_i. The execution time of job J_j is p_j/s_i for machine M_i. m as above.

R, Rm: the shop consists of a set of parallel machines, but they are completely unrelated, such that a processing time must be specified for each job on each machine; m as above.

J: the Job shop problem, in which every job consists of a collection of tasks. Each task must be processed on a specified machine. The machines are unrelated and may be passed in different order.

F: the Flow shop problem, in which every job consists of an identical sequence of tasks. In contrast to the Job shop, all jobs pass through the specified machines in the same order, for instance coupled by a transport belt.

O: the Open shop problem, in which every job J_j corresponds to one task for each machine i in the shop. Task O_{ij} can be be scheduled in any order: there are no precedence constraints. Task O_{ij} must be processed for p_{ij} time units.

Secondly, β describes the job or machine characteristics that are present in the shop: possibly none, but usually one or multiple. A selection of commonly used constraints are these:

r_j : a release date is specified for each job (or task, depending on the problem).

d_j : a due date is specified for each job (or task).

$bkdwn$: machines may suffer (un)expected breakdowns.

s_{jk} : sequence-dependent setup times are present between successive tasks on a machine.

block: a task remains on its machine after processing until it is processed by the next machine, blocking the machine from processing another task.

pmtn: pre-emption of tasks is allowed.

Lastly, the γ field shows the objective of the shop. Here, some popular choices are:

C_{max}: the goal is to minimize the makespan, which is the latest completion time of all jobs.

L_{max}: minimize the maximum lateness of all jobs. The lateness is defined as the difference between the due date of the job and its completion time. The lateness can possibly have a negative value if it is completed before the due date.

T_{max}: minimize the maximum tardiness of all jobs. The tardiness is defined as the difference between the due date of the job and its completion time. However, it has a strictly non-negative value, and is set to 0 if the job is completed before its due date.

$\Sigma_i T_i$, $\Sigma_i w_i T_i$: minimize the *total* tardiness of all jobs. If w_i is not specified, all jobs have an equal weight, otherwise multiply each value by the job's weight.

Several of these scheduling characteristics are discussed by Pinedo (2022).

2.3. Flexible Job Shop

A special field of Job Shop scheduling is the Flexible Job Shop Scheduling Problem (FJSP). The FJSP is denoted in the α-field by FJ. In the FJSP, tasks of a job need not necessarily be processed on a specific machine, but can be processed on one of multiple machines. These machines may differ in speed, and as such, the FJSP can be regarded as a combination of types R and J for the α-field in the three-field notation. This is a realistic case for a factory where a number of parallel machines (one or more) are available for a specific task.

The FJSP is notoriously hard. Where the classical Job Shop Scheduling Problem (JSSP) deals with sequencing and scheduling of the tasks, the FJSP also needs to consider machine assignment.

The JSSP is already proven Strongly NP-hard (Garey et al. 1976), and by extension the FJSP is NP-hard as well. The RCPSP mentioned previously is also known to be NP-hard (Blazewicz et al. 1983).

The first publication on FJSP as we currently know it dates back to an article by Brucker and Schlie (1991) - back then, the problem was named Job Shop Scheduling Problem with Multi-Purpose Machines (JSSP-MPM). Brucker et al. considered a very restricted case with two jobs on two machines, and were able to derive a polynomial algorithm. However, only shortly after, the same problem with three jobs on two machines (also denoted as 3×2) was already proven NP-hard (Jurish 1992). The name Flexible Job Shop was coined soon after by Brandimarte (1993) and is now commonly used.

Dauzère-Pérès, Ding, et al. (2024) argue that the flexibility in the FJSP should be named *operation flexibility*, as to distinguish it from other types of flexibility. This operation flexibility is the typical flexibility used in the FJSP as discussed in the previous section ($\alpha = FJ$). However, there are two other types of flexibility. The first of these is *sequencing flexibility* ($\alpha = FSJ$), which relaxes some of the precedence constraints. The other type of flexibility is *processing flexibility* ($\alpha = FPJ$), which is defined as the availability of alternative process plans (also named alternative routes).

2.4. Modelling

With realistic cases quickly exceeding the 3×2 size, optimal solutions are hard to find. There are two main techniques for solving (or trying to solve) these larger instances: using generalized methods or specialized methods. The former are easier to formulate and are usually built to be able to find exact solutions, however, generally slow. The specialized methods (also known as (meta)heuristics) try to exploit certain features of the problem at hand and as such may be faster, but these sometimes come with the trade-off that they may not be able to find an optimal solution and are usually much more complex.

There are two popular generalized methods: (Mixed) Integer Linear Programming (denoted MILP) which has been around for a long time, and more recently Constraint Programming (denoted CP) has gained traction. Both of these methods are able to find exact solutions up to medium-sized instances in a decent time-frame, and feasible solutions for large instances that are not computed up to optimality in a certain time-frame (Dauzère-Pérès, Ding, et al. 2024).

However, many researches have opted for metaheuristic algorithms for certain (sub)problems in the scheduling field. There are many types of metaheuristics; we will quickly list a few of them. Brandimarte (1993) employed a Tabu Search algorithm; Najid et al. (2002) used Simulated Annealing. Li and Gao (2016) added a Hybrid Genetic Algorithm to a Tabu Search algorithm, a method that was subsequently improved by Chen et al. (2020) by using Reinforcement Learning.

2.5. Sequence-dependent Setup Times

One of the extensions investigated in this paper is the inclusion of Sequence-Dependent Setup Times (SDST).

To be complete, a distinction can be made between two types of SDST: separable and non-separable (Dauzère-Pérès, Ding, et al. 2024). If setup times are separable, this means that a machine can be set up during its idle time, even if the preceding task of the upcoming job is still being processed. In contrast, non-separable setup times mean that a machine can only undergo its setup time once the preceding task is released from its machine and is available to the machine in question. Özgüven et al. (2012) show what the impact of this distinction is. However, most of the references to SDST in literature silently imply that setup times are separable, which is a fair assumption for many industries.

SDST are used in different forms throughout literature. Heinz et al. (2022) for example study a parallel machine environment with setup times, but include servers (or: workers) in their model, often required to switch over a machine from one function to another. Similar to Lunardi et al. (2020), they find that CP is good, but they also provide heuristics to speed up the solver. Their key idea is to batch similar jobs (creating so-called job families), which locally minimize setup times and idle time. This strategy for heuristics is also used by Li, Zheng, et al. (2024), where orders that require the same colours of ink are grouped together. An interesting contrast to heuristics that warm-start the CP solution, is explored by Abreu and Nagano (2022): they hybridized their CP solutions with Large Neighbourhood Search as main routine and CP as subroutine. This turned out to be an effective method for larger instances.

2.6. Alternative Process Plans

Another important part of this research is the inclusion of alternative process plans (APPs), also referred to as Alternative Routing (Ali et al. 2025), or Automated or Integrated Process Planning (Lin et al. 2020). In principle, these APPs are only concerned with an *or*-operation: choose plan A *or* plan B. Commonly, APPs are generalized to include *and*-actions as well: for instance in the printing environment, the *and*-action would be used in the production of a book. Both the cover and the book block must be produced, but these can be produced in parallel, there is no need for these to be sequential. To that end, a (dummy) *and*-node can be used with multiple successors. While this is strictly speaking not the essence of APPs, it is useful to capture this generalization.

In early literature on this topic, it was already noted that APP-extension can be computationally expensive (Kusiak and Finke 1988). Throughout literature, representations of alternative process plans have varied. Roughly speaking, there are two variants of alternative process planning (Dauzère-Pérès, Ding, et al. 2024):

- $\alpha = FPFJ$: a job is defined by multiple linear routes, of which only one should be selected for sequencing and scheduling. This can also be viewed as an enumeration tactic. This typically does not use *and*-nodes.
- $\alpha = FPFSFJ$: a job is defined using an And/Or graph instead of a number of linear routes. Here, operations can have multiple predecessors or successors.

An example of the former can be found in a paper by Kusiak and Finke (1988): they enumerated all process plans by a vector that holds the information for each process plan. However, two more interesting variants are Petri nets and And/Or networks, which are closely related.

And/Or networks were used first in a paper by Kis (2003). There, the name And/Or-graph was only subtly used. It then re-emerged from the field of RCPSP for use in Alternative Process Plans more recently in 2017, inspired by Activity-on-Node networks and And/Or trees used in artificial intelligence (Tao and Dong 2017). The authors initially used Simulated Annealing to find a solution to the instance as a whole, but in a later research decomposed the problem into a master problem to determine the selected tasks and a sub-problem to generate a schedule, using Tabu Search to solve the master problem (Tao and Dong 2018). Another research using Activity-on-Node networks used a Genetic Algorithm as a solver (Servranckx et al. 2024). In the RCPSP, there are two other useful reports on Alternatives: Hauder et al. (2020) perform a comparison between MILP and CP on the same Activity-on-Node networks, and finds that CP yields better results. R. Čapek et al. (2015) use a slightly different Nested Temporal Network with Alternatives: the core functionality is identical, but it includes some extra details. The authors implement an MILP solver with additional heuristic algorithm and yield great results.

A less frequently used, but similarly useful idea is a Petri net. Petri nets are a strong formalism to model or simulate parallel, sequential and optional tasks at once. In a Petri net, Or-nodes may be present - these imply that only one of their successive nodes must be selected. Thus, the Or-nodes are able to encode alternative chains in a process plan, resulting in different process plans. Čapek et al. (2012) for example show that Petri nets can be encoded into a matrix and solved by an ILP. There are also examples of using Petri nets with CP, however dated (Richard et al. 1995) (Boutet and Motet 1998). Overall, Petri nets do not turn up in competitive results for scheduling problems.

3 Problem description

3.1. Notation

Set	Description
$\mathcal{J}$	Set of jobs - $\mathcal{J}_j$ is the j'th job of $\mathcal{J}$
$\mathcal{O}_j$	Set of tasks of job $\mathcal{J}_j$ - O_{jk} is the k'th task of $\mathcal{O}_j$
$\square$	Empty (dummy) task
$\mathcal{A}_{jk}$	Set of allocation options for $\mathcal{O}_{jk}$ - A_{jkm} allocates $\mathcal{O}_{jk}$ on $\mathcal{M}_m$ with processing time p_{jkm}
$\mathcal{M}$	Set of machines - $\mathcal{M}_m$ is the m'th machine of $\mathcal{M}$
$\mathcal{M}_{jk}$	Set of eligible machines for operation O_{jk}
$\mathcal{N}_j$	Network capturing And/Or-relations between tasks $\mathcal{O}_j$

3.2. Description

Here, we will discuss the necessary elements of our problem from a formal standpoint.We will use the FJSP as the basis of our model. Referencing the three-field notation as introduced in Section 2.2, this means we set the machine environment to $\alpha = FJ$.

Jobs In the FJSP, there is a set of jobs $\mathcal{J}$ to be scheduled on a set of machines $\mathcal{M}$. Each job $\mathcal{J}_j$ is defined by a set of tasks or operations $\mathcal{O}_j$. A task is not necessarily atomic (such as printing a single sheet), but rather the compound of identical atomic actions (e.g. printing all covers for a book order). All tasks $\mathcal{O}_j$ are represented by an And/Or-network $\mathcal{N}_j$, as discussed in Section 2.6. In this network, all tasks are represented using nodes. However, as it includes alternative process plans, not all nodes need to be selected to produce the job.

To be more specific, an And/Or-network consists of And-nodes and Or-nodes as well as dummy start/end nodes. The reason for these dummy nodes will be clear when the model is discussed. If an And-node is selected as part of the path, all of its successors must be selected as well ($\wedge$). If an Or-node is selected, exactly one of its successors must be chosen ($\vee$). As a result, choices between process plans are encoded using Or-nodes. An example can be seen in Figure 3.1 for the third job of the example in Section 1.7. Note: a task that can be processed by one of multiple machines could be modelled using Or-nodes as well, but we choose to group these for easier understanding and to be consistent with the model used in the experiments.

The alternative process plans are assumed given (e.g. generated by the system upon order placement). Therefore, for each job j there is a network $\mathcal{N}_j = (V_j, A_j)$ that represents the And/Or-network.

Tasks Every task O_{jk} in the And/Or-network of a job is specified by a tuple $(\mathcal{A}_{jk}, P_{jk})$. $\mathcal{A}_{jk}$ corresponds to the set of machines available for processing this task, with their respective processing times P_{jk}, and can simply be regarded as a list of tuples (m, p). Tasks cannot be

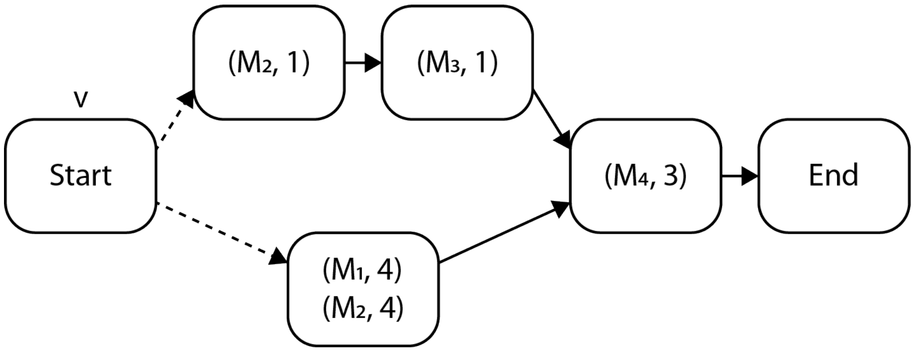

Figure 3.1.: And/Or-network for example job. The dummy start node is an Or-node, indicated by the ∨ and dashed outgoing arcs.

pre-empted nor interrupted: once they start, they must be fully processed. The task may be blocking: in that case, if a machine finished processing a task, that task remains on that machine until the product can move on to the next machine.

Machines The set $\mathcal{M}$ represents the available machines in the shop. These machines may be multifunctional, for instance being able to print on multiple sizes of materials, or using different techniques for different materials. Between operations that use different of these functionalities, setup times may occur. As machines cannot instantaneously switch between functions, for any pair of tasks (say α and β) that are allocated to the same machine and that are their immediate successors on that machine, the start time of β is at least the end-time of α plus a change-over time from α to β. This setup time is set to 0 if the tasks require the same functionality. The setup time for the first operation on each machine is ignored. Lastly, it is assumed that the machines are always operational, i.e. they do not have (un)scheduled downtime or breakdowns.

Blocking tasks Commonly, machines have either an input- or an output-buffer. For instance, a printer can store a stack of papers that it just printed. While some shops have a certain number of places available to hold partial products in a separate place on the shop floor, for simplicity we will assume that there are no buffers besides the output buffer of each machine. As a result, a machine is blocked from processing further tasks until its output is cleared - or, in other words, a machine idles until the next task of its last job can be processed by that task's allocated machine. Manufacturing equipment may have both input and output buffers; however, we assume that there is just one buffer, as that appears to be the common choice in literature.

Objective The objective is to minimize the makespan of the fulfilment of the orders.
As rush jobs may occur – e.g. a priority order must be scheduled during the current shift, or a machine failure resulting in a failed order production means that an order must be re-produced in the shift before the order deadline is exceeded – the goal is to quickly (in a few minutes) produce a decent (sub-optimal) schedule. This allows an operator to start up the shift or get a drink, and be able to start production upon return.

Three-field notation The setting described here can be characterized as $FJ|prec, s_{ijk}, block|C_{max}$. $\beta = prec$ is due to the precedence constraints; $\beta = s_{ijk}$ denotes the presence of sequence-dependent setup times, and $\beta = block$ signals the blocking nature of tasks. $\gamma = C_{max}$ characterizes the optimization of the makespan. The alternative process plans are denoted by either $\alpha = FPFJ$ or $\alpha = FPFSFJ$ depending on the method chosen.

4 | Constraint Programming

4.1. Background

As discussed in the literature review (Chapter 2), Constraint Programming is a very powerful scheduling solver nowadays. While traditionally MILP solvers are good at scheduling, in general there is a trend that CP beats MILP (Naderi, Ruiz, et al. 2023). Many recent studies and literature reviews support this conclusion, however specialized MILP solutions with heuristics for specific scenarios may still yield better results (Laborie 2018), (Dauzère-Pérès, Ding, et al. 2024).

A case study that is closely related to our setting is probably the Online Printing Shop problem, as proposed by Lunardi et al. (2020). They discuss a setting that, in a number of aspects, matches our problem, among which the inclusion of sequence-dependent setup times. The focus of their paper however is *machine unavailability*, here not considered - they in turn do not use alternative process plans. Lunardi et al. provide both a MILP and CP model, and conclude that their CP model yields much better results.

A useful feature of Constraint Programming in general is its *anytime* ability. This term was coined by Dean and Boddy (1988), and in essence boils down to this: a solver may find some initial solution, and while it keeps running, it will incrementally try to improve this solution until it can prove optimality or a runtime cut-off occurs. Especially for larger instances, this is welcome, since it takes very long to prove optimality: being able to stop the solver at any point with a suboptimal schedule is always better than no schedule at all. Note that MILP models are also anytime; this is not a feature exclusive to CP models.

Overall, with CP proven to be a powerful general framework for scheduling, in this chapter we will put together an FJSP model with the discussed extensions. Specifically, we define a set of constraints for IBM's CP Optimizer (CP Optimizer for short). An equivalent CP model for Google's OR-Tools CP-SAT (OR-Tools for short) is available in Appendix A.

4.2. Notation

The notation used throughout this chapter builds upon the notation used in Section 3.1 and is extended with additions from Naderi and Roshanaei (2021).

4.3. Flexible Job Shop model

In this section, we first present the basic model for a Flexible Job Shop ($\alpha = FJ$, $\beta = \emptyset$, $\gamma = C_{max}$), and subsequently incrementally expand the model to include more extensions.

Description	
Parameters	
Task^*_{jk}	An interval variable whose domain all interval variables Task_{jkm}
Task_{jkm}	An interval variable corresponding to an allocation A_{jkm}
p_{jkm}	The processing times of operation O_{jk} on machine m
V	A very large positive number, representing ∞
Functions	
`BinaryVar()`	Returns a binary variable with value 0 or 1
`IntervalVar(p, Task`$^*_{jk}$`, Optional)`	Returns an optional interval variable of size $s = p$ (processing time) on the domain $[0, V)$ synchronizing its properties with $\texttt{Task}_{jk}$
`IntervalVar([p, q))`	Returns an optional interval variable of size $p \leq s < q$ on the domain $[0, V)$
`StartOf(a)`	Returns the start of interval variable a
`EndOf(a)`	Returns the end of interval variable a
`Alternative(a, B)`	Creates an alternative constraint between interval variable a and the set of subsequent interval variables B: one variable of B must be present as successor to a
`NoOverlap(B)`	Constrains a set of interval variables B not to overlap each other
`EndBeforeStart(a, b)`	Ensures `EndOf(a)` $\leq$ `StartOf(b)`
`StartBeforeEnd(a, b)`	Ensures `StartOf(a)` $\leq$ `EndOf(b)`
`PresenceOf(a)`	True (1) if a is present, False (0) if a is absent

A plain FJSP model consists of the following goal and constraints[1]:

$$\text{minimize} \quad C_{\max} \tag{4.1}$$

$$\text{subject to} \quad \text{Task}^*_{jk} = \text{IntervalVar}(\big[\min_{m \in \mathcal{M}_{jk}} p_{jkm}, \max_{m \in \mathcal{M}_{jk}} p_{jkm}\big]) \qquad \forall j \in \mathcal{J}, k \in \mathcal{O}_j \tag{4.2}$$

$$\text{Task}_{jkm} = \text{IntervalVar}(p_{jkm}, \text{Task}^*_{jk}, \text{Optional}) \qquad \forall j \in \mathcal{J}, k \in \mathcal{O}_j, m \in \mathcal{M}_{jk} \tag{4.3}$$

$$\text{Alternative}(\text{Task}^*_{jk}, \{\text{Task}_{jkm} : m \in \mathcal{M}_{jk}\}) \qquad \forall j \in \mathcal{J}, k \in \mathcal{O}_j \tag{4.4}$$

$$\text{EndBeforeStart}(\text{Task}^*_{jk-1}, \text{Task}^*_{jk}) \qquad \forall j \in \mathcal{J}, k \in \mathcal{O}_j \tag{4.5}$$

$$\text{NoOverlap}(\text{Task}_{jkm} : j \in \mathcal{J}, k \in \mathcal{O}_j | m \in \mathcal{M}_{jk}) \qquad \forall m \in \mathcal{M} \tag{4.6}$$

$$C_{\max} = \max_{j \in \mathcal{J}}(\text{EndOf}(\text{Task}^*_{j|\mathcal{O}_j|})) \tag{4.7}$$

Constraint 4.2 ensures that there exists a variable for each task. Constraint 4.3 then ensures that for each allocation on each machine, an optional variable exists with the corresponding

[1] `https://ibmdecisionoptimization.github.io/docplex-doc/cp/refman.html`

processing time. These allocation variables are combined in Constraint 4.4 to ensure that exactly one allocation is selected per task, because tasks and machines are unitary as discussed in Section 2.1. Constraint 4.5 imposes the precedence constraints of tasks in a job: the previous task must be processed before the next one can start (this constraint does allow $\text{StartOf}(\text{Task}_{jk}) = \text{EndOf}(\text{Task}_{jk-1})$). The unitarity of machines is constrained in 4.6. Finally, the makespan is defined in Equation 4.7 as the latest end of each job's final task variable; the minimization of the makespan is the goal of the model (4.1).

4.4. Sequence-Dependent Setup Times — $\beta = s_{jk}$

When SDST are activated, the 'NoOverlap' constraint is modified to include a transition matrix:

$$\text{NoOverlap}(\text{Task}_{jkm} : j \in \mathcal{J},\ k \in \mathcal{O}_j \mid m \in \mathcal{M}_{jk},\ M_m) \qquad \forall m \in \mathcal{M} \tag{4.8}$$

This transition matrix M_m ensures that a specified minimum delay (≥ 0) occurs between any two subsequent tasks on machine m. M_m is of size $|\mathcal{O}| \times |\mathcal{O}|$ and contains a value for any pair of tasks. If there is a setup time between two tasks on this machine, the new task is delayed by this value and the machine must idle. This may have impact on the predecessor of the new task if it is blocking, as this predecessor must stay buffered in its machine until the setup time has passed. If there is no setup time between two tasks, the corresponding entry in the matrix is zero and the new task can start at once.

4.5. Blocking tasks — $\beta = block$

For tasks that block a machine, Constraint 4.3 must be modified to relax the fixed processing duration:

$$\text{Task}_{jkm} = \text{IntervalVar}([p_{jkm}, V), \text{Optional}) \qquad \forall j \in \mathcal{J},\ k \in \mathcal{O}_j,\ m \in \mathcal{M}_{jk} \tag{4.9}$$

In this modification, the minimum size of the variable is set to the processing time as before, but the maximum size is now set to infinite, as this task may block the machine for a while longer as described in Section 3.2. By extent, Constraint 4.2 is modified similarly. Moreover, a successive task within a job is now only allowed to start exactly when the preceding task is released from its machine, unblocking that machine. Therefore, an extra constraint is added:

$$\text{StartBeforeEnd}(\text{Task}^*_{jk}, \text{Task}^*_{jk-1}) \qquad \forall j \in \mathcal{J},\ k \in \mathcal{O}_j \tag{4.10}$$

Constraint 4.10 is the reverse of 4.5, which results in a tight coupling of Task_{jk} and Task_{jk-1}. The resulting behaviour is that the predecessor stays in the buffer of its machine, thereby blocking it, until the successor starts.

4.6. Alternative Process Plans — $\alpha = FPFSFJ$

APPs are, as per Section 3.2 defined by And/Or-networks. In contrast to the topological ordering of activities, the precedences are now captured by the arcs in the network. Consequently, Constraint 4.5 must be reformulated to look at the dependencies in the graph instead:

$$\text{EndBeforeStart}(\text{Task}^*_{ja}, \text{Task}^*_{jb}) \qquad \forall j \in \mathcal{J},\ (a, b) \in \mathcal{N}_j \tag{4.11}$$

If a task is a blocking task, the StartBeforeEnd constraint must be modified similarly.

Now, a solution should always select exactly one process plan for each job. There are two options as to how these choices can be implemented: using constraints on nodes or using constraints on arcs.

Constraining node selection This strategy exploits the *presence* of tasks (nodes). The first task of each job is marked as required, and all the job's subsequent tasks are marked as optional to the solution. Then for each task, a constraint is added that if this task is part of the solution, one of its successors must also be present. This models Or-nodes where one process plan must be selected, as well as And-nodes with one successor. (It is possible to force all its successors to be present e.g. in case of disassembly.)

CP solvers implement a so-called IFTHEN-constraint that nicely captures this behaviour: *if* a certain condition is met (presence of a task), *then* the statement (presence of a successor) is imposed. If a process plan is selected, this constraint is easily propagated through all successive tasks on this path. This constraint comes with a caveat however: it does not constrain the behaviour of the statement if the condition is not met (and as such, a successor may be marked as present by the solver even if its predecessor is not). This leads to false inclusions and redundant branches in the search process. In practice, this results in significant overhead during the solving process (testing indicated a roughly 3x-5x performance penalty compared to the next strategy). This cannot simply be solved by applying a negated constraint where the absence of a task requires its successors to be absent. For if a task is not present, its successor may still be present in case both plans come back to the same tasks later in their process plan. This only works when all process plans are fully enumerated. However, this solution is not as clean and powerful as the arc selection strategy. For this reason, we do not use this technique.

Constraining arc selection In this strategy, not the presence of predecessors dictates the presence of successors, but rather a 'flow' across arcs forces this presence. We lend ideas from flow networks, where a flow must exist through the network from one place to another. To that end, we now discern three types of tasks: a *source* s, a *sink* t and a normal task. For each job, a dummy source and sink are added to the network, with arcs from the source to the original starting task(s), and arcs from the final task(s) to the sink. These sources and sinks are mandatory to be present in the solution. They all have a processing time $p_{js} = p_{jt} = 0$ on a dummy machine 0. The CP solvers are able to stack 'overlapping' intervals with size 0 within one time unit. Hence, these dummy tasks do not affect the solution. All tasks in the And/Or-graph are normal tasks. Just as for the node selection strategy, all these tasks are now marked as optional.

To guarantee that exactly one process plan is selected, a flow of value 1 should exist from each job's source to its sink. Accordingly, flow variables are added for all arcs in the network. For the source nodes, the flow should be $+1$ (only outgoing); for sinks, it should be -1 (only incoming). For all other nodes, the incoming and outgoing flow should be equal to the presence of the task: if the task is part of the solution, then there must be a flow across this node; otherwise, there should be none.

With respect to the model, Constraint 4.2 is modified to mark all tasks as optional:

$$\text{Task}^*_{jk} = \text{IntervalVar}([\min_{m \in \mathcal{M}_{jk}} p_{jkm}, \max_{m \in \mathcal{M}_{jk}} p_{jkm}], \text{Optional}) \quad \forall j \in \mathcal{J}, k \in \mathcal{O}_j \tag{4.12}$$

Then, the source and sink variables are added with processing time 0:

$$\text{Task}_{js} = \text{IntervalVar}(0) \quad \forall j \in \mathcal{J} \tag{4.13}$$

$$\text{Task}_{jt} = \text{IntervalVar}(0) \quad \forall j \in \mathcal{J} \tag{4.14}$$

Next, flow variables are added for all arcs in the network:

$$\text{Flow}_{juv} = \text{BinaryVar}() \quad \forall j \in \mathcal{J}, (u, v) \in \mathcal{N}_j \tag{4.15}$$

Finally, the flow constraints are imposed:

$$\sum_{(s,u)\in\mathcal{N}_j} \text{Flow}_{jsu} = 1 \quad \forall j \in \mathcal{J} \tag{4.16}$$

$$\sum_{(v,t)\in\mathcal{N}_j} \text{Flow}_{jvt} = 1 \quad \forall j \in \mathcal{J} \tag{4.17}$$

$$\sum_{(u,v)\in\mathcal{N}_j} \text{Flow}_{juv} = \text{PresenceOf}(\text{Task}^*_{jv}) \quad \forall j \in \mathcal{J}, \forall v \in \mathcal{N}_j \tag{4.18}$$

$$\sum_{(v,w)\in\mathcal{N}_j} \text{Flow}_{jvw} = \text{PresenceOf}(\text{Task}^*_{jv}) \quad \forall j \in \mathcal{J}, \forall v \in \mathcal{N}_j \tag{4.19}$$

Constraints 4.16 and 4.17 model the flow out of the source and into the sink respectively. Constraints 4.18 fix the flow into a task to be identical to the task's presence: if the task is present, there should be a path leading up to this task; otherwise, there should not be a flow into this task. Constraint 4.19 forces the subsequent flow out of this task, similar to the flow into the task: if the task is present, there should be a path to the sink.

An important note is that these constraints assume a single path from source to sink: there are no assembly or disassembly activities. However, extending the model to include such activities can be implemented rather straightforwardly by specifying a certain flow value (other than 1) into and out of each node.

4.7. Warmstart

Both CP Optimizer and OR-Tools support an optional *warmstart*. A warmstart means that the solver is seeded with a feasible solution. This immediately provides a starting point and upper bound which kickstarts the solver, potentially decreasing the runtime.

4.8. Summary

Overall, we see that the basic properties of the FJSP are captured with a handful of constraints. These constraints can be modified and/or extended quite easily, although there is quite some administrative work involved to enforce the constraints between all jobs, tasks and allocations. With the constraints described in this chapter, a complete $FPFSFJ|prec, s_{ijk}, block|C_{max}$ model can be constructed.

The nice part about a CP model is that it only requires a set of boundaries, it does not need to know how to create a feasible and optimal solution. This is all handled internally by constraint propagation and objective optimization.

5 Multivalued Decision Diagrams

5.1. Background

Anticipating the computational results from Section 6.2, it is clear that sequence-dependent setup times have a significant impact on the runtime of the CP model, even when the instance size remains the same. With the result becoming more noticeable for larger instances, and the open-source OR-Tools clearly lagging behind the commercial CP Optimizer, we want to explore a different technique. Here, we will explore the use of Decision Diagrams. In recent years, more attention has gone to these (Multivalued) Decision Diagrams, which can both be used as an exact solution method, but also for quickly finding feasible solutions. We will explore this method and evaluate whether it yields useful results.

For a full in-depth explanation on Decision Diagrams, we refer the reader to the dissertation by Bergman et al. (2016) with a compact version available by van Hoeve (2024). Here, we will provide a brief introduction to the necessary details for our model.

The basic idea of a Decision Diagram (DD) stems from a decision tree. Given a set of binary variables, one can build a decision tree with all possible combinations. Such a tree, and therefore a DD, consists of multiple *layers*: in layer i, i decisions have been made, with $i = 0$ being the root node. Traversing the path from the root to any terminal node (a node in the final layer) will yield an assignment of all variables. In principle, a decision tree is a DD. If the decision space consists not only of binary variables, but includes discrete variables with more than two choices, the DD turns into a Multivalued DD (MDD), with each node possibly having more than two children. Just as a decision tree can have costs associated with the arcs connecting nodes on consecutive layers, a Decision Diagram usually includes a certain cost on its arcs.

A few things set DDs apart from simple decision trees. Firstly, there is a notion of 'domination' between nodes such that the tree can be pruned. This is explained later. Moreover, there are options to remove and/or merge nodes. Instead of expanding all variables into a full DD, certain assignments may be removed because they do not contribute to a solution, or certain nodes may be merged to reduce the DD size favouring faster computation time at the cost of introducing some infeasibilities. There are three flavours of DDs:

Reduced In a reduced (also called exact) decision diagram, the paths from root to terminal nodes represent all unique solutions. To achieve this, during construction, all duplicate nodes in the DD are merged. Duplicate nodes are those that have made the same (number) of decisions so far in the construction (possibly in a different order, depending on the problem specification). Merging nodes means that the incoming arcs to these nodes are all redirected to a single (new) node, and the duplicate nodes are removed. Even though it is called 'reduced', a reduced DD may still be very large in size, depending on what defines a unique solution (for example: a sorted set of values yields many fewer unique solutions than an unordered set).

Restricted In a restricted decision diagram, all paths from root to a terminal node form a unique solution, but not necessarily all solutions are present. Commonly, one defines a maximum width w for the diagram: if this width is exceeded in the diagram, nodes are removed until the diagram is at most w wide. This type of diagrams can be useful in situations aimed at satisfiability, where any solution is good enough and it need not be optimal (Bryant 1986). Their smaller size means that they can be traversed much quicker. For optimization, they can be used to find an upper bound (in case of a minimization problem; for a maximization problem it will find a lower bound) in a short time, but a restricted DD is less likely to yield an optimal solution.

Relaxed In contrast to restricted DDs, a relaxed DD may include paths from root to terminal node that may not be a feasible solution. Where a restricted DD has valid solutions removed, the construction of a relaxed DD may insert solutions that do not satisfy the original requirements and therefore are infeasible. In a relaxed DD, all nodes that cause the DD to exceed the width w are not removed, but merged instead. This action is performed by a *merge operator*: it applies a relaxation on the states, and all ingoing and outgoing arcs are directed to this new node. This merge operator may need to act on nodes with non-identical states. In that case, it will need to decide on a new state. This new state may be a combination of the old states, which may introduce infeasibilities. This infeasibility could be that a precedence constraint is broken or that two tasks overlap on a certain machine. Hence, a path that includes relaxed nodes may be too optimistic (for a minimisation problem). As a result, a path from root to terminal node that includes relaxed nodes provides a lower bound on the solution. Relaxed decision diagrams contain a superset of all solutions: all feasible solutions are always present, with infeasible solutions added due to the relaxation. With respect to size, it is larger than a restricted diagram since there are many more arcs, but smaller than an exact diagram since the number of nodes is substantially lower.

A Decision Diagram can either be exact or restricted and/or relaxed. Any DD that is relaxed or restricted is not exact, but a DD can be both restricted and (partially) relaxed.

Traversing a DD once yields a solution (or a bound on the solution). However, just like in Constraint Programming, the DD is effectively a Search Tree. As such, it can be used for optimization by repeatedly traversing the tree, either exhaustively to find the lowest (highest) value, or by closing the gap between the lower and upper bound. This means that it can be used as an anytime-technique (Fontaine et al. 2023), just as CP. For optimization, relaxed DDs are most useful, as their smaller size makes them much more tractable, while still being able to provide dual bounds which can improve with longer runtime. Restricted diagrams are also small, but cannot provide a dual bound.

5.2. Structure and functions

A Decision Diagram can be constructed by defining a state and a handful of functions that work on this state. We will briefly discuss the essentials here, and show their implementation in the next section.

> The `Knapsack` problem will be used as a running example. In this problem, layer i corresponds to item i, where the decision can be to include item i or not. This is therefore a Binary Decision Diagram.

State The *state* is the most crucial piece of a DD. Each node in the DD is defined by a state, which holds the variables corresponding to the chain of decisions made to get to this node. As such, it is a stepping stone that can be traversed to find a path from the root of the DD to a terminal node. The contents of the state are completely dependent on the problem.

> A `KnapsackState` holds the remaining capacity of the knapsack, and the number i of the current layer. Note that the state does not need to include the selected items nor the associated prize: this is known from the traversed arcs. However, to easily recognize dominance later, the state also tracks the accrued prize.

Decision function To progress the construction of a DD, we need to know to what states we can go to from a certain state. To this end, a *decision function* must be specified: given a state, it returns the decisions that can be made.

> The function `Decision(KnapsackState)` checks the weight of the next item i. If its weight is equal to or lower than the remaining capacity, the decision function returns both `Yes` and `No`. If the weight exceeds the remaining capacity, it only returns `No`.

Transition function The *transition function* is used to construct the DD. This function takes a state and a decision, and calculates the new state that results from applying the decision to the previous state.

> The `Transition(KnapsackState, DecisionValue)` function increases the depth of the state by 1. Then it checks whether the value of the decision is `Yes` or `No`. If `Yes`, the remaining capacity is decreased by the weight of item $i + 1$. If the decision is `No`, the remaining capacity remains unaltered.

Cost function With a DD constructed, one can traverse it from the root to a terminal node to find a solution. To measure the quality of a solution, a cost can be associated with each arc in the DD. The *cost function* takes two states (before and after a transition) and calculates the cost associated with going from the former to the latter state. The cost depends on the objective of the problem.

> The function `Cost(KnapsackState, DecisionValue)` returns 0 if the decision is `No`. If the decision is `Yes`, the cost (in this case 'prize') is the prize of item $i + 1$ (i is available as a property from the state).

Dominance function To achieve a reduced DD, a *dominance function* must be implemented that specifies whether a certain state dominates another. Dominance is defined only between similar nodes: those that have made the same (number) of decisions (depending on the problem). A state dominates another if all its 'coordinates' are strictly equal or better. For the simple Knapsack problem, these are the profit and capacity: if the same number of items is taken and the capacity dominates (equal or more remaining) and the profit dominates (equal or higher), this state dominates. Note that dominance cannot simply be defined between nodes in a different layer (different number of decisions made), as these states have a different number of decisions still to be made.

> The function `Dominance(KnapsackState A, KnapsackState B)` checks if `A` dominates `B` or vice versa. A `KnapsackState` dominates another iff, with the same number of decisions made (equal depth), the state has an equal or higher remaining capacity and an equal or higher cost (prize). This means that the state dominates also if both capacity and cost are equal, since there is no point keeping both.

Merge function The *merge function* takes two or more states and returns the new state that results from merging the given states by applying a merge operator. The merge operator may 'mix' states to generate a lower or upper bound. For duplicate nodes with dominance, the merge operator should always return the dominating state.

> The `Merge([KnapsackState])` function takes a list of states, and returns the state with the highest remaining capacity, as this safely provides an upper bound on the optimal solution. In case of a tie based on capacity, it returns the state with the highest cost (as the `Knapsack` problem is a maximization problem). It does not need to merge states.

Ranking function (opt.) Optionally, a *ranking function* can be defined for constructing a relaxed DD. This function uses a heuristic to select nodes for merging if the specified width is exceeded in a layer. For example, the worst states can be merged, thus keeping the best states intact as they are more likely to generate a good solution.

> The function `Rank(KnapsackState A, KnapsackState B)` compares the states `A` and `B`. A state ranks higher (better) if it has a higher remaining capacity. In case of a tie, it ranks higher if it has a higher cost.

5.3. MDDs for Scheduling

MDDs have been used in scheduling before. They can be used quite easily for single-machine scheduling: in that case, the algorithm only needs to sequence the jobs (or tasks). Moreover, the merging operator is quite powerful, as it can group all nodes that have processed the same set of jobs (even when SDST are present, because it can remember the optimal path to this node, and for the future layers, the order of the previous jobs does not matter). This is for example illustrated in Bergman et al. (2016), used in Cire and van Hoeve (2012), and a variant on it is used by Matsumoto et al. (2018). They show that in certain cases or settings, MDDs can be (significantly) faster than MILP or CP models. For example, when SDST are involved with relatively large values compared to processing times, MDDs can get up to orders of magnitude speed improvements.

The problem becomes more challenging however when looking at a job shop with more than one machine. Solutions are readily available for shops with parallel machines of identical functionality and speed; in this case, two states can be compared by sorting the F-vector (containing the current machine times) and for example picking one with the lowest maximum - there is no distinction between the machines anyway. For that case, solutions are demonstrated by van den Bogaerdt and de Weerdt (2019) and Kowalczyk and Leus (2018). However, for the makespan minimization problem with multiple unrelated machines, van den Bogaerdt (2018) shows that an MDD implementation yields extremely poor solving speeds. To find out how their MDD formulation compares to our CP model for the FJSP, we decided to implement this method.

5.4. Framework

The MDD structure as described in this chapter is implemented in DDO (Gillard et al. 2020). DDO is a multithreaded MDD-framework written in Rust with an interface that offers all discussed functions. There is one minor caveat in using this framework: DDO only offers support for maximization problems. The solution is to flip the signs of the model: instead of working with positive processing times, we use negative processing times. As a consequence, we want to find the least-negative makespan - this is now a maximization problem! To be consistent with the

CP model however, we will discuss the implementation in this chapter as a usual minimization problem.

During runtime, DDO tries to build an exact decision diagram. This can grow very large though for instances with a large search space. To make the MDD construction more tractable, DDO can revert to a restriction and/or relaxation: the width must be limited. With a limit for the width, the construction happens in two passes: first, DDO builds a restricted diagram; secondly, it applies a relaxation.

The restricted diagram is constructed layer by layer from top to bottom. Starting from the root, all possible decisions for the next layer are considered. If a specified width w is then exceeded on the next layer (there are more than w nodes), all nodes from that layer except the best w nodes are discarded. The nodes that are discarded are those with the worst ranking, according to the specified ranking function: they are least likely to generate a good solution. The discarded nodes are not removed completely: they are put onto a 'fringe': the unexplored border around the diagram, to be expanded during the relaxation phase. In the restricted phase, DDO can only report a *primal* (or: upper) bound for the solution: any path from root to terminal node is a valid primal bound; it cannot provide a lower bound since only a subset of the paths is present in the diagram.

Once the restricted diagram is finished, the relaxation phase starts. In this phase, nodes on the fringe will be expanded. In principle, this phase considers all fringe nodes across all layers in a priority queue according to the same ranking function. However, as the width of the diagram is already saturated from the restricted phase, it can only add additional arcs or merge 'new' nodes with existing nodes, which is likely to introduce infeasibilities. For this reason, a relaxation rarely contributes to an useful improvement on the solutions. The most useful feature of a relaxed diagram is that it may provide a lower bound on the optimum; a true lower bound can be found once the fringe is emptied completely and therefore all solutions are present (as well as infeasible solutions). If the construction of the relaxed diagram takes longer than a specified runtime time-limit cut-off, not all paths have been explored, and therefore there is no true lower bound - there could be better paths than those present in this restricted diagram. Thus, if the relaxation phase ends prematurely due to a time limit cut-off, the reported lower bound cannot be taken as a true lower bound.

5.5. Flexible Job Shop model

To model the FJS in a reduced DD, we define the state $S = (V, F, T)$. We follow the naming scheme from van den Bogaerdt (2018). Here, $V \in (O_1 \cup \{\square\}) \times \cdots \times (O_n \cup \{\square\})$ is the frontier of the tasks for each job in $\mathcal{J}$ - note that $\square$ means that no task has been scheduled yet for this job (see Section 3.1). This set V is required to keep track of precedence constraints by recording the last selected task within each job. Initially, all entries $V_j = \square$ since no task is processed. $F \in \mathbb{R}^{|M|}$ corresponds to the latest completion times of all machines in $\mathcal{M}$, and initially starts with all values set to 0. $T \in \mathbb{R}^{|J|}$ corresponds to the latest completion times of the tasks in V, and similarly starts off with all values set to 0, for each job in $\mathcal{J}$. The T-vector is required to check when the previous task within a job has finished processing, before the next task can be scheduled: a task can only start once both the machine and the previous task of this job (if applicable) are finished.

The decision function is rather straightforward: given a vector V (from a state S), it returns all options from $O = \bigcup_j O_j$ that are permitted given the current precedence constraints from V. To that end, it lists all next tasks for each job (if the job is not completed yet), and subsequently lists all available machines for that task. Therefore, the set of decisions it returns is a list of allocation options A_{uvw}: the v'th task from job u allocated on machine w. All these decisions will result in a new node in the next layer of the diagram.

The transition function is also quite easy: given a state S and a decision O_{uvw} (with corresponding processing time p_{uvw}), it calculates the modified state $\overline{S} = \left(\overline{V}, \overline{F}, \overline{T}\right)$. $\overline{V}$, $\overline{F}$ and $\overline{T}$ are defined to be

$$\overline{V_i} = \begin{cases} O_{uv}, & \text{if } i = u. \\ V_i, & \text{otherwise.} \end{cases} \tag{5.1}$$

$$\overline{F_i} = \begin{cases} \max(F_w, T_u) + p_{uvw}, & \text{if } i = w. \\ F_i, & \text{otherwise.} \end{cases} \tag{5.2}$$

$$\overline{T_i} = \begin{cases} \max(F_w, T_u) + p_{uvw}, & \text{if } i = u. \\ T_i, & \text{otherwise.} \end{cases} \tag{5.3}$$

The cost function is defined as the difference between the makespan in the new state and the makespan in the old state (given the objective to minimize the makespan). This can be calculated as $c(F, \overline{F}) = \max_{m \in \mathcal{M}}(\overline{F}_m) - \max_{m \in \mathcal{M}}(F_m)$.

The dominance function requires a notion of 'duplicate states'. Two states $S^1 = (V^1, F^1, T^1)$ and $S^2 = (V^2, F^2, T^2)$ are considered duplicates if $V^1 = V^2$. If they are duplicates, state S^1 is said to dominate S^2 iff $F^1 \leq F^2$ with the $\leq$-sign performing an element-wise minimum. S^2 dominates S^1 iff $F^2 \leq F^1$. In any other case, neither S^1 nor S^2 dominates and these states cannot be merged, because the 'best' state cannot be chosen unambiguously.

The merge function for a reduced DD is straightforward since it only merges duplicate (dominating and dominated) states. These states have an identical vector V, and the vectors F and T can be picked from the dominating state. As such, the merge function can simply return the dominating state.

The ranking function is not used for a reduced DD, as relaxations will not be applied.

With this formulation, the width of an exact MDD diagram becomes very large even for small instances. For an instance with 4 jobs with in total 12 tasks on 6 machines (with approximately 2 machines available per task), the width grows to over 21,000 nodes on the widest layer. The same instance with a fifth job with another three tasks yields a width of over half a million nodes on the widest layer, and a total number of nodes of over three million. Consequently, the required amount of memory for such an instance grows to roughly 30GB.

5.6. Restricting the FJS model

Creating a restricted FJS model is quite straightforward, as it only involves the removal of nodes. The main question here is: which nodes should be cut off? Ideally, the nodes that lie on the optimal path should be retained - however, knowing this during construction would require knowing the solution beforehand. Instead, we must retain the *most promising* node(s) on each layer. We are looking for the ranking function described previously: the node(s) with the best rank is/are kept; the others are discarded.

There are multiple options of guessing or estimating the promise of a node. The simplest option is to look at the current makespan: a node with a small makespan is more likely to yield a good solution than a node with a large makespan. Thus, nodes are simply compared by their current state only. However, for the relaxation phase, this may be suboptimal, as this ranking function is also used for the fringe during the relaxation phase (see Section 5.4).

The second option is to not look at the *current* makespan, but instead trying to estimate the total makespan given the current state and the decisions that still must be made. This strategy is similar to the classic A^* algorithm (Hart et al. 1968). In A^*, the estimator should never overestimate the real cost in order to be considered optimal (for a minimization problem). Such an estimator is called *admissible*, and provides a lower bound on the solution. However, in the FJSP, a task may be processed by one of multiple machines. And as the estimation must be calculated for each node, the complexity must be kept minimal to keep the runtime as fast as possible. An estimation with minimal runtime is typically a *greedy* one. A greedy estimation is not easily admissible though. Consider the following example: there are two machines and three jobs consisting of one task each. Each task can be processed on both machines: $[(M_1, 2) \vee (M_2, 4)]$. A greedy option would be to allocate the fastest option for each task. That would result in a makespan of 6 as all tasks are allocated on machine 1. However, the optimal schedule is to allocate two tasks on machine 1, and one on machine 2, yielding a makespan of 4.

Instead, as we are looking at *ranking*, which is relative to other nodes, a strictly admissible function is not truly required. It may be easier to get a consistent ranking the other way around: crafting an estimate for the upper bound. If good nodes are consistently ranked lower than others, this would still yield a good solution.

A consistent estimator function for the upper bound could be constructed as follows: given a state $S = (V, F, T)$, we know for sure that all tasks in V are scheduled with the corresponding machine times F. All tasks that are not in V still need to be scheduled. We schedule all of these tasks, each task on all of the machines that can process it, taking precedence constraints into account.

Consider the same example with two machines and three jobs. A decision to allocate the first job on machine 1 yields an estimated makespan of 8 (due to the other two tasks being scheduled on machine 2), while a decision to schedule this job on machine 2 would yield an estimated makespan of 12. The former state would be ranked better.

The order of scheduling tasks matters due to precedence constraints. To keep the complexity minimal, we propose two greedy ways to handle this:

1. Sort by jobs: process all jobs one by one, scheduling their remaining tasks consecutively.

2. Sort by tasks: process the next task from each job, repeating this until all jobs are completed.

For either alternative, the previous task of a job must be finished on all its machines before the next task can start. Both these techniques have a complexity that scales with the input size - $O(n)$ - as they consider all items O_{jkm} at most once.

Option 1 may, depending on the number of machines that can process a task, result in quite a poor estimate as gaps are very likely to occur due to precedence constraints. Although, as discussed, the actual estimated value is not really of concern, it is the relative *ranking* that must be good. Option 2 is likely to yield fewer or smaller gaps between tasks as a consecutive task within a job has a much smaller probability of requiring a delay to precedence constraints. However, this is much more dependent on the tasks that are already processed so far and may therefore provide a less consistent estimated makespan.

For either option, an additional choice can be made: to select one allocation for each task at random, instead of allocating on all possible machines. This introduces some variance or uncertainty, but is more likely to yield a realistic estimate of the real makespan.

In conclusion, we propose three ranking functions for a total of five implementations:

1. Lowest current makespan.

2. Sort by jobs.

   2a. Allocate each task on all machines.
   2b. Allocate each task on one random machine.
3. Sorted by tasks.
   3a. Allocate each task on all machines.
   3b. Allocate each task on one random machine.

Note that regardless of the ranking function, a restricted DD will only ever provide an upper bound on the makespan. As the diagram is incomplete, it cannot provide an actual lower bound on the solution.

5.7. Relaxing the FJS model

To get a measure of the quality of the upper bound of a restricted diagram and possibly find better solutions, a relaxation can be applied. Because a relaxed diagram includes a superset of all feasible solutions, the shortest path through a decision diagram provides a lower bound on the optimal solution. These bounds are calculated using a branch-and-bound scheme (Bergman et al. 2016).

The relaxation depends on the merge operator. This merge operator takes any number of states and returns a new relaxed 'superstate'. A relaxed state differs from an exact state in both the precedence constraints and the timing constraints.

To track precedence constraints in this relaxed state, Cire and van Hoeve extend the state S with an additional vector U. We once again follow the naming scheme from van den Bogaerdt (2018). As we will show below, the set V from now on contains the nodes that are shared between all merged paths from the root r to a node v (the intersection of all paths), while the set U contains all nodes on the merged paths from r to v (the union of all paths). For an 'exact' node that has no merged nodes on the path from r to v, both V and U are equal and contain the set of decisions on the path. Once two nodes are merged, V and U are the intersection and union respectively of the paths leading up to the merged node. As a result, we now track which tasks are certainly done, which tasks may still need to be processed on some paths, and which tasks cannot be scheduled from any path due to precedence constraints. This as a best-effort attempt at trying to make complete solutions by preventing schedules that are definitely infeasible (by breaking precedence constraints) or surely suboptimal (by scheduling tasks twice).

To be able to provide a lower bound on the solution, the relaxed node takes the pairwise minima of all F-vectors of the states that must be merged; the same applies to the T-vectors holding the completion time of the last task for each job. As an example, merging the vectors $F = (0, 5)$ and $\dot{F} = (4, 0)$ results in $F' = (0, 0)$.

An important detail must be tackled for the ranking strategy that is based on the current makespan. One cannot simply rank merged states based on the maximum of the F-vector, as the merge of two dissimilar states is likely to result in a very low makespan according to F. Looking at the previous example for F', this merged state would rank as (one of) the best state(s), while merged states are not ideal for generating good solutions due to the infeasibilities they introduce. Therefore, we want not only to track this 'minimized' F, but also the upper bound on the machine times, F^{ub}. We also introduce the vector T^{ub} which corresponds to the upper bound version of T: the finish times of the last task of each job. When a task is selected, T^{ub} and F^{ub} are updated similarly to F, but referencing the new vectors:

$$\overline{F_i^{\text{ub}}} = \begin{cases} \max(F_w^{\text{ub}}, T_u^{\text{ub}}) + p_{uvw}, & \text{if } i = w. \\ F_i^{\text{ub}}, & \text{otherwise.} \end{cases} \tag{5.4}$$

$$\overline{T_i^{\text{ub}}} = \begin{cases} \max(F_w^{\text{ub}}, T_u^{\text{ub}}) + p_{uvw}, & \text{if } i = u. \\ T_i^{\text{ub}}, & \text{otherwise.} \end{cases} \tag{5.5}$$

Then, when merging, the pairwise maximum of the machine times is taken for F^{ub}, and the makespan ranking is calculated by taking the maximum of F^{ub}.

In conclusion, we use the merge operator $\otimes$ that merges two states as follows: given two states $S(V, U, F, F^{\text{ub}}, T, T^{\text{ub}})$ and $\dot{S}(\dot{V}, \dot{U}, \dot{F}, \dot{F}^{\text{ub}}, \dot{T}, \dot{T}^{\text{ub}})$,

$$S \otimes \dot{S} = \left(V \cap \dot{V}, U \cup \dot{U}, \min(F, \dot{F}), \max(F^{\text{ub}}, \dot{F}^{\text{ub}}), \min(T, \dot{T}), \max(T^{\text{ub}}, \dot{T}^{\text{ub}})\right) \tag{5.6}$$

The functions $\min(A, B)$ and $\max(A, B)$ take the pairwise minima and maxima of the values in A and B. van den Bogaerdt (2018) established and proved correctness for this merge operator: a valid merge operator should result in a relaxed diagram that contains a superset of all solutions.

5.8. Relaxation in DDO

DDO offers different ways of constructing a decision diagram during the relaxation phase. There are two types regarding node caching: caching and no caching. As far as we are aware, the caching variant includes an additional datastructure that allows some heuristics to be added instead of some default optimizations. This hash-set prevents DDO from doing redundant work at the cost of additional memory. This is discussed in more detail by Coppé et al. (2024). We did not implement any of these caching heuristics. Then, there are three different modes regarding the branch-and-bound algorithm, some of which suggested by Bergman et al. (2016). DDO offers a *frontier-* (*Fc*), *last-exact-layer-* (*Lel*) and *Pooled* cutset for the branch-and-bound algorithm. The pooled cutset combines the frontier and lel-cutset. Combining these types with caching and no-caching, DDO offers six construction types.

During the relaxation phase, DDO may (partially) explore subtrees and expand promising nodes (Coppé et al. 2024) to find improved bounds. The nodes that

Therefore, it neither does breadth-first search nor depth-first search, but best-first search with some branch-and-bound technique. And, as mentioned before, if the relaxation phase ends prematurely due to a time limit cut-off, the relaxation cannot provide an actual lower bound because not all subtrees may have been considered.

5.9. Sequence-Dependent Setup Times — $\beta = s_{jk}$

Implementing Sequence-Dependent Setup Times in an MDD is rather straightforward. The state must be extended by one additional vector L - this L contains the last task that was processed on each machine. The transition function is modified to include the SDST between the previous and current task on this machine. Thus, the transition for the vectors F and T is defined as: $\overline{F_w} = \max(F_w + s(L_w, O_{uv}), T_u) + p_{uvw}$ and $\overline{T_u} = \max(F_w + s(L_w, O_{uv}), T_u) + p_{uvw}$. Here, $s(L_w, O_{uv})$ is the setup time between the last task on machine w and the upcoming task O_{uv}. Then, the vector L is updated: $\overline{L} = (L_1, ..., L_w = O_{uv}, ..., L_m)$.

5.10. Blocking tasks — $\beta = block$

To add blocking tasks, two modifications are required. The first modification concerns the completion time of tasks: a preceding task must remain on its machine until the subsequent task starts. This is a simple change as long as the previous task is currently the last task on its machine: we can just modify its completion time and set it to the start time of the subsequent task. However, if there is already another task scheduled after this previous task, we encounter some trouble. In the best case, we would need to shift the start and/or end time of certain tasks; however, in the worst case, there is an infeasible situation. Consider the example in Figure 5.1: the task $(1, 2)$ is under consideration to be scheduled. However, there are already two tasks behind its predecessor $(1, 1)$, and they cannot be shifted to resolve the problem. The only option is to switch the order of tasks, but that is non-trivial and goes against the incremental construction of Decision Diagrams. And besides, the diagram will already contain those paths in another branch.

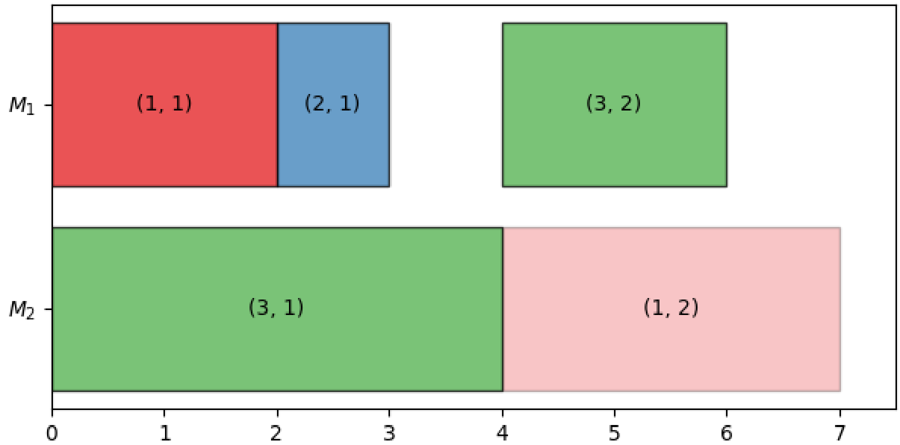

Figure 5.1.: Infeasible schedule for a shop with blocking tasks.

We conclude that if the predecessor of a considered task is not the last task on its machine, this task cannot safely be scheduled. Thus, another modification is required: the decision function must not schedule a task on a machine if the successor of the last task on that machine is not scheduled yet. This will prevent the situation in Figure 5.1 from happening. Unfortunately, this will introduce some dead paths, e.g. if there are two tasks remaining that must both be scheduled on the machine of the other task's predecessor. This is a case for which a solution exists, but this is not handled with our modifications. The feasible paths will still be in the exact decision diagram; however, a restricted diagram may not always be able to construct a solution.

5.11. Alternative Process Plans — $\alpha = FPFJ$

Alternative Process Plans are not as easy to implement as SDST. While it is conceptually easy to draw a diagram that includes APP, it is much more difficult to capture them in a (DDO) model. Creating an exact diagram requires only minor modifications:

- MDDs typically expect any arc to connect two nodes on consecutive layers. However, if one process plan contains fewer tasks than another, long arcs must be created that span to the end of the longest process plan. DDO provides an interface that does not actually create such a long arc, but creates as many copies as the long arc would span a number of layers; each short arc between these copies has an empty decision with zero cost.

- Jobs that include choices (APPs) at multiple points, either nested or consecutively, need an additional data structure to track the next task given the currently selected process plan.

With these modifications, both an exact and a restricted diagram can be built.

The main roadblock is encountered in a relaxed diagram however. Consider a job that has two process plans $A = \{A_1, A_2\}$ and $B = \{B_1, B_2\}$. For simplicity, this is the only job. Consider $S = (V, U, F, T)$ with $V = U = A_1$ and $\dot{S} = (\dot{V}, \dot{U}, \dot{F}, \dot{T})$ with $\dot{V} = \dot{U} = B_1$. Here we omit F^{ub} and T^{ub} for simplicity. How would we define $S \otimes \dot{S}$? We cannot simply take $V \cap \dot{V}$ nor $U \cup \dot{U}$. Instead, at least U and for best results also V should itself be a vector to track the progress for both process plans - in this case $U \otimes^U \dot{U} = (A_1, B_1)$ and $V \otimes^V \dot{V} = (A_1, B_1)$ as well. Here, $\otimes^U$ and $\otimes^V$ are the 'partial' merge operators as defined for vectors U and V. This strategy could work for an instance where the number and structure of process plans is identical for all jobs. However, if some jobs have nested process plans, let alone multiple layers of nesting, the datatype of V and U must be flexible and becomes intractable. There is no decently fast programming language that allows this; the Rust language in which DDO is written does not permit this either.

This problem can be tackled by enumerating all process plans beforehand into all unique paths for this job. Then, the decision function must be modified to only select one of the enumerated paths for each job, as a solution should only include one process plan for each job. Considering the previous example with a job with two process plans $A = \{A_1, A_2\}$ and $B = \{B_1, B_2\}$, this now effectively turns into an instance with two jobs A and B. When constructing the MDD, the decision function should return both A_1 and B_1 as an allowed decision for the root state. But, once a certain process plan is selected, the decision function should only continue that process plan. For instance, given a vector $V = (A_1, \square)$, it should only return A_2 as an option, and for $V = (\square, B_1)$ it should only return B_2 as an option. This ensures only one process plan is present in a solution. Regarding the merge operator, the merge operator from the original relaxation can be left intact, with $V \otimes^V \dot{V} = V \cap \dot{V}$ and $U \otimes^U \dot{U} = U \cup \dot{U}$. The decision function subsequently should continue all process plans that are in progress given U. For instance, $(A_1, \square) \otimes^U (\square, B_1) = (A_1, B_1)$, and the decision function subsequently should yield all machine options corresponding to both A_2 and B_2.

Sadly, this formulation does not support the And-nodes as discussed for the And/Or-networks, for the same reason of the nested paths. This is not as easily solved using enumeration, as the paths would need to be coupled. Our formulation only supports Or-nodes. Therefore, our model does not support $\alpha = FPFSFJ$, but $\alpha = FPFJ$ instead (see Section 2.6).

5.12. Warmstart

DDO allows setting an initial (primal) solution. This solution simply acts as a restricted diagram of $w = 1$, which can act as a starting point for relaxation.

5.13. Summary

In this chapter, we have discussed the MDD model and how we implement the FJSP and its extensions in DDO. While these extensions are handled separately from each other, they can all be combined without problems to create a $FPFJ|prec, s_{ijk}, block|C_{max}$ model. Working with MDDs is very different from CP: instead of specifying a large set of constraints and leaving out the way to get there, for DDO we fully specified how to create a solution and what the best solution would be, but with much less administration regarding precedence and timing. This is an advantage of the incremental nature of Decision Diagrams, as they are built layer by layer, without revisiting earlier decisions. Unfortunately, we were unable to match the APP formulation from CP in our MDD model, ending up with a less general formulation.

6 Computational results

In this chapter, we first select applicable datasets for the general FJSP (Section 6.1.1), SDST (Section 6.1.2), Blocking (Section 6.1.3) and APP (Section 6.1.4). Then, we benchmark these using both CP models in Section 6.2. Finally, we use these datasets to benchmark our MDD model in Section 6.3, including comparisons between the two approaches.

6.1. Benchmark instances

We start with datasets for the plain FJSP model. With a baseline set for these datasets, we collect additional instances for the SDST and APP extensions. All datasets used here are available from our repository[1].

6.1.1. FJSP

There is a substantial number of datasets in the FJSP literature, comprising hundreds of instances in total. After a thorough investigation through older and more recent papers, we came to the following datasets:

Brandimarte The oldest dataset stems from Brandimarte (1993), and comprises 10 instances from small to medium size. Two of its largest instances are not known to be solved to optimality, the others are.

Dauzere-Paulli The second dataset comes from Dauzère-Pérès and Paulli (1994): it consists of mostly medium-size instances, but is mostly known for its high number of operations per job.

Hurink The dataset with the highest number of instances is from Hurink et al. (1994): it consists of three sets of each 66 instances with increasing operation flexibility. This is outlined in Table 6.1 for the three datasets *edata*, *rdata* and *vdata*.

Barnes Barnes and Chambers (1995) created a dataset of small to medium sized instances, most with a very low operation flexibility. As a result, all its instances have known optimal solutions.

Kacem Another dataset has been created by Kacem et al. (2002): it has only four small to medium-size instances, all with complete operation flexibility (meaning that all operations can be processed on all machines), but a low number of operations per job, and as a result all optimal solutions are known.

Fattahi Fattahi et al. (2007) created a dataset of small and medium instances and relatively low flexibility with few operations per job. All instances have known optimal solutions. The small instances are so small that we only included the 10 medium instances.

Behnke As many datasets have known optimal solutions, Behnke and Geiger (2012) decided to create a dataset with medium to large instances. With on average fewer operations per job than others, some instances have known optimal solutions, while for a large number of them, only bounds are known for feasible solutions.

[1] `https://github.com/StevenCellist/FJSP-with-APP-using-CP-and-MDD-thesis`

Naderi The most recent dataset has been created by Naderi and Roshanaei (2021): they are all large instances with a high operation flexibility, many operations per job and many jobs. Consequently, only one instance has a known optimal solution, and for all the others, only feasible solutions with bounds are known.

These datasets vary in several aspects, such as (a) the number of instances, (b) instance size (defined by the number of jobs and machines), and (c) the average number of operations per job. Moreover, as discussed in Chapter 2, two crucial aspects of the FJSP are (d) the average number of eligible machines per operation -as an absolute value- and (e) the average flexibility rate (calculated as the ratio of eligible machines per operation to the total number of machines) - a relative value. For instance, in the dataset proposed by Brandimarte, the flexibility rate is 0.36, meaning that, on average, 36% of available machines (or in this case 2.5 machines) can process a given operation.

For a quick overview, these characteristics are outlined in Table 6.1 for the datasets discussed.

Table 6.1.: Public benchmark instances for the FJSP.

Dataset	Instances	Jobs	Machines	Operations	Machs/ops	Flexibility
Barnes	21	10-15	11-18	11.9	1.2	0.09
Brandimarte	10	10-20	4-15	9.1	2.5	0.36
Dauzere-Paulli	18	10-20	5-10	19.5	2.6	0.33
Fattahi	20	2-12	2-8	3.3	2.3	0.52
Hurink-edata	66	6-30	4-15	9.0	1.1	0.15
Hurink-rdata	66	6-30	4-15	9.0	2.0	0.26
Hurink-vdata	66	6-30	4-15	9.0	4.7	0.48
Kacem	4	4-15	5-10	3.3	8.8	1.0
Behnke	60	10-100	20-60	5.0	12.4	0.32
Naderi	96	30-100	10-20	10.7	6.0	0.40

The first six datasets (Brandimarte, Barnes, Dauzère-Paulli, Fattahi, Hurink and Kacem) have been benchmarked numerous times. These are sometimes referred to as the *classic* datasets throughout this chapter, consisting of mostly small- and medium-sized instances. Some were solved using general models, with other researches using more tailored models. As a result, of the 271 instances in this 'classic' benchmark suite, 244 have been solved to optimality. The other two datasets (Behnke and Naderi) are newer and larger, and commonly referred to as the *modern* datasets. There is no common maximum runtime between different benchmark reports, with runtimes varying from ten minutes to two hours on one or multiple processing cores.

Results with bounds are scattered over numerous reports and papers. A thorough although non-exhaustive search was performed, and we found the best results in the following papers for the six classic datasets: Naderi and Roshanaei (2021) and Dauzère-Pérès, Ding, et al. (2024). For the Behnke dataset, results were found in the original paper by Behnke and Geiger (2012), as well as papers from Lei et al. (2022) and Wan et al. (2024). For the Naderi dataset, upper bounds were made available by Lan and Berkhout (2025). All bounds can be found in Appendix B.

There exist multiple repositories each with a number of these datasets, with the complete one we used collected by Lan and Berkhout (2025).

6.1.2. SDST

The FJSP with Sequence Dependent Setup Times is a combination that has not seen much research. To the best of our knowledge, there are no publicly available benchmarks that are also reviewed in literature. However, there are some papers that refer to the dataset *SDST-HUdata*. This dataset is an extension of the dataset by Hurink et al. with SDST, as proposed by Oddi et al. (2011). It is benchmarked in a handful of papers: Fernández et al. (2013), Azzouz, Ennigrou, et al. (2016), Azzouz, Ennigrou, et al. (2017), Azzouz, Chaabani, et al. (2020) and Ben Ali et al. (2024). Upon request, Azzouz provided the instances *la21* through *la40* of the Hurink-edata set that they used for their results, although we expect that they are different from those used in the paper by Oddi et al. (2011). The values of the SDST are roughly 20% of the average processing times of the tasks.

Besides this dataset from Oddi et al. (2011), a dataset with 20 instances from Fattahi et al. (2007) was found. This dataset is identical to Fattahi's original dataset with 10 small and 10 medium instances, and expanded by the company Hexaly[2] to include SDST. The values of the SDST are roughly 50% of the average processing times of the tasks. Only one paper was found that mentions this dataset (Reijnen et al. 2025); the author kindly provided us with their results upon request.

Table 6.2.: Public benchmark instances for the FJSP-SDST.

Dataset	Instances	Jobs	Machines	Operations	Machs/ops	Flexibility
SDST-HUdata	20	10-20	5-10	5.9	1.1	0.20
Fattahi-SDST	20	2-12	2-8	3.3	2.3	0.52

A short overview of both datasets is given in Table 6.2.

Some authors have generated custom SDST instances which are not publicly available, and with various levels of detail on how they were generated (Özgüven et al. 2012), (Rossi 2014), (Tayebi-Araghi et al. 2014), (Shen et al. 2018). We reached out to the authors, but unfortunately did not get any response. Thus, we cannot use their results.

6.1.3. Blocking

For the blocking FJSP, we did not find any public mentions of datasets. However, we decided to keep it simple: we take the known FJSP datasets as discussed in Section 6.1.1, and mark all tasks as blocking.

6.1.4. APP

Similarly to the FJSP-SDST problem, the FJSP with Alternative Process Plans also is a seldomly researched category. Currently, there is no record of any dataset that can be used to benchmark this scenario. Just one paper by Kis (2003) was found that benchmarked their algorithm on a randomly generated dataset, using the same And/Or-graphs. They give a description on how their instances were generated, but do not provide their instances online.

As the APPs are a core part of our research, we took the description from Kis and generated our own instances from that. As Kis phrased it: *"Each job was a sequence of 1, 2 or 3 or-subgraph(s), and each or-subgraph had two or three branches, where a branch consisted of either one and-subgraph of five operations, or a sequence of two and-subgraphs one of them comprising 2, the other 3 operations. The processing times and the machine requirements of the operations were generated at random with the restriction that the five operations on the same branch of an or-subgraph always required five different machines. The processing time of the operations varied between 2 and 99."* (Kis 2003)

[2] `https://www.hexaly.com/example/flexible-job-shop-problem-with-setup-times-fjsp-sdst`

We made one modification to this description: instead of one machine per operation, an operation may be performed by more than one machine, which re-introduces the operation flexibility that is used in the FJSP.

Table 6.3.: New benchmark instances for the FJSP-APP setting.

Instances	Machines	Jobs	Or-nodes	Machs/ops
10	5	10	1	1
10	5	10	2	1
10	5	10	3	1
10	5	10	1	2
10	5	10	2	2
10	5	10	3	2
10	5	5	1	1
10	5	10	1	1
10	5	15	1	1
10	5	20	1	1
10	10	5	1	1
10	10	10	1	1
10	10	15	1	1
10	10	20	1	1
10	10	10	1	2
10	10	10	2	2
10	10	10	3	2
10	10	30	1	2
10	10	30	2	2
10	10	30	3	2
10	10	50	1	2
10	10	50	2	2
10	10	50	3	2

The parameters for the 230 instances we generated are listed in Table 6.3.

As our MDD model does not support And-nodes, the instances from Kis cannot be used for this model. Subsequently, we will need to generate instances that do not use these parallel paths. We took existing instances and added APPs to them. Given a job (consisting of a set of tasks), we define a number of process plans for this job. A process plan is defined by indices corresponding to the task-list of that job. Originally, the first job of instance `la01` from Hurink's edata looks like this:

```
5    1 2 21    1 1 53    1 5 95    1 4 55    2 3 34 5 34
```

This job consists of 5 tasks; the first four can be processed by one machine, while the last one can be processed by two.

To include APPs, we transform it like so:

```
3    5    1 2 21    1 1 53    1 5 95    1 4 55    2 3 34 5 34
3    1 3 5
3    1 2 3
5    1 2 3 4 5
```

We prepend the number of APPs to the job line (in this case 3); after the job line, as many lines follow with an APP. The first APP in this example consists of tasks 1, 3 and 5; the second of tasks 1, 2 and 3, while the third APP consists of tasks 1 through 5. Note that this format is a full enumeration, while a DAG could be constructed that resembles the And/Or-network formulation shown before. Such a DAG formulation is harder to capture in an instance file however.

We generated APPs for 271 instances using the following settings: for each job, generate randomly between 1 and 3 APPs, where each APP consists of a random selection of between 30% and 90% of all tasks in the job. This means that on average, a job can be made according to 2 process plans, with on average 60% of the original tasks. The average length of the shortest of these two process plans is 50%. Accordingly, we expect an average decrease of 50% in makespan.

6.2. Constraint Programming

We tested both Google's OR-Tools CP-SAT[3] and IBM's ILOG CP Optimizer[4] on a range of benchmark instances. The three datasets (FJSP, FJSP-SDST and FJSP-APP) were benchmarked using 8 threads, for a duration of 900 seconds (unless mentioned otherwise), on a pair of Intel Xeon Gold 6134 CPUs @ 3.20GHz with 256GB of memory. The benchmarks in this section serve to verify the performance of the models and act as a baseline to compare with the MDD model. All programs used here are available from our repository[5].

Table 6.4.: Summary of CP Optimizer results for the FJSP datasets

Dataset	Instances	Optimal #	Avg Time (s)	Avg Gap%
Brandimarte	10	7	330.08	5.68
Dauzere	18	2	802.01	24.16
Hurink-e	66	58	121.31	0.67
Hurink-r	66	30	521.17	16.34
Hurink-v	66	27	531.92	21.32
Barnes	21	21	14.87	0.00
Kacem	5	5	4.09	0.00
Fattahi	10	10	19.38	0.00
Behnke	60	15	677.02	44.22
Naderi	96	3	874.57	55.80

FJSP Full results for this benchmark can be found in Appendix B. Table 6.4 shows a comprehensive overview per dataset for CP Optimizer, with Table 6.5 showing results for OR-Tools. In these tables, we report the number of instances in each dataset, the number of instances that are solved to optimality, the average runtime of all instances in that dataset, and the average gap (LB divided by UB) across these instances. Table 6.6 combines these all into a quick overview for the classic and modern datasets.

Regarding the classic benchmarks (see Section 6.1.1), literature reveals optimal bounds for 244 out of the 271 instances. Using our CP model for CP Optimizer, we solved 170 instances to optimality, with another 33 instances finding an optimal upper bound but the solver not yet able

[3]`https://developers.google.com/optimization/cp/cp_solver`, version 9.12.4544
[4]`https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-cp-optimizer`, version 22.1.2.0
[5]`https://github.com/StevenCellist/FJSP-with-APP-using-CP-and-MDD-thesis`

Table 6.5.: Summary of OR-Tools results for the FJSP datasets

Dataset	Instances	Optimal #	Avg Time (s)	Avg Gap%
Brandimarte	10	5	451.32	4.27
Dauzere	18	2	805.50	24.78
Hurink-e	66	59	71.88	0.89
Hurink-r	66	33	508.00	11.17
Hurink-v	66	27	537.88	18.74
Barnes	21	21	6.83	0.00
Kacem	5	5	8.43	0.00
Fattahi	10	10	75.26	0.00
Behnke	60	25	583.85	34.36
Naderi	96	1	892.12	56.66

Table 6.6.: Average bounds for CP solvers.

Solver	LB	UB	Avg. Time (s)
Classic datasets			
CP Optimizer	1173.8	1362.0	27.9
OR-Tools	1185.2	1374.6	28.7
Modern datasets			
CP Optimizer	558.6	1628.1	798.6
OR-Tools	563.0	1720.2	773.6

to close the gap within the given runtime. Using our CP model for OR-Tools, we optimally solved 172 instances, with optimal upper bounds for 37 more instances. In the detailed results (see Appendix B), bounds are included from Naderi and Roshanaei (2021). They used a very similar CP model and ran their experiments on a CPU with half the number of threads and slightly slower speed, but with a time-limit of 7200 seconds (versus our 900 seconds). As such, they managed to solve more instances as the additional running time allows more gaps to be closed. However, with roughly a quarter of the available total time, our results are very competitive, setting a positive baseline for further results.

For the modern datasets (see Section 6.1.1), optimal solutions were found for 18 instances using CP Optimizer and 26 for OR-Tools out of the 156 instances, while we found optimal bounds for only one instance in literature. An interesting observation is that CP Optimizer yields slightly better results on the Naderi dataset, while OR-Tools performed much better on the Behnke dataset, solving almost all medium-size instances. Here, CP shows its strengths by finding much better bounds across the board than were found in literature. The used CP models scale better than the Linear Programming or Machine Learning models that were used for the previously best-known bounds.

From Table 6.6, we conclude that both solvers (CP Optimizer and OR-Tools) are competitive on the classic datasets. Google's OR-Tools pulls ahead with a slight edge on average here, but either can be used just as well. Regarding the modern datasets with large instances, CP Optimizer clearly finds better upper bounds (-5.5%), while OR-Tools finds slightly better lower bounds (+0.8%). But by all means, the OR-Tools solutions are good as well.

Table 6.7.: Summary of CP Optimizer results for the FJSP-SDST datasets

Dataset	SDST	Instances	Optimal #	Avg Time	Avg Gap%
Fattahi	N	20	20	10.11	0.00
HUdata	N	20	20	0.47	0.00
Fattahi	Y	20	19	51.90	0.74
HUdata-SDST	Y	20	11	431.90	5.51

Table 6.8.: Summary of OR-Tools results for the FJSP-SDST datasets

Dataset	SDST	Instances	Optimal #	Avg Time	Avg Gap%
Fattahi	N	20	20	29.35	0.00
HUdata	N	20	20	0.21	0.00
Fattahi	Y	20	18	93.15	1.45
HUdata	Y	20	10	517.91	6.44

SDST In Table 6.7 and Table 6.8, we report our findings for the SDST datasets. We first benchmarked these instances excluding setup times (the first two entries), and including setup times after (the last two entries).

For the SDST datasets, optimal solutions were found for 28 of the 40 available instances using OR-Tools, with CP Optimizer solving the same 28 plus another 2 instances to optimality for a total of 30.

In contrast to the plain FJSP benchmark, the SDST benchmark shows a measurable disparity between CP Optimizer and OR-Tools, with better results for the former. It appears that OR-Tools has more difficulty with sequencing when SDST are present. This is strengthened by the observation that the computation time drops back to a similar level when the instances are parsed as a normal FJSP instance, ignoring the SDST. Inspecting individual instances (see Table B.4), CP Optimizer is able to solve instances more than five times faster than OR-Tools.

Comparing the runtime between the 'plain' instances (without SDST) and the instances including SDST, there is a notable difference. What we see is that the increase in complexity for the Fattahi instances is significant but not extraordinary; the difference for the HUdata instances is much worse however. This is due to the relatively higher values of SDST for the HUdata set: artificially increasing the values of the Fattahi from on average 20% to 50% of the processing time instances confirms this.

During testing, we also examined the effect of applying partial SDST (for instance on 50% of the tasks). For CP Optimizer, we observed linear scaling between the number of tasks with a setup time and the overall runtime. For OR-Tools, this was much worse: it looks like OR-Tools activates setup times almost globally if only a few actual values are specified.

Blocking For the Blocking FJSP, we tested the classic set of 271 instances using a runtime of 60 seconds. Average results are listed in Table 6.9, and full results are available in Table B.5. Of course, the resulting upper bounds are higher, since tasks are likely to have a larger duration as they usually remain longer on a machine. However, the lower bound hardly increased for the blocking instances, showing that they are much harder to solve. This is also clear from the increased runtime; looking for instance at the instance `mt10xyz` from Barnes' dataset, the runtime to prove optimality increased from 1.48 to 27.3 seconds for CP Optimizer, and from 0.70 to 57.6 seconds for OR-Tools. OR-Tools even failed to find feasible solutions for six instances,

Table 6.9.: Summary of results for the Blocking FJSP benchmark.
Average bounds exclude instances without a feasible solution.

Solver	Blocking	Avg LB	Avg UB	Feasible #	Avg Time (s)
CP Optimizer	N	1173.8	1362.0	271	27.9
OR-Tools	N	1185.2	1374.6	271	28.7
CP Optimizer	Y	1183.9	1659.6	271	46.1
OR-Tools	Y	1210.8	1766.7	265	50.8

showing that the modification to a blocking shop is non-trivial.

APP For the APP instances, we only found optimal solutions for the smaller instances, as the additional tasks mean that these instances (judging by the number of selected tasks) are still substantial in size. As we have generated these ourselves, we do not have numbers available for comparison.

Table 6.10.: Summary of results for the APP dataset

Dataset	Instances	Optimal #	Avg Time	Avg Gap%
AFJSP (CP Optimizer)	290	186	350.64	26.30
AFJSP (OR-Tools)	290	184	341.63	16.95

The first thing to mention is that the results from CP Optimizer versus OR-Tools are almost exactly equal, with CP Optimizer solving 2 more instances to optimality, but OR-Tools closing the gap between upper and lower bound substantially better.

Table 6.11.: Summary of OR-Tools results for AFJSP datasets

Dataset	Instances	Optimal #	Avg Time	Avg Gap
m05_j10_f1	30	20	345.74	11.42
m05_j10_f2	30	10	613.88	21.57
m05_or1_f1	40	40	8.78	0.00
m10_or1_f1	40	40	1.26	0.00
m10_or1_f2	30	10	600.31	32.85
m10_or2_f2	30	9	634.35	44.18
m10_or3_f2	30	4	822.82	51.09

Table 6.11 shows the results of the same dataset for OR-Tools, split out in a few different partitions, which gives insight into the effect of the number of machines, jobs and tasks. The abbreviation m refers to the number of machines, j the number of jobs, or the number of Or-nodes and f the average number of machines per task. Refer to Table 6.3 for more details.

The first pair shows the effect of having on average two machines per task instead of just one. Instead of 20 optimal solutions, just 10 are fully solved, and the average gap doubles. From the second pair (5 machines versus 10 machines), it is clear that an increase in machines is easier to solve; likely because there are fewer permutations to consider per machine. The third entry shows the differences between one, two or three consecutive Or-subgraphs in each job (resulting in more choices and thus process plans). The 10 smaller instances included in this dataset become very hard to solve when increasing the number of process plans, with the larger instances never yielding an optimal bound, resulting in a significant gap.

6.2.1. Progression of bounds

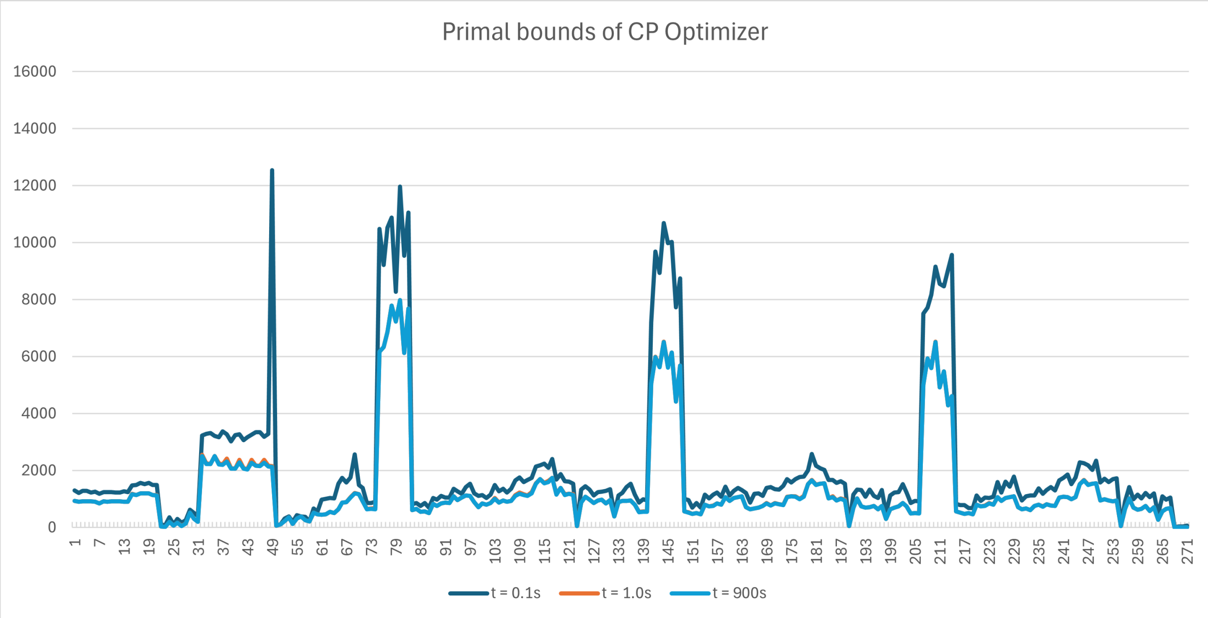

Figure 6.1.: Primal bounds of CP Optimizer over time.

We have analysed the progression of the bounds throughout the solving process of all FJSP instances. These are shown in Figure 6.1 for three different timestamps. From this, we see that the first solution of the solver is not very good (an average makespan of 2088.5), but it is quickly able to improve its solutions. The reported upper bound at the 1 second mark (on average 1366.0) is hardly improved up to the 900 second runtime cut-off (on average 1360.1).

6.2.2. Summary

Overall, we observe that CP solvers are very good at solving FJSP instances. But if the constraints become more complicated, especially regarding timing (through Blocking tasks or Setup times), their performance drops and the runtimes increase vastly. Mostly, they have trouble with finding better lower bounds to close the gap, but in some cases, even finding primal bounds is a challenge. Regarding the difference in both solvers, we see that both yield good results, but CP Optimizer generally finds marginally better upper bounds while OR-Tools usually yields slightly better lower bounds. Either way, the obtained results are very competitive with known results for the normal FJSP instances.

6.3. Multivalued Decision Diagrams

Our MDD model is benchmarked using DDO[6]. The same hardware is used as for the CP benchmarks, with again 8 threads, but now 60 seconds of runtime (compared to 900 for CP) unless specified otherwise. The main reason for this will become apparent later in this section.

[6] `https://github.com/xgillard/ddo`, version 2.0.0

Table 6.12.: Comparing bounds for five MDD construction heuristics on relaxed diagrams. *The lower-bound values as reported by DDO are incorrect due to the time-limit cutoff.*

Heuristic	LB (reported)	UB
1	938.2	1608.2
2a	954.5	1650.7
2b	926.8	1635.0
3a	931.3	1642.9
3b	954.7	1643.6

The results here will be compared to the results from the CP models. All programs used here are available from our repository[7].

6.3.1. FJSP

In this section, we first test some of the details of the framework and construction heuristics, in order to establish the best choices for our benchmarks.

Construction heuristics As the ranking function influences the bounds or solutions that can be obtained, we first present the results of comparing the five ranking strategies as discussed in Section 5.6. These heuristics are tested by generating relaxed decision diagrams for 271 instances (the classic benchmark sets), with a maximum width $w = 1$. We choose this width for the reason that it forces a maximal dependency on the ranking function in the restricted phase: there can only be one node on each layer, namely the 'best' one according to the ranking function. With a greater width, chances are higher that similar solutions are constructed, even if they are ranked in a different order, minimizing the effect of the ranking function. The node selection in the relaxation phase also depends on the ranking function: the better the ranking function, the better the relaxation if the relaxation does not complete within the specified time limit. For these reasons, we construct a relaxed decision diagram with a width $w = 1$, and we analyse which ranking function yields the best solutions.

The average primal and dual bounds obtained by DDO are listed in Table 6.12. DDO was able to build a relaxed diagram and therefore find bounds for all instances. However, an important observation is that DDO is unable to explore the complete fringe within the time limit. The reported lower bound then is only the lower bound with respect to the currently constructed diagram, while nodes that are still on the fringe may yield better solutions. Therefore, we cannot be certain about the reported lower bound values.

From Table 6.12, we see that there is very little difference between the upper bounds of the proposed ranking functions. The reported upper bounds are best for '1'. (which we recall from Section 5.6 simply ranks states according to the current makespan). From this, we conclude that the current makespan is a better indicator for a good solution than any of the estimation attempts, although the other heuristics do yield decent results. The reported lower bounds are best for '2a' and '3b'. However, as discussed, these values cannot be relied on. All further results in for MDDs will be generated using ranking function 1.

Construction modes As described in Section 5.6, DDO offers some different construction modes. We tested *NoCachingFc*, *CachingFc*, *CachingLel* and *CachingPooled* on a relaxed diagram with $w = 5$. The average results are available in Table 6.13. The caching modes show a clear

[7] `https://github.com/StevenCellist/FJSP-with-APP-using-CP-and-MDD-thesis`

Table 6.13.: Comparing bounds for four MDD construction modes on relaxed diagrams.
The lower-bound values as reported by DDO are incorrect due to the time-limit cutoff.

Method	LB (reported)	UB
`NoCachingFc`	937.2	1638.3
`CachingFc`	938.2	1608.2
`CachingLel`	937.1	1608.8
`CachingPooled`	939.8	1608.9

Table 6.14.: Comparing bounds for different numbers of threads.
The lower-bound values as reported by DDO are incorrect due to the time-limit cutoff.

Threads	Runtime cut-off (s)	LB (reported)	UB	Avg Time (s)
1	80	933.4	1623.7	60.8
2	40	931.5	1628.8	30.7
4	20	931.7	1633.3	15.5
8	10	930.0	1637.7	7.9

advantage with respect to the upper bound. We did not track RAM usage as a metric across the benchmarks, but a quick inspection every now and then did not reveal a large disparity in used memory. The additional memory used for caching is likely saved by fewer nodes being present on the fringe. Also, there is hardly any difference between the different modes that do include caching. Apparently, our model is not influenced by the distinctions between the cutsets.

Threads There are few reports of the effectiveness of multithreading for Multivalued Decision Diagrams. The authors of DDO did include an option for multithreading, which distributes the workload equally across the specified cores, and list some results using varying number of threads for a `MaxClique` problem (Gillard et al. 2020). To verify their results, we tested whether adding more threads is beneficial for constructing a relaxed MDD with $w = 1$, using different numbers of threads with an allowed total runtime of 80 seconds for all threads combined. Given perfect workload distribution, they should all yield near-identical results. Table 6.14 shows the used values for number of threads and maximum runtime, and their results. The results show that multithreading results in a slightly worse score, but the difference is small. Given a fixed runtime cut-off, adding more threads is surely beneficial.

Restriction — Width As DDO first constructs a restricted diagram, we investigate the effect of the width on the upper bound. Table 6.15 shows the results for $w \in \{1, 5, 10, 50, 100\}$. We observe that the average runtime increases roughly linearly with the width. This is as expected, as the number of nodes in the diagram increases linearly with w (until w exceeds the maximum width for a problem).

The construction of a restricted diagram is very fast for small- and medium-sized instances. Even a diagram with $w = 100$ is constructed in a few seconds for medium-sized instances, and for widths up to 10, the restricted diagram can be built in what we consider 'real-time': a fraction of a second. It does get progressively worse though for large instances: a width of 5 results in a runtime of 37.2 seconds for the largest instance (100 jobs, 10 machines, 30 tasks per job). Regardless, a restricted diagram is still generated within a very usable time, especially for $w = 1$.

An increase in width does yield a valuable improvement with respect to the primal bound;

Table 6.15.: Comparing primal bounds for restricted MDD width.

Width	UB	Reduction (%)	Avg. Time (s)	Max. Time (s)
Classic datasets				
1	1848.8	0	0.019	0.105
5	1726.5	6.4	0.066	0.470
10	1700.0	7.3	0.122	0.951
50	1631.6	11.7	0.680	6.78
100	1629.7	12.1	1.56	17.2
Modern datasets				
1	2080.4	0	0.851	6.78
5	2005.3	2.6	4.58	37.2

Table 6.16.: Comparing bounds for MDD construction.

Width	Restriction	Relaxation	Avg Time (s)	Improvement (%)
1	1848.8	1608.2	45.8	12.0
5	1726.5	1503.0	55.3	12.7

after $w = 50$ that we see only a little improvement. For the large instances in the modern datasets, the increase from $w = 1$ to $w = 5$ does not yield as much an improvement. The very large search space likely means that some additional width does not directly translate into much better solutions.

Relaxation — Limitations Moving on to the relaxation phase, we encounter a difficulty. While constructing the relaxed MDD, DDO consumes an enormous amount of memory. For larger instances, the number of nodes that could be created (given an unlimited width) on a layer grows immensely. As DDO tries to minimize the gap between the bounds, it keeps expanding these nodes, merging them with existing nodes and adding more and more arcs. As a result, large instances require a very large amount of memory, reaching a limit of 80GB after roughly 200 seconds (without a bound on the runtime). The program stalls once it reaches this 80GB limit and does not yield any results. With the cap on 60 seconds runtime in place, the relaxation fails to find any improvements over the restricted diagram, even for $w = 1$. Due to these problems, we did not include the Behnke and Naderi instances in the benchmarks unless specifically reported, and limited the runtime for DDO to 60 seconds.

Relaxation — Width Figure 6.2 shows the upper bounds that were achieved by building a relaxed MDD for $w = 1$ and $w = 5$. On average, a makespan of 1608.2 and 1503.0 was found respectively, with an average improvement of 7.1%. It is useful to note that the restricted MDD finished with a makespan of 1848.8 for $w = 1$ and 1724.6 for $w = 5$ - thus, the decrease on makespan after the relaxation phase is logically better as well. However, the absolute improvement for $w = 5$ is lower, as the optimal values are being approached.

Relaxation — Improving the Restriction Figure 6.3 shows the results of limiting DDO to just construct a restricted diagram, as well as the bounds when allowing relaxation to occur up to the 60 second cut-off. The averages are reported in Table 6.16. In most cases, the improvement

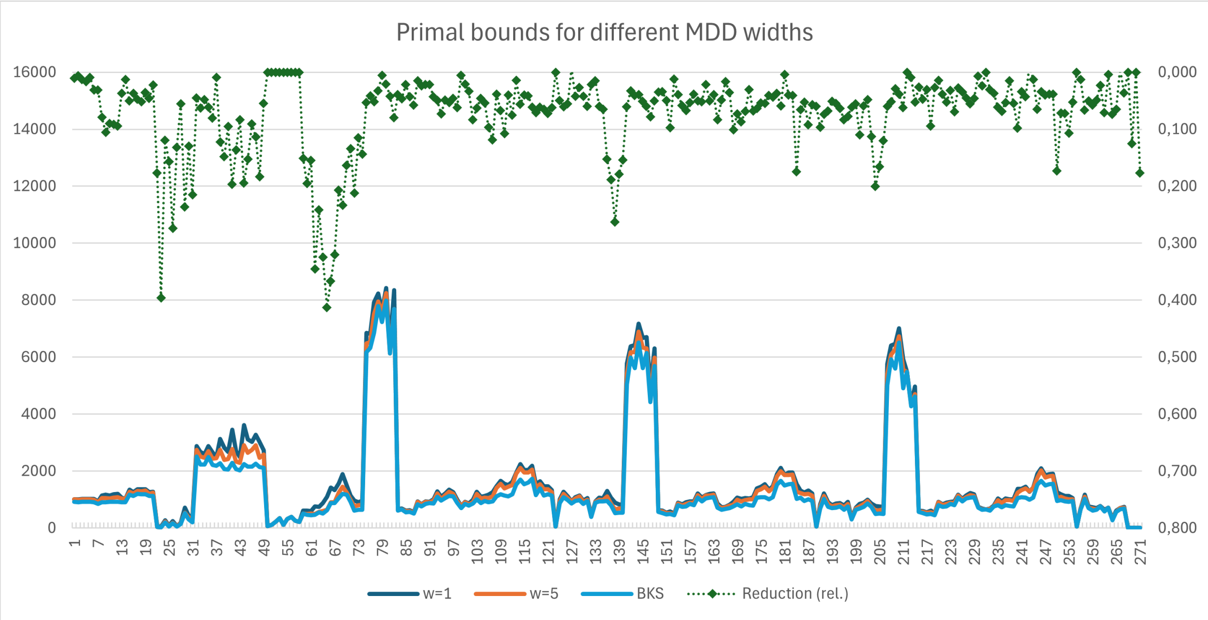

Figure 6.2.: Primal bounds for 271 FJSP benchmark instances, including Best-Known Solutions.

Table 6.17.: Average bounds for 40 SDST instances.
The lower-bound values as reported by DDO are incorrect due to the time-limit cutoff.

Solver	SDST	LB	UB	Time (s)
CP Optimizer	N	649.2	654.4	2.3
OR-Tools	N	641.9	654.6	4.4
DDO	N	386.7	702.1	39.5
CP Optimizer	Y	681.0	729.8	19.4
OR-Tools	Y	668.8	743.9	32.5
DDO Restricted	Y		762.9	0.13
DDO Relaxed	Y	419.5	709.5	35.9

of the relaxation is roughly 10-20%. However, as reported earlier, this improvement diminishes with the size of the instance. For the instances of the Behnke and Naderi datasets, the relaxation did not yield any improvements.

It is also worth noting that while relaxation for $w = 1$ yields an average makespan of 1608.2 in 45.8 seconds, increasing the restricted MDD from $w = 1$ to $w = 100$ yields an average makespan of 1629.7 in 1.56 seconds. The advantage of the relaxing a $w = 1$-diagram in DDO is that it works good as an anytime-algorithm, returning an initial solution almost instantaneously and continuously improving it; a restricted MDD only returns a solution once it completes the final layer. However, on average, a wider restricted MDD yields much better performance over runtime.

6.3.2. SDST

Table 6.17 shows the results for the 40 instances that include Sequence-Dependent Setup Times, compared to the same instances without, using CP Optimizer, OR-Tools and DDO ($w = 100$, as we want to achieve competitive results). The same observation is apparent as before that OR-Tools has more difficulty when SDST are included. Then, we see that the inclusion of SDST

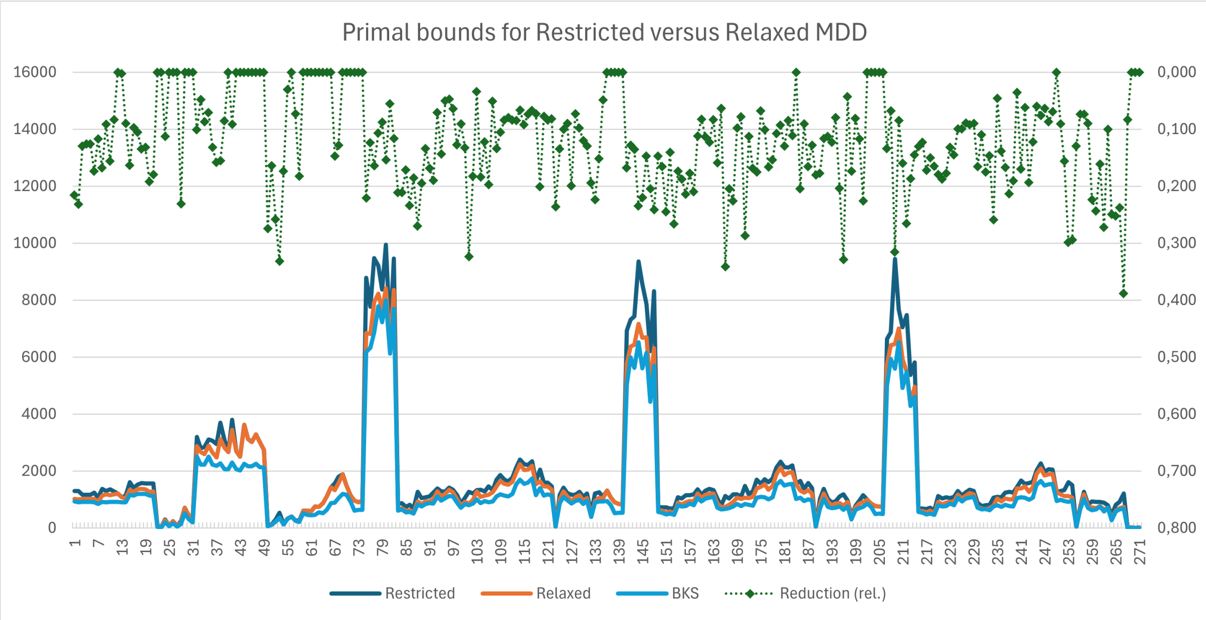

Figure 6.3.: Primal bounds for 271 FJSP benchmark instances, including Best-Known Solutions.

Table 6.18.: DDO results for the Blocking datasets.
The lower-bound values as reported by DDO are incorrect due to the time-limit cutoff.

Width	# Res. feasible	# Rel. feasible	Rel. LB	Rel. UB
1	29	103	1538.7	2900.5
5	37	139	1166.6	2543.1
10	43	153	1216.1	2512.5
50	84			
100	100			

has no effect on the runtime of the MDD algorithm versus without. Here, we even see DDO beating both CP solvers based on runtime and upper bound values! However, DDO is not able to provide a lower bound as good as the CP models. The advantage of the MDD model for SDST is that adding the SDST is just a constant-complexity operation. It does not result in any different or additional behaviour. This contrasts with our findings for the CP solvers, where the addition of SDST had a major impact on the performance.

6.3.3. Blocking

The DDO results for the Blocking datasets are not so strong, just as we saw that the CP solvers have trouble with Blocking tasks. This is clear from Table 6.18, where we list the number of solutions obtained for restricted and relaxed MDDs, as well as the average lower bound for the relaxed MDDs. Restricted diagrams do not yield any solution for a large number of instances. This is caused by the infeasibilities that can occur with these blocking tasks, as described in Section 5.10. Increasing the width does result in more feasible solutions: there is a higher chance that there is a path that does not end up in an infeasible situation. A relaxation is able to find a lower bound for all instances; however, as before, these may be incomplete. For the $w = 5$ and $w = 10$ relaxation, we do see lower bound values in the expected region. The upper bound values are high, but an increase in width helps a lot here. The relaxation also manages to find

Table 6.19.: Comparing bounds for APP instances.
The lower-bound values as reported by DDO are incorrect due to the time-limit cutoff.

Solver	LB (reported)	UB	Time(s)
	Without APP		
CP Optimizer	1173.8	1362.0	27.9
OR-Tools	1185.2	1374.6	28.7
DDO	938.2	1608.2	45.8
	With APP		
CP Optimizer	720.9	780.0	15.2
OR-Tools	755.9	781.7	11.9
DDO	160.7	965.0	57.2

many new solutions, as it is able to merge some dead paths into valid paths. Overall, a restricted solution would be much preferred, but our method is not good enough for this; the relaxation does help quite well.

6.3.4. APP

A quick check shows that the runtime of the APP formulation is on par with the FJSP model for normal instances. To verify this, we converted the FJSP instances into APP instances with one process plan, consisting of all tasks of the job. Except for small variances both up and down, the average results are the same.

We tested 271 instances as described in Section 6.1.4. Full results are listed in Table B.10, with averages displayed in Table 6.19. We observe an average decrease in makespan upper bound of 43% for CP Optimizer and OR-Tools, and a decrease of 40% for DDO, approaching the estimated 50% reduction as the process plans in these instances are shorter and there are more options (see Section 6.1.4). The average runtime of both CP solvers roughly halved, while almost all instances ran into the runtime limit for DDO. The addition of these process plans therefore does not result in a large difference between our CP and MDD model, although the CP models do achieve a better runtime. The increase in number of jobs has a very large impact on the number of available decisions per layer in the decision diagram, as we demonstrated in Section 5.5.

6.3.5. Summary

The results from DDO for our MDD model can be summarized as follows: Decision Diagrams can be used to very quickly generate feasible solutions, especially if the width is very small. The quality of the solution strongly depends on the decisions that are made, as a poor decision will result in a poor solution because the diagram is built incrementally; it does not revisit earlier decisions. MDDs are very useful for certain timing constraints because of this incremental construction: the SDST extension can be added without any decrease in performance, which as a result outperforms the CP solvers. Other extensions resulted in varying success: generally, feasible solutions are found, but CP outperforms our model in many situations.

Table 6.20.: Comparing CP cold-start versus a warm-start from MDD model.

Solver	LB	UB	Time (s)
CP Optimizer	1173.8	1362.0	27.9
DDO + CP Optimizer	1173.7	1362.1	27.8
OR-Tools	1185.2	1374.6	28.8
DDO + OR-Tools	1183.6	1376.6	28.7

6.4. Combining MDD and CP

Both CP Optimizer and OR-Tools support an optional warmstart. As DDO is able to quickly generate a feasible solution, we tested whether they are useful as a primal solution to the CP solver.

- Generate a restricted MDD with $w = 100$.
- Take the best solution from the MDD and convert it into a warmstart for a CP model.
- Run the CP model for 60 seconds.
- Compare against the same CP model without warmstart and 60 seconds of runtime.

Table 6.20 shows the results of CP only versus CP with a warmstart, for both CP-Optimizer and OR-Tools. We see virtually no difference in performance with or without this warmstart. This does align with the findings in Section 6.2.1. The story is likely somewhat better for SDST instances.

The opposite technique is also possible: DDO supports setting a 'primal' solution, similar to the warmstart for the CP models. We tried to apply a reverse of the warmstart technique to generate an initial solution with CP and construct an MDD from this primal solution. However, DDO was unable to yield any improvement in reported bounds over the CP models after the same 60 seconds.

7 | Discussion

In this research, we explored the Flexible Job Shop Problem (FJSP) with two extensions using two different techniques. First, we modelled and evaluated the FJSP using Constraint Programming (CP), a widely used paradigm for scheduling. We extended this model with Sequence-Dependent Setup Times (SDST) and Alternative Process Plans (APP). These results serve as a baseline for our next method and future research. Then, we evaluated the same problem and extensions using Multivalued Decision Diagrams (MDD). For the general FJSP, there are numerous datasets with public benchmark results; for the SDST and APP extensions, there are few or no public instances with no known bounds. Therefore, we crafted some datasets which are available online[1], and we report our best-found bounds for them in Appendix B.

7.1. Constraint Programming

For the CP model, we evaluated two constraint solvers: the commercial IBM CPLEX CP Optimizer solver and Google's open-source OR-Tools CP-SAT solver. We showed that our implementation yields very competitive results using either solver. In many circumstances, both solvers are competitive with minor differences every now and then. A slight edge goes to CP Optimizer for the upper bound values, while OR-Tools usually finds a (significantly) better lower bound. The biggest difference was observed when including Sequence-Dependent Setup Times, as it appears that OR-Tools has more trouble with sequencing than CP Optimizer. Its results are still decent and usable though.

Constraint Programming is a very flexible paradigm: adding extensions is easy and in most cases quite straightforward. However, the selection and implementation of constraints can greatly impact the performance, and a small modification of or extension to the basic FJSP model may already worsen the runtime significantly. Adding Sequence-Dependent Setup Times into the model, we observed a big step back in performance. CP Optimizer was better able to handle these setup times, with OR-Tools lagging behind. However, both models are still able to generate good solutions; mostly the gap to the dual bound is harder to close. Switching to a fully blocking shop, where tasks must remain on a machine until the next task starts processing, we see a strong degradation in performance. OR-Tools even fails to find feasible solutions within a smaller time limit.

Alternative Process Plans can be implemented into the CP models without any limitations. Some additional administration is required to capture all And/Or-precedence constraints, but all of it is possible. Once again, the additional complexity means that the reported gap between primal and dual bound increases, but the generated solutions are good for many instances.

Overall, Constraint Programming is proven to be a good framework for (Flexible Job Shop) scheduling, both in literature and in our results. For plain, and small- and medium-sized instances, it is well capable of finding great if not optimal bounds, but for larger instances or with extensions, it requires quite some time to yield good solutions and even more to close the gap between the upper and lower bound.

[1] `https://github.com/StevenCellist/FJSP-with-APP-using-CP-and-MDD-thesis`

7.2. Multivalued Decision Diagrams

Next, we looked at the same problems from the perspective of Decision Diagrams, using the Rust-implementation DDO. We concluded that exact MDDs are infeasibly large for any non-trivial FJSP instance. Thus, we applied a restriction and relaxation, with the consequence that it is difficult to find an optimal solution.

Using a restricted MDD, decent primal bounds were found in real-time for small- and medium-sized instances, and near-real-time for large instances. Increasing the maximum allowed width of the restricted diagram greatly helps finding better solutions, but this does not scale as well to larger instances.

When relaxing the restricted MDD, dual bounds were obtained for small- and medium-sized instances within a minute, but for large instances, DDO was unable to report any lower bound. Building a relaxed MDD can also improve the upper bound; however, we found that a wider restricted diagram may be a better idea than spending time relaxing a small restricted diagram.

Ideally, construction would allow incrementing the width of the restricted diagram as we saw in Table 6.15; starting with $w = 1$, increasing the width until a specified duration is exceeded (or some other form of cut-off). This way, even large instances could see useful improvements with longer runtime, where DDO now failed to relax these instances in a useful way. Only once an increase in width results in marginal improvements, we would recommend applying a relaxation.

We also compared heuristics to select the best nodes while constructing the MDDs, concluding that the current makespan of a partial schedule yields better results than a simple estimation of the total makespan. However, a more sophisticated estimation of the makespan may result in improved upper bounds.

Adding Sequence-Dependent Setup Times has virtually no effect on the runtime or performance of our model. This is a clear advantage of the incremental nature of Decision Diagrams: any decision that is made is final. The setup times are simply a shift in the start and end-time calculation (a constant-complexity operation), nothing more. Here, DDO manages to outperform the CP solvers, demonstrating the power of such an incremental construction. Regarding the Blocking instances, we demonstrated that MDDs can quickly yield good solutions for smaller instances. But as our decision function allowed traversal into dead ends, we were unable to find solutions to larger instances in the restricted phase. An improved decision function that would prevent the traversal of these paths would be a useful improvement.

We also came up with a formulation for Alternative Process Plans. This formulation is not as flexible as for CP, as it does not work with And-nodes; however, for the alternatives, we essentially only require Or-nodes. The results for DDO versus both CP solvers are comparative to plain FJSP instances, but as the search space is larger, the observed behaviour is magnified: DDO quickly yields a good solution from restricted diagrams, but in longer runs, the CP solvers outperform DDO easily.

Finally, we note that DDO consumes a rather large amount of RAM while relaxing large instances. While it is manageable for server equipment, it outgrows consumer-hardware during relaxation of larger instances.

7.3. MDD + CP

We also tried a warmstart technique for CP for the FJSP instances: giving the solution from a restricted MDD as an initial solution to a CP solver. This did not result in any performance improvements, as the CP solvers also manage to find a good solution in short time on their own. However, if improved solutions can be generated using restricted MDDs, this may result in a useful hybridization in the future.

7.4. Conclusion

In Chapter 1, we asked the question whether it is feasible *"to produce a schedule for a typical daily shift of a moderately sized shop subject to High-Mix Low-Volume: e.g. a set of up to 100 orders"*, with the additional question *"what is the effect of including setup times and alternative process planning into the scheduler on the quality and makespan of the produced schedule?"* We conclude that both Constraint Programming and Multivalued Decision Diagrams are a useful means for producing a schedule. MDDs can be used to very quickly generate a decent schedule, but (at least regarding the DDO framework we used) it is not very suited for getting (near)optimal solutions, or a useful estimate on the quality of the schedule. Some modifications (such as large processing times) and some extensions (such as SDST) lend themselves perfectly to solving using MDDs due to the incremental construction of the diagram, while others are harder to implement (such as APP) due to the fixed structure of these diagrams. CP is much more flexible in that regard, as it does not have such a rigid structure due to the freedom of the constraints. However, CP does suffer from modifications or extensions. While the tested CP solvers are capable of finding good bounds for a plain FJSP instance, it is not as fast when for instance SDST are involved, or when processing times get large. Given more runtime, they do manage to gradually improve their bounds.

If the runtime of schedule creation (approaching real-time) is of more importance than the quality of the produced schedule, MDDs appear to be a good tool. However, if there is some more time available (such as an operator starting up machines and getting a cup of coffee), CP is likely to yield better results.

7.5. Future work

The Flexible Job Shop field is quite mature, with an extensive track record of public research. Constraint Programming for (Flexible) Job Shop Scheduling has been around for long enough that we have not seen major improvements over the last few years. Improvements may be obtainable with newer versions of the CP solvers or even tighter constraint formulations. The latter may especially be the case for the Alternative Process Plans.

More interesting however is the development around using MDDs for scheduling. This application of MDDs might still be considered in its infancy given the limited amount of available research, even though the results we demonstrated are promising. The main improvement that we would like to try out is the use of an A^*(-like) algorithm (Hart et al. 1968) for MDD construction. With such an algorithm, the (restricted) Decision Diagram is not necessarily built layer by layer, but using a priority queue that works across all layers. Consequently, construction would be neither breadth-first nor depth-first, but a hybrid. Literature shows that such an anytime construction algorithm can be very useful (Fontaine et al. 2023, Horn et al. 2021). Another construction option would be some equivalent to beam search, where the width of the restricted diagram is gradually increased, serving as another useful anytime construction algorithm.

Besides a different construction technique, there is likely also room for improvement regarding the ranking functions. There may be better estimation functions that result in improved solutions for restricted diagrams. And not only the ranking function may be improved, the decision function for the Blocking case can likely be improved.

Lastly, our MDD formulation considers the full solution space, with all decisions being considered during construction, even though a restricted set may be selected. With improved dominance rules or heuristics, the solution space can maybe be pruned, improving the runtime or yielding better restricted or relaxed diagrams.

In conclusion, we challenge others to investigate the use of Multivalued Decision Diagrams for scheduling and improve our results.

A | CP model for OR-Tools

In Chapter 4, the Constraint Programming model is given for ILOG's CP Optimizer. Here, we describe the model used for Google's OR-Tools. It is similar in most regards, however, there are some differences, mostly regarding setup times.

The notation used is identical to that in Chapter 4, with some additions listed in Appendix A.

	Description
Variables	
$T = \{1, 2, ..., N\}$	The indices of the concatenation of all variables Task_{jkm}
m_t	The machine corresponding to task $t \in T$
D	A dummy binary variable
Functions	
`ExactlyOne(B)`	Creates a constraint such that exactly one of the boolean variables in B is selected
`If`(e, s)	If the expression e evaluates to 1, activate the statement s

A.1. Flexible Job Shop model

A plain FJSP model consists of the following goal and constraints[1]:

$$\text{minimize} \quad C_{\max} \tag{A.1}$$

$$\text{subject to} \quad \text{Task}^*_{jk} = \text{IntervalVar}([\min_{m \in \mathcal{M}_{jk}} p_{jkm}, \max_{m \in \mathcal{M}_{jk}} p_{jkm}]) \qquad \forall j \in \mathcal{J}, k \in \mathcal{O}_j \tag{A.2}$$

$$\text{Task}_{jkm} = \text{IntervalVar}(p_{jkm}, \text{Task}^*_{jk}, \text{Optional}) \qquad \forall j \in \mathcal{J}, k \in \mathcal{O}_j, m \in \mathcal{M}_{jk} \tag{A.3}$$

$$\text{ExactlyOne}(\text{PresenceOf}(\text{Task}_{jkm}) : m \in \mathcal{M}_{jk}) \qquad \forall j \in \mathcal{J}, k \in \mathcal{O}_j \tag{A.4}$$

$$\text{StartOf}(\text{Task}^*_{jk}) \leq \text{EndOf}(\text{Task}^*_{jk-1}) \qquad \forall j \in \mathcal{J}, k \in \mathcal{O}_j \tag{A.5}$$

$$\text{NoOverlap}(\text{Task}_{jkm} : j \in \mathcal{J}, k \in \mathcal{O}_j | m \in \mathcal{M}_{jk}) \qquad \forall m \in \mathcal{M} \tag{A.6}$$

$$C_{\max} = \max_{j \in \mathcal{J}}(\text{EndOf}(\text{Task}^*_{j|\mathcal{O}_j|})) \tag{A.7}$$

There are some slight differences compared to the CP Optimizer implementation. The first is in Constraint A.4: OR-Tools does not include a function that intrinsically links the machine selection to a task; however, a generic constraint is used to enforce presence of exactly one of the

[1] `https://developers.google.com/optimization/reference/python/sat/python/cp_model`

machine-variables for this task. Secondly, where CP Optimizer implements the EndBeforeStart function as an abstraction, this must be added by hand in OR-Tools as an inequality (Constraint A.5).

A.2. Sequence-Dependent Setup Times

Where CP Optimizer presents a concise function that takes a matrix for the SDST, this requires more work for OR-Tools. The model is extended with some additional constrains:

$$\begin{aligned}
&\mathrm{Arc}_{muv} = \mathrm{BinaryVar}() && \forall m \in \mathcal{M}, \forall u, v \in \mathcal{O} \cup D | u \neq v; m_u = m_v && \text{(A.8)}\\
&\mathrm{Circuit}(u, v, \mathrm{Arc}_{muv} | u, v \in \mathcal{O}) && \forall m \in \mathcal{M} && \text{(A.9)}\\
&\mathrm{Arc}_{muv} \leq \mathrm{PresenceOf}(u) && && \text{(A.10)}\\
&\mathrm{Arc}_{muv} \leq \mathrm{PresenceOf}(v) && && \text{(A.11)}\\
&\mathrm{If}(\mathrm{Arc}_{muv}, \mathrm{EndOf}(u) + s_{uv} \leq \mathrm{StartOf}(v)) && \forall m \in \mathcal{M}, \forall u, v \in \mathcal{O} | u \neq v && \text{(A.12)}\\
& && && \text{(A.13)}
\end{aligned}$$

Effectively, these constraints constitute a complete matrix of size $|\mathcal{O}| \times |\mathcal{O}|$ per machine m, similar to the matrix M_m for CP Optimizer. Respectively, these constraints add arc variables for all possible permutations of tasks on a machine, ensure that all activated arcs form one circuit (through the additional dummy variable), that all associated tasks are processed on this machine, and obey the selected permutation and setup time constraints.

A.3. Alternative Process Plans

Constraint A.5 is reformulated to look at the dependencies in the precedence graph:

$$\mathrm{EndOf}(\mathrm{Task}_{ja}) \leq \mathrm{StartOf}(\mathrm{Task}_{jb}) \qquad \forall j \in \mathcal{J},\ (a, b) \in \mathcal{N}_j \tag{A.14}$$

All modifications and additions with respect to optional tasks and flow variables is completely identical to the CP Optimizer model.

Constraint A.4 is not valid any more when tasks are not present in the solution. As such, it must be modified:

$$\sum_{m \in \mathcal{M}_{jk}} \mathrm{PresenceOf}(\mathrm{Task}_{jkm}) = \mathrm{PresenceOf}(\mathrm{Task}^*_{jk}) \qquad \forall j \in \mathcal{J}, k \in \mathcal{O}_j \tag{A.15}$$

$$\tag{A.16}$$

Unfortunately, this constraint is less performant than the ExactlyOne constraint, with regular FJSP problems taking roughly a 20% performance hit using this constraint instead of A.4.

B | Results

In this appendix, we report all upper and lower bounds per instance, including the runtime. Where applicable, best-known values are presented.

Note: some papers have an additional Kacem instance between 'Kacem1' and 'Kacem2' with a makespan of 14.

B.1. FJSP — Constraint Programming

In Table B.1, we present the results of both CP solvers for all classic datasets (see Section 6.1.1). Best-known bounds are taken from Dauzère-Pérès, Ding, et al. (2024). Results from Naderi and Roshanaei (2021) are included since they have a similar CP model, to validate our results. The maximum allowed runtime was set to 900 seconds: any instance for which this runtime is reached is not solved to optimality.

Table B.1.: Results of CP Optimizer and OR-Tools for classic datasets.

	Best-known		**Naderi et al.**		**CP Optimizer**			**OR-Tools**		
Name	LB	UB	LB	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
					Barnes					
mt10c1	927		927		927		6	927		2
mt10cc	908		908		908		4	908		< 1
mt10x	918		918		918		5	918		2
mt10xx	918		918		918		4	918		< 1
mt10xxx	918		918		918		5	918		1
mt10xy	905		905		905		3	905		< 1
mt10xyz	847		847		847		2	847		< 1
setb4c9	914		914		914		4	914		1
setb4cc	907		907		907		3	907		< 1
setb4x	925		925		925		8	925		4
setb4xx	925		925		925		9	925		4
setb4xxx	925		925		925		8	925		5
setb4xy	910		910		910		8	910		2
setb4xyz	902		902		902		4	902		1
seti5c12	1169		1169		1169		27	1169		8
seti5cc	1135		1135		1135		60	1135		30
seti5x	1198		1198		1198		9	1198		5
seti5xx	1194		1194		1194		13	1194		2
seti5xxx	1194		1194		1194		13	1194		3
seti5xy	1135		1135		1135		57	1135		29
seti5xyz	1125		1125		1125		61	1125		40
					Brandimarte					

Table B.1.: Results of CP Optimizer and OR-Tools for classic datasets (continued).

	Best-known		**Naderi et al.**		**CP Optimizer**			**OR-Tools**		
Name	LB	UB	LB	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
Mk01	40		40		40		< 1	40		< 1
Mk02	26		26		26		598	25	26	900
Mk03	204		204		204		< 1	204		2
Mk04	60		60		60		1	60		< 1
Mk05	172		172		136	172	900	158	172	900
Mk06	57		50	57	39	57	900	50	59	900
Mk07	139		139		133	139	900	133	139	900
Mk08	523		523		523		< 1	523		< 1
Mk09	307		307		307		1	307		10
Mk10	189	193	189	195	183	197	900	183	206	900
					Dauzère-Paulli					
01a	2505		2505		2505		21	2505		72
02a	2228		2228	2231	1691	2235	900	1644	2243	900
03a	2228		2228		1392	2228	900	1409	2235	900
04a	2503		2503		2503		14	2503		25
05a	2192	2203	2193	2219	1643	2216	900	1703	2224	900
06a	2163	2171	2163	2186	1351	2193	900	1364	2214	900
07a	2216	2254	2206	2276	2206	2320	900	2206	2308	900
08a	2061		2061	2070	1400	2068	900	1400	2121	900
09a	2061		2061	2062	1400	2062	900	1400	2084	900
10a	2212	2241	2197	2294	2197	2287	900	2197	2323	900
11a	2018	2037	2019	2066	1354	2063	900	1383	2083	900
12a	1969	1984	1969	2023	1310	2032	900	1310	2084	900
13a	2197	2236	2195	2252	2138	2271	900	2109	2309	900
14a	2161		2161	2164	1354	2164	900	1354	2207	900
15a	2161		2161	2162	1354	2162	900	1354	2206	900
16a	2193	2231	2189	2255	2138	2276	900	2138	2286	900
17a	2088	2105	2088	2143	1303	2147	900	1309	2183	900
18a	2057	2070	2056	2120	1289	2133	900	1289	2174	900
					Fattahi					
SFJS1			66		66		< 1	66		< 1
SFJS2			107		107		< 1	107		< 1
SFJS3			221		221		< 1	221		< 1
SFJS4			355		355		< 1	355		< 1
SFJS5			119		119		< 1	119		< 1
SFJS6			320		320		< 1	320		< 1
SFJS7			397		397		< 1	397		< 1
SFJS8			253		253		< 1	253		< 1
SFJS9			210		210		< 1	210		< 1
SFJS10			516		516		< 1	516		< 1
MFJS1			468		468		< 1	468		< 1
MFJS2			446		446		< 1	446		< 1
MFJS3			466		466		< 1	466		< 1

Table B.1.: Results of CP Optimizer and OR-Tools for classic datasets (continued).

	Best-known		Naderi et al.		CP Optimizer			OR-Tools		
Name	LB	UB	LB	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
MFJS4			554		554		< 1	554		< 1
MFJS5			514		514		< 1	514		< 1
MFJS6			634		634		< 1	634		< 1
MFJS7			879		879		2	879		< 1
MFJS8			884		884		2	884		2
MFJS9			1055		1055		30	1055		16
MFJS10			1196		1196		160	1196		734
Hurink edata										
abz5	1167				1167		2	1167		< 1
abz6	925				925		1	925		< 1
abz7	604	610			564	633	900	575	633	132
abz8	625	636			586	654	900	579	732	< 1
abz9	644				588	646	900	581	719	1
car1	6176				6176		< 1	6176		< 1
car2	6327				6327		< 1	6327		< 1
car3	6856				6856		< 1	6856		< 1
car4	7789				7789		< 1	7789		< 1
car5	7229				7229		< 1	7229		< 1
car6	7990				7990		2	7990		< 1
car7	6123				6123		< 1	6123		< 1
car8	7689				7689		< 1	7689		< 1
la01	609		609		609		< 1	609		< 1
la02	655		655		655		< 1	655		< 1
la03	550		550		550		< 1	550		< 1
la04	568		568		568		< 1	568		< 1
la05	503		503		503		< 1	503		< 1
la06	833		833		833		< 1	833		< 1
la07	762		762		762		< 1	762		< 1
la08	845		845		845		< 1	845		< 1
la09	878		878		878		< 1	878		< 1
la10	866		866		866		< 1	866		< 1
la11	1103		1103		1103		< 1	1103		< 1
la12	960		960		960		< 1	960		< 1
la13	1053		1053		1053		< 1	1053		< 1
la14	1123		1123		1123		< 1	1123		< 1
la15	1111		1111		1111		< 1	1111		< 1
la16	892		892		892		< 1	892		< 1
la17	707		707		707		1	707		< 1
la18	842		842		842		1	842		< 1
la19	796		796		796		1	796		< 1
la20	857		857		857		< 1	857		< 1
la21	1009		1009		1009		100	1009		80
la22	880		880		880		4	880		3
la23	950		950		950		5	950		2

Table B.1.: Results of CP Optimizer and OR-Tools for classic datasets (continued).

	Best-known		**Naderi et al.**		**CP Optimizer**			**OR-Tools**		
Name	LB	UB	LB	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
la24	908		908		908		17	908		26
la25	936		936		936		10	936		9
la26	1106		1106		1106	1117	900	1106	1112	900
la27	1181		1181		1181		196	1181		145
la28	1142		1142		1124	1142	900	1119	1142	900
la29	1107		1107		1069	1110	900	1080	1107	900
la30	1188		1148	1192	1148	1197	900	1155	1207	900
la31	1532		1532		1490	1541	900	1532		362
la32	1698		1698		1698		5	1698		34
la33	1547		1547		1547		23	1547		22
la34	1599		1599		1599		146	1599		35
la35	1736		1736		1736		1	1736		4
la36	1160		1160		1160		31	1160		16
la37	1397		1397		1397		2	1397		1
la38	1141		1141		1141		98	1141		90
la39	1184		1184		1184		11	1184		8
la40	1144		1144		1144		100	1144		140
mt06	55		55		55		< 1	55		< 1
mt10	871		871		871		2	871		< 1
mt20	1088		1088		1088		< 1	1088		2
orb1	977				977		8	977		10
orb2	865				865		4	865		< 1
orb3	951				951		6	951		3
orb4	984				984		5	984		1
orb5	842				842		2	842		< 1
orb6	958				958		5	958		2
orb7	389				389		2	389		< 1
orb8	894				894		< 1	894		< 1
orb9	933				933		2	933		< 1
orb10	933				933		1	933		< 1
					Hurink rdata					
abz5	954				954		2	954		4
abz6	807				807		< 1	807		< 1
abz7	493	522			448	533	900	463	545	900
abz8	507	535			469	550	900	489	564	900
abz9	517	536			487	546	900	506	562	900
car1	5034				3559	5047	900	5034		755
car2	5985				4078	5987	900	5985		434
car3	5622				3979	5625	900	4563	5625	900
car4	6514				4521	6514	900	5250	6514	900
car5	5615				5615		89	5615		72
car6	6147				6147		1	6147		< 1
car7	4425				4425		< 1	4425		< 1
car8	5692				5692		< 1	5692		< 1

Table B.1.: Results of CP Optimizer and OR-Tools for classic datasets (continued).

	Best-known		Naderi et al.		CP Optimizer			OR-Tools		
Name	LB	UB	LB	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
la01	570		570		434	572	900	570		434
la02	529		529		529		130	529		242
la03	477		477		477		577	477		280
la04	502		502		381	502	900	502		556
la05	457		457		380	457	900	428	457	900
la06	799		799		502	799	900	639	799	900
la07	749		749		453	749	900	572	749	900
la08	765		765		451	765	900	623	765	900
la09	853		853		560	853	900	677	853	900
la10	804		804		480	804	900	663	804	900
la11	1071		1071		668	1071	900	729	1071	900
la12	936		936		634	936	900	720	936	900
la13	1038		1038		541	1038	900	718	1038	900
la14	1070		1070		658	1070	900	768	1070	900
la15	1089		1089		631	1089	900	715	1089	900
la16	717		717		717		< 1	717		< 1
la17	646		646		646		< 1	646		< 1
la18	666		666		666		1	666		< 1
la19	700		700		700		< 1	700		< 1
la20	756		756		756		< 1	756		< 1
la21	808	825	805	839	719	852	900	759	850	900
la22	741	753	733	764	677	772	900	723	762	900
la23	816	831	809	850	673	851	900	733	853	900
la24	775	795	775	811	717	811	900	742	805	900
la25	768	779	751	784	736	789	900	749	787	900
la26	1056	1057	1055	1063	717	1065	900	760	1088	900
la27	1085		1085	1089	769	1089	900	826	1097	900
la28	1075	1076	1075	1082	758	1081	900	787	1094	900
la29	993	994	993	1003	737	999	900	771	1004	900
la30	1068	1071	1068	1082	815	1076	900	852	1094	900
la31	1520		1520		1006	1522	900	1006	1524	900
la32	1657		1657		976	1658	900	1019	1659	900
la33	1497		1497		801	1499	900	839	1500	900
la34	1535		1535		874	1536	900	939	1535	900
la35	1549		1549	1550	813	1550	900	824	1551	900
la36	1023		1023		1023		8	1023		29
la37	1062		1062		1062		278	1055	1067	900
la38	954		954		954		8	954		31
la39	1011		1011		1011		55	1011		153
la40	955		955		955		830	955		817
mt06	47		47		47		< 1	47		< 1
mt10	686		686		686		1	686		1
mt20	1022		1022		632	1022	900	700	1022	900
orb1	746				746		1	746		< 1
orb2	696				696		1	696		1

Table B.1.: Results of CP Optimizer and OR-Tools for classic datasets (continued).

	Best-known		Naderi et al.		CP Optimizer			OR-Tools		
Name	LB	UB	LB	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
orb3	712				712		2	712		2
orb4	753				753		< 1	753		< 1
orb5	639				639		1	639		< 1
orb6	754				754		1	754		1
orb7	302				302		2	302		3
orb8	639				639		2	639		2
orb9	694				694		< 1	694		< 1
orb10	742				742		1	742		2
					Hurink vdata					
abz5	859				859		< 1	859		10
abz6	742				742		< 1	742		2
abz7	492				410	494	900	410	527	900
abz8	506	507			443	509	900	443	540	900
abz9	497				467	500	900	467	526	900
car1	5005				3312	5006	900	3719	5006	900
car2	5929				3794	5929	900	4224	5929	900
car3	5597				3518	5598	900	3956	5598	900
car4	6514				3883	6514	900	4588	6514	900
car5	4909	4910			4037	4917	900	4241	4914	900
car6	5486				5486		< 1	5486		< 1
car7	4281				4281		< 1	4281		< 1
car8	4613				4613		< 1	4613		1
la01	570		570		413	570	900	474	570	900
la02	529		529		394	529	900	442	529	900
la03	477		477		349	477	900	418	477	900
la04	502		502		379	502	900	452	502	900
la05	457		457		380	457	900	399	457	900
la06	799		799		441	799	900	504	799	900
la07	749		749		422	749	900	446	749	900
la08	765		765		370	765	900	502	765	900
la09	853		853		407	853	900	526	853	900
la10	804		804		443	804	900	485	804	900
la11	1071		1071		448	1071	900	533	1071	900
la12	936		936		416	936	900	486	936	900
la13	1038		1038		444	1038	900	544	1038	900
la14	1070		1070		443	1070	900	550	1070	900
la15	1089		1089		401	1089	900	519	1089	900
la16	717		717		717		< 1	717		1
la17	646		646		646		< 1	646		2
la18	663		663		663		< 1	663		3
la19	617		617		617		< 1	617		3
la20	756		756		756		< 1	756		1
la21	800		800	802	717	802	900	717	816	900
la22	733		733	735	619	733	900	619	747	900

Table B.1.: Results of CP Optimizer and OR-Tools for classic datasets (continued).

	Best-known		**Naderi et al.**		**CP Optimizer**			**OR-Tools**		
Name	LB	UB	LB	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
la23	809		809	810	640	812	900	640	819	900
la24	773		773	774	704	774	900	704	784	900
la25	751		751	754	723	752	900	723	760	900
la26	1052		1052		717	1052	900	717	1059	900
la27	1084		1084		686	1085	900	686	1092	900
la28	1069		1069		756	1069	900	756	1076	900
la29	993		993	994	723	994	900	723	1000	900
la30	1068		1068	1069	726	1069	900	726	1075	900
la31	1520		1520		717	1520	900	717	1521	900
la32	1657		1657		756	1658	900	756	1659	900
la33	1497		1497	1498	723	1498	900	723	1500	900
la34	1535		1535		656	1535	900	656	1536	900
la35	1549		1549		647	1550	900	647	1552	900
la36	948		948		948		< 1	948		62
la37	986		986		986		< 1	986		97
la38	943		943		943		< 1	943		58
la39	922		922		922		< 1	922		96
la40	955		955		955		< 1	955		32
mt06	47		47		47		< 1	47		< 1
mt10	655		655		655		< 1	655		3
mt20	1022		1022		387	1022	900	477	1022	900
orb1	695				695		< 1	695		2
orb2	620				620		< 1	620		3
orb3	648				648		< 1	648		2
orb4	753				753		< 1	753		3
orb5	584				584		< 1	584		4
orb6	715				715		< 1	715		2
orb7	275				275		< 1	275		3
orb8	573				573		< 1	573		2
orb9	659				659		< 1	659		2
orb10	681				681		< 1	681		2
					Kacem					
1			11		11		< 1	11		< 1
2			11		11		< 1	11		< 1
3			7		7		< 1	7		< 1
4			11		11		20	11		42

In Table B.2, we present the results of both CP solvers for the Behnke datasets (see Section 6.1.1). Bounds [1] by Behnke and Geiger (2012), [2] by Lei et al. (2022) and [3] by Wan et al. (2024). The maximum allowed runtime was set to 900 seconds: any instance for which this runtime is reached is not solved to optimality.

Table B.2.: Results of CP Optimizer and OR-Tools for Behnke dataset.

	[1]		[2]	[3]	**CP Optimizer**			**OR-Tools**		
Name	LB	UB	UB	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
1	70	91				87	22		87	1
2	75	91				87	13		87	2
3	79	91				86	11		86	1
4	76	97				84	9		84	2
5	71	91				87	14		87	1
6	78	131	143		74	115	900		114	453
7	84	130	142		76	117	900		117	270
8	76	128	139		78	125	900		125	668
9	74	129	144		73	113	900		113	107
10	81	133	146		76	124	900		123	141
11	163	259	250	278	77	220	900	99	220	900
12	157	251	247	247	81	213	900	101	213	900
13	160	252	249	261	76	214	900	99	217	900
14	164	258	257	262	81	225	900	105	230	900
15	159	262	255	260	82	223	900	99	225	900
16	327	566	437	451	80	391	900	89	399	900
17	320	535	430	438	83	392	900	91	401	900
18	321	555	428	442	82	400	900	96	408	900
19	323	532	423	447	81	400	900	93	401	900
20	322	522	427	444	84	402	900	95	411	900
21	78	85				85	6		85	2
22	69	87				87	6		87	1
23	72	86				85	4		85	1
24	70	87				87	8		87	1
25	80	87				87	6		87	2
26	70	122	129		74	114	900		113	174
27	81	132	140		84	124	900		122	434
28	73	123	133		75	115	900		114	351
29	75	125	138		78	118	900		117	625
30	80	127	142		82	121	900	117	121	900
31	79	272	271	264	85	228	900	106	234	900
32	77	259	267	255	79	224	900	97	228	900
33	77	245	261	253	80	225	900	104	227	900
34	78	265	250	262	80	222	900	104	227	900
35	79	253	249	252	82	214	900	101	214	900
36	152	531	442	446	82	391	900	95	405	900
37	153	536	444	435	84	396	900	107	405	900
38	151	527	454	434	83	393	900	98	395	900
39	153	516	471	454	86	391	900	92	401	900
40	156	521	460	456	86	419	900	95	423	900
41	68	87				90	3		90	2
42	75	87				91	1		91	1
43	68	86				91	4		91	1
44	68	85				97	4		97	1
45	68	87				91	5		91	3

Table B.2.: Results of CP Optimizer and OR-Tools for Behnke dataset (continued).

	[1]		[2]	[3]	**CP Optimizer**			**OR-Tools**		
Name	LB	UB	UB	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
46	73	124	145		81	127	900	114	126	900
47	76	126	144		89	125	900	104	128	901
48	74	134	141		78	124	900	100	127	900
49	66	121	134		80	125	900	125		278
50	73	131	147		89	131	900	110	133	900
51	75	259	253	245	86	232	900	108	256	900
52	76	255	242	240	83	222	900	101	239	900
53	74	257	256	244	84	232	900	105	252	900
54	75	267	268	258	90	236	900	108	257	900
55	77	256	262	248	86	231	900	99	254	900
56	99	538	439	442	91	420	900	110	478	900
57	99	535	442	433	85	408	900	101	460	900
58	100	531	442	439	89	404	900	98	469	900
59	99	532	443	441	89	406	900	110	465	900
60	101	537	458	444	90	404	900	100	448	900

In Table B.3, we present the results of both CP solvers for the Naderi datasets (see Section 6.1.1). Best-known upper bounds by Lan and Berkhout (2025). The maximum allowed runtime was set to 900 seconds: any instance for which this runtime is reached is not solved to optimality.

Table B.3.: Results of CP Optimizer and OR-Tools for Naderi dataset.

	Best-known	**CP Optimizer**			**OR-Tools**		
Name	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
1	1009	578	1025	900	589	1019	900
2	1158	585	1159	900	619	1159	900
3	970	420	1002	900	427	987	900
4	1121	542	1121	900	542	1123	900
5	2183	1153	2252	900	1155	2266	900
6	2413	1170	2415	900	1170	2418	900
7	2097	970	2212	900	970	2162	900
8	2397	1048	2397	900	1048	2404	900
9	641	505	640	900	505	663	900
10	691	554	692	900	554	698	900
11	641	512	647	900	512	653	900
12	720	522	720	900	522	722	900
13	1336	1058	1342	900	1058	1403	900
14	1602	1211	1602	900	1211	1658	900
15	1542	1034	1567	900	1034	1601	900
16	1565	1252	1565	900	1252	1605	900
17	540	518	535	900	518	559	900
18	601	580	603	900	580	618	900
19	452	539		245	539	561	900
20	638	638		1	638		120

Table B.3.: Results for Naderi dataset (continued).

	Best-known	CP Optimizer			OR-Tools		
Name	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
21	1218	1051	1198	900	1051	1279	900
22	1280	1167	1283	900	1167	1411	900
23	1053	949	1021	900	949	1112	900
24	1158	1158		3	1158	1228	900
25	1692	641	1750	900	646	1729	900
26	1838	594	1840	900	572	1839	900
27	1610	553	1691	900	553	1649	900
28	1896	620	1896	900	620	1897	900
29	3590	1080	3789	900	1091	3743	900
30	4008	1192	4010	900	1192	4013	900
31	3314	996	3545	900	996	3550	900
32	3992	1180	3993	900	1180	3999	900
33	1036	517	1084	900	517	1066	900
34	1324	613	1326	900	613	1326	900
35	1124	505	1199	900	505	1169	900
36	1296	666	1296	900	666	1304	900
37	2266	986	2426	900	986	2505	900
38	2650	1255	2651	900	1255	2662	900
39	2158	1064	2320	900	1064	2296	900
40	2521	1147	2520	900	1147	2568	900
41	836	507	874	900	507	872	900
42	909	678	908	900	678	914	900
43	763	437	809	900	437	803	900
44	920	596	921	900	596	931	900
45	1784	1001	1859	900	1001	1888	900
46	1991	1152	1992	900	1152	2107	900
47	1662	973	1736	900	973	1778	901
48	1865	1241	1866	900	1241	1972	901
49	2474	663	2566	900	648	2558	900
50	2719	784	2720	900	719	2720	900
51	2239	518	2307	900	518	2311	900
52	2717	563	2717	900	563	2718	900
53	4948	1119	5217	900	1132	5295	900
54	5532	1417	5534	900	1322	5563	900
55	4729	1113	5039	900	1113	4852	900
56	5493	1242	5494	900	1242	5502	900
57	1563	563	1660	900	564	1653	900
58	1739	622	1741	900	622	1740	900
59	1482	555	1602	900	555	1528	900
60	1866	577	1867	900	577	1870	900
61	3320	1012	3588	900	1008	3791	900
62	3616	1298	3618	900	1298	3650	900
63	2994	993	3272	900	993	3164	901
64	3664	1241	3664	900	1241	3703	900
65	1122	562	1196	900	562	1154	900

Table B.3.: Results for Naderi dataset (continued).

	Best-known	CP Optimizer			OR-Tools		
Name	UB	LB	UB	CPU (s)	LB	UB	CPU (s)
66	1335	584	1336	900	584	1340	900
67	1091	551	1189	900	551	1133	900
68	1358	670	1359	900	670	1364	900
69	2492	1022	2669	900	1022	2681	900
70	2679	1208	2679	900	1208	2751	900
71	2154	934	2361	900	934	2307	901
72	2649	1117	2649	900	1117	2809	901
73	3519	724	3667	900	728	3698	900
74	3705	905	3707	900	831	3707	900
75	3181	539	3271	900	539	3317	900
76	3789	584	3790	900	584	3790	900
77	6920	1390	7345	900	1316	7470	900
78	7569	1431	7569	900	1334	7915	900
79	6786	1083	7000	900	1083	7556	900
80	7824	1178	7825	900	1178	7225	900
81	2111	528	2263	900	528	2205	900
82	2519	685	2521	900	685	2521	900
83	2098	494	2260	900	494	2145	900
84	2579	613	2581	900	613	2589	900
85	4517	1010	4892	900	1009	4845	901
86	5108	1229	5109	900	1229	5431	901
87	4512	1020	4945	900	1020	5123	900
88	4994	1177	4995	900	1177	5216	900
89	1666	521	1795	900	521	1735	900
90	1778	576	1779	900	576	1782	900
91	1558	528	1713	900	528	1655	900
92	1923	705	1924	900	705	1948	900
93	3240	928	3555	900	928	3458	901
94	3792	1191	3790	900	1191	3853	901
95	3273	970	3594	900	970	3590	900
96	3897	1284	3896	900	1284	5898	900

B.2. SDST — Constraint Programming

In Table B.4, we present the results of both CP solvers for all SDST datasets (see Section 6.1.2). The maximum allowed runtime was set to 900 seconds: any instance for which this runtime is reached is not solved to optimality.

Table B.4.: Results from CP Optimizer and OR-Tools for the SDST datasets.

	CP Optimizer			**OR-Tools**		
Name	UB	LB	t (s)	UB	LB	t (s)
			Fattahi			
SFJS1	70		< 1	70		< 1
SFJS2	112		< 1	112		< 1
SFJS3	233		< 1	233		< 1
SFJS4	374		< 1	374		< 1
SFJS5	126		< 1	126		< 1
SFJS6	334		< 1	334		< 1
SFJS7	397		< 1	397		< 1
SFJS8	262		< 1	262		< 1
SFJS9	220		< 1	220		< 1
SFJS10	541		< 1	541		< 1
MFJS1	482		< 1	482		< 1
MFJS2	468		< 1	468		< 1
MFJS3	490		< 1	490		< 1
MFJS4	591		< 1	591		< 1
MFJS5	546		< 1	546		< 1
MFJS6	659		< 1	659		1
MFJS7	939		2	939		5
MFJS8	934		4	934		55
MFJS9	1130		131	958	1130	900
MFJS10	1086	1276	900	1107	1284	900
			Hurink edata			
la21	721		3	721		14
la22	737		2	737		10
la23	652		3	652		105
la24	673		4	673		267
la25	602		5	602		239
la26	833	952	900	864	963	900
la27	749	914	900	755	930	900
la28	845	947	900	834	940	900
la29	856	988	900	874	997	900
la30	951		490	909	987	900
la31	1043	1243	900	1104	1271	900
la32	960	1062	900	960	1162	900
la33	1053	1181	900	1065	1220	900
la34	1121	1235	900	1125	1254	900
la35	1136	1259	900	1153	1351	900
la36	1007		3	1007		84
la37	851		4	851		169
la38	985		5	985		90
la39	951		15	951		256
la40	997		4	997		122

B.3. Blocking & APP — Constraint Programming

In Table B.5, we present the results of both CP solvers for all Blocking and APP datasets (see Section 6.1.3 and Section 6.1.4). The maximum allowed runtime was set to 60 seconds: any instance for which this runtime is reached is not solved to optimality.

Table B.5.: Results from CP Optimizer and OR-Tools for the Blocking and APP datasets.

	Alternative Process Plans						**Blocking**					
	CP Optimizer			**OR-Tools**			**CP Optimizer**			**OR-Tools**		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
Barnes												
mt10c1	482		2	482		< 1	869	1024	60	954	1038	60
mt10cc	516		1	516		< 1	861	997	60	975		48
mt10x	542		1	542		< 1	868	1028	60	979	1025	60
mt10xx	474		1	474		< 1	868	1028	60	1025		34
mt10xxx	496		1	496		< 1	868	1025	60	982	1025	60
mt10xy	543		< 1	543		< 1	976		25	976		10
mt10xyz	605		< 1	605		< 1	914		21	914		58
setb4c9	620		2	620		< 1	911	1375	60	974	1354	60
setb4cc	634		1	634		< 1	911	1319	60	968	1300	60
setb4x	579		1	579		< 1	911	1325	60	980	1353	60
setb4xx	529		1	529		< 1	911	1352	60	978	1333	60
setb4xxx	698		1	698		< 1	911	1323	60	978	1315	60
setb4xy	477		1	477		< 1	896	1369	60	952	1228	60
setb4xyz	519		1	519		< 1	892	1219	60	948	1207	60
seti5c12	681		3	681		< 1	1148	1725	60	1205	1651	60
seti5cc	628		3	628		< 1	1069	1578	60	1149	1564	60
seti5x	779		< 1	779		< 1	1189	1668	60	1224	1692	60
seti5xx	618		4	618		< 1	1189	1664	60	1224	1570	60
seti5xxx	630		2	630		< 1	1189	1598	60	1224	1622	60
seti5xy	571		< 1	571		< 1	1069	1578	60	1149	1578	60
seti5xyz	684		3	684		< 1	1069	1403	60	1140	1477	60
Brandimarte												
Mk01	18		< 1	18		< 1	42		2	42		6
Mk02	13		1	13		< 1	25	33	60	26	34	60
Mk03	104		< 1	104		< 1	204	205	60	204	213	60
Mk04	26		< 1	26		< 1	55	69	60	63	73	60
Mk05	44	84	60	78	84	60	136	213	60	156	217	60
Mk06	24		4	24		2	39	72	60	45	82	60
Mk07	46	67	60	58	68	60	133	181	60	133	195	60
Mk08	234		3	234		< 1	523	634	60	523	627	60
Mk09	144	146	60	144		12	307	388	60	307	453	60
Mk10	73	96	60	72	99	60	183	294	60	183	371	60
Dauzère-Paulli												
01a	874	1233	60	1143	1233	60	2522	3961	60	2555	4396	60
02a	739	847	60	766	841	60	1691	3042	60	1650	3633	60

Table B.5.: Results from CP Optimizer and OR-Tools for the Blocking and APP datasets (continued).

	Alternative Process Plans						Blocking					
	CP Optimizer			OR-Tools			CP Optimizer			OR-Tools		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
03a	1273		12	1273		16	1393	2532	60	1392	3364	60
04a	1051		10	1051		4	2503	3838	60	2552	4281	60
05a	744	987	60	848	995	60	1643	3179	60	1628	3445	60
06a	1032	1101	60	1032	1102	60	1352	2509	60	1355	3239	60
07a	1278		5	1278		1	2206	4016	60	2224	4978	60
08a	807	895	60	807	897	60	1400	2834	60	1405	3429	60
09a	846	907	60	846	972	60	1400	2305	60	1400	4215	60
10a	1208		14	1208		6	2202	3939	60	2218	4397	60
11a	1012		55	1012	1048	60	1356	2785	60	1377	4090	60
12a	977		21	977	1001	60	1310	2316	60	1310	—	60
13a	814	899	60	868	905	60	2158	3998	60	2113	4947	60
14a	1072	1129	60	1072	1179	60	1354	2763	60	1354	—	60
15a	1000	1144	60	1000	1261	60	1354	2371	60	1354	—	60
16a	1077	1219	60	1086	1216	60	2142	4305	60	2093	5456	60
17a	717	914	60	717	992	60	1303	2745	60	1305	—	60
18a	770	959	60	770	1013	60	1289	2374	60	1289	—	60
Fattahi												
SFJS1	24		< 1	24		< 1	66		< 1	66		< 1
SFJS2	43		< 1	43		< 1	107		< 1	107		< 1
SFJS3	106		< 1	106		< 1	256		< 1	256		< 1
SFJS4	179		< 1	179		< 1	396		< 1	396		< 1
SFJS5	73		< 1	73		< 1	128		< 1	128		< 1
SFJS6	160		< 1	160		< 1	320		< 1	320		< 1
SFJS7	247		< 1	247		< 1	397		< 1	397		< 1
SFJS8	146		< 1	146		< 1	253		< 1	253		< 1
SFJS9	80		< 1	80		< 1	210		< 1	210		< 1
SFJS10	173		< 1	173		< 1	533		< 1	533		< 1
MFJS1	285		< 1	285		< 1	473		< 1	473		< 1
MFJS2	273		< 1	273		< 1	448		< 1	448		< 1
MFJS3	145		< 1	145		< 1	473		< 1	473		< 1
MFJS4	154		< 1	154		< 1	566		< 1	566		< 1
MFJS5	223		< 1	223		< 1	559		< 1	559		< 1
MFJS6	215		< 1	215		< 1	667		< 1	667		< 1
MFJS7	546		< 1	546		< 1	954		3	954		2
MFJS8	482		1	482		< 1	953		12	953		12
MFJS9	519		< 1	519		< 1	808	1178	60	952	1160	60
MFJS10	640		2	640		< 1	956	1358	60	1061	1344	60
Hurink edata												
abz5	764		< 1	764		< 1	1108	1435	60	1239	1467	60
abz6	483		< 1	483		< 1	890	1118	60	1019	1162	60
abz7	349		9	349		1	565	1082	60	581	1063	60

Table B.5.: Results from CP Optimizer and OR-Tools for the Blocking and APP datasets (continued).

	Alternative Process Plans						Blocking					
	CP Optimizer			OR-Tools			CP Optimizer			OR-Tools		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
abz8	349		5	349		1	592	1048	60	613	1103	60
abz9	304		9	304		1	589	1024	60	616	1054	60
car1	2596		< 1	2596		< 1	6194	7022	60	6338	7022	60
car2	3128		< 1	3128		< 1	6214	7409	60	6176	7435	60
car3	2351		< 1	2351		< 1	6825	7947	60	6874	7856	60
car4	3182		2	3182		< 1	7789	8474	60	7792	8405	60
car5	3366		< 1	3366		< 1	7932		16	7932		22
car6	4576		< 1	4576		< 1	8289		4	8289		4
car7	2864		< 1	2864		< 1	6270		< 1	6270		< 1
car8	3947		< 1	3947		< 1	8130		5	8130		2
la01	322		< 1	322		< 1	791		17	791		37
la02	302		< 1	302		< 1	758		27	731	758	60
la03	274		< 1	274		< 1	653		13	653		8
la04	320		< 1	320		< 1	689		11	689		32
la05	286		< 1	286		< 1	619		13	619		27
la06	434		< 1	434		< 1	833	1109	60	838	1107	60
la07	450		< 1	450		< 1	749	989	60	752	1017	60
la08	377		< 1	377		< 1	845	1109	60	850	1051	60
la09	455		2	455		< 1	854	1198	60	870	1147	60
la10	404		< 1	404		< 1	866	1122	60	887	1093	60
la11	508		3	508		< 1	1043	1486	60	1015	1466	60
la12	478		2	478		< 1	960	1297	60	943	1310	60
la13	608		2	608		< 1	1053	1474	60	1056	1388	60
la14	462		1	462		< 1	1121	1480	60	1124	1468	60
la15	426		2	426		< 1	1136	1499	60	1152	1501	60
la16	516		< 1	516		< 1	845	1020	60	970	1038	60
la17	432		< 1	432		< 1	723	866	60	807	866	60
la18	540		< 1	540		< 1	809	999	60	944	990	60
la19	535		1	535		< 1	743	992	60	896	1023	60
la20	658		< 1	658		< 1	998		49	981	998	60
la21	557		2	557		< 1	930	1581	60	976	1532	60
la22	447		2	447		< 1	869	1325	60	938	1300	60
la23	679		< 1	679		< 1	950	1518	60	977	1453	60
la24	538		2	538		< 1	899	1424	60	945	1412	60
la25	477		2	477		< 1	919	1421	60	975	1397	60
la26	623		6	623		< 1	1106	1981	60	1099	2056	60
la27	704		2	704		< 1	1181	1995	60	1191	2047	60
la28	592		2	592		< 1	1126	1984	60	1142	2153	60
la29	583		4	583		< 1	1072	1813	60	1090	1894	60
la30	659		4	659		< 1	1148	1812	60	1186	2078	60
la31	927		25	927		5	1490	2870	60	1455	2949	60
la32	955		12	955		3	1698	3005	60	1698	3013	60

Table B.5.: Results from CP Optimizer and OR-Tools for the Blocking and APP datasets (continued).

	Alternative Process Plans						Blocking					
	CP Optimizer			OR-Tools			CP Optimizer			OR-Tools		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
la33	937		11	937		1	1547	2597	60	1548	3171	60
la34	876		41	876		4	1603	3009	60	1577	3131	60
la35	858		21	858		6	1736	2902	60	1736	3158	60
la36	590		3	590		< 1	1090	1753	60	1174	1726	60
la37	808		3	808		< 1	1397	1872	60	1414	1831	60
la38	597		2	597		< 1	1068	1840	60	1137	1721	60
la39	753		2	753		< 1	1160	1735	60	1210	1639	60
la40	833		< 1	833		< 1	1069	1604	60	1148	1747	60
mt06	34		< 1	34		< 1	58		< 1	58		< 1
mt10	529		< 1	529		< 1	821	1110	60	928	1021	60
mt20	584		< 1	584		< 1	1088	1417	60	1091	1460	60
orb1	538		2	538		< 1	872	1121	60	999	1111	60
orb2	602		2	602		< 1	815	1009	60	941	1035	60
orb3	435		1	435		< 1	880	1080	60	997	1097	60
orb4	476		< 1	476		< 1	900	1135	60	1053	1134	60
orb5	509		< 1	509		< 1	829	966	60	916	966	60
orb6	491		2	491		< 1	874	1153	60	1019	1152	60
orb7	225		< 1	225		< 1	362	464	60	424	484	60
orb8	552		2	552		< 1	955		36	931	955	60
orb9	639		1	639		< 1	1001		53	1001		36
orb10	543		< 1	543		< 1	920	1110	60	1054	1110	60
							Hurink rdata					
abz5	534		< 1	534		< 1	927	1187	60	983	1205	60
abz6	413		< 1	413		< 1	807	877	60	811	937	60
abz7	304		2	304		2	448	816	60	452	934	60
abz8	280		2	280		2	469	899	60	478	1000	60
abz9	289		1	289		1	491	861	60	483	991	60
car1	2706		4	2706		< 1	3614	5946	60	3679	6100	60
car2	1598	3229	60	3229		1	4078	6977	60	4266	6781	60
car3	2379		9	2379		< 1	3979	6443	60	4057	6734	60
car4	3098		41	3098		< 1	4521	7389	60	4850	7339	60
car5	3034		4	3034		< 1	4813	6701	60	5280	6783	60
car6	3600		< 1	3600		< 1	6364		2	6364		6
car7	2985		< 1	2985		< 1	4763		2	4763		3
car8	3511		< 1	3511		< 1	6036		3	6036		11
la01	344		< 1	344		< 1	443	674	60	500	660	60
la02	297		< 1	297		< 1	426	629	60	503	627	60
la03	237		1	237		< 1	394	564	60	421	548	60
la04	248		< 1	248		< 1	381	600	60	483	608	60
la05	314		< 1	314		< 1	380	553	60	426	553	60
la06	326	481	60	481		1	502	950	60	522	986	60

Table B.5.: Results from CP Optimizer and OR-Tools for the Blocking and APP datasets (continued).

	Alternative Process Plans						Blocking					
	CP Optimizer			OR-Tools			CP Optimizer			OR-Tools		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
la07	245	365	60	364		8	453	873	60	512	916	60
la08	382		22	382		< 1	451	924	60	530	906	60
la09	279	416	60	414		4	560	1022	60	576	1027	60
la10	413		19	413		2	480	987	60	535	979	60
la11	212	463	60	320	462	60	668	1276	60	625	1374	60
la12	284	430	60	353	428	60	634	1095	60	593	1167	60
la13	257	478	60	351	477	60	541	1196	60	581	1304	60
la14	298	544	60	386	544	60	658	1258	60	645	1335	60
la15	287	532	60	367	531	60	631	1268	60	642	1366	60
la16	572		< 1	572		< 1	717	811	60	723	829	60
la17	397		< 1	397		< 1	646	686	60	648	692	60
la18	450		< 1	450		< 1	673	729	60	699	776	60
la19	575		< 1	575		< 1	691	846	60	721	836	60
la20	536		< 1	536		< 1	756	803	60	776	846	60
la21	594		< 1	594		< 1	731	1219	60	752	1184	60
la22	489		3	489		1	690	1078	60	714	1189	60
la23	540		1	540		< 1	674	1175	60	703	1252	60
la24	441		23	441		17	736	1194	60	732	1164	60
la25	540		< 1	540		< 1	736	1113	60	744	1212	60
la26	414	497	60	420	495	60	717	1511	60	727	1668	60
la27	512	557	60	529	552	60	776	1619	60	790	1793	60
la28	509	563	60	509	559	60	758	1509	60	763	1790	60
la29	552		4	552		2	738	1432	60	750	1605	60
la30	553	584	60	556	586	60	821	1482	60	840	1590	60
la31	459	744	60	489	741	60	1006	2179	60	936	2622	60
la32	459	817	60	555	812	60	976	2449	60	943	2899	60
la33	527	820	60	538	819	60	801	2208	60	785	2626	60
la34	630	849	60	630	847	60	874	2183	60	868	2661	60
la35	495	883	60	518	883	60	813	2269	60	823	2700	60
la36	580		3	580		< 1	1025	1331	60	1027	1479	60
la37	711		< 1	711		< 1	1030	1455	60	1048	1640	60
la38	666		< 1	666		< 1	955	1283	60	972	1408	60
la39	692		1	692		1	1008	1383	60	999	1499	60
la40	637		4	637		< 1	955	1319	60	955	1451	60
mt06	28		< 1	28		< 1	48		< 1	48		< 1
mt10	438		1	438		< 1	686	754	60	711	751	60
mt20	253	506	60	388	502	60	632	1234	60	630	1305	60
orb1	467		< 1	467		< 1	746	857	60	773	857	60
orb2	454		< 1	454		< 1	758		45	722	812	60
orb3	400		< 1	400		< 1	695	815	60	736	852	60
orb4	398		< 1	398		< 1	753	856	60	770	830	60
orb5	357		< 1	357		< 1	629	739	60	678	735	60

Table B.5.: Results from CP Optimizer and OR-Tools for the Blocking and APP datasets (continued).

	Alternative Process Plans						Blocking					
	CP Optimizer			OR-Tools			CP Optimizer			OR-Tools		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
orb6	439		2	439		< 1	774	916	60	798	851	60
orb7	250		< 1	250		< 1	289	349	60	310	363	60
orb8	471		1	471		< 1	710		49	680	730	60
orb9	346		< 1	346		< 1	708	767	60	731	759	60
orb10	455		< 1	455		< 1	730	832	60	789	850	60
Hurink vdata												
abz5	560		< 1	560		< 1	859		3	859	1033	60
abz6	592		< 1	592		< 1	742		< 1	742	761	60
abz7	345		2	345		10	410	704	60	410	1242	60
abz8	312		8	312		6	443	710	60	443	1411	60
abz9	382		1	382		4	467	692	60	467	1264	60
car1	1564	2387	60	2386		27	3315	5704	60	3507	5853	60
car2	1551	3144	60	3144		7	3794	6621	60	3777	6455	60
car3	1928	2719	60	2248	2709	60	3518	6458	60	3573	6528	60
car4	1937	3445	60	3444		9	3883	7285	60	3899	7347	60
car5	3181		< 1	3181		< 1	4037	5758	60	4202	6050	60
car6	3158		< 1	3158		< 1	5486		< 1	5486		2
car7	2883		< 1	2883		< 1	4281		< 1	4281		1
car8	3837		< 1	3837		< 1	4613		< 1	4613		23
la01	292		5	292		1	413	627	60	454	652	60
la02	234		3	234		< 1	394	582	60	468	581	60
la03	223		7	223		2	349	546	60	410	539	60
la04	281		< 1	281		< 1	388	583	60	444	584	60
la05	201		2	201		< 1	380	553	60	401	553	60
la06	251	372	60	305	372	60	441	912	60	466	953	60
la07	292	319	60	319		18	422	868	60	432	886	60
la08	157	303	60	221	303	60	370	856	60	437	902	60
la09	287	449	60	346	448	60	407	939	60	475	1004	60
la10	262	414	60	330	413	60	443	871	60	465	931	60
la11	255	562	60	317	562	60	448	1186	60	494	1261	60
la12	258	466	60	282	466	60	416	1051	60	452	1099	60
la13	230	507	60	314	507	60	444	1152	60	469	1222	60
la14	269	550	60	319	550	60	443	1160	60	461	1278	60
la15	296	567	60	329	567	60	401	1188	60	433	1277	60
la16	471		< 1	471		1	717		< 1	717	722	60
la17	369		< 1	369		< 1	646		< 1	646		24
la18	420		< 1	420		< 1	663		< 1	663		37
la19	336		< 1	336		< 1	617		< 1	617	691	60
la20	475		< 1	475		< 1	756		< 1	756		23
la21	416		3	416		8	717	995	60	717	1089	60
la22	469		< 1	469		3	619	922	60	619	1151	60

Table B.5.: Results from CP Optimizer and OR-Tools for the Blocking and APP datasets (continued).

	Alternative Process Plans						**Blocking**					
	CP Optimizer			**OR-Tools**			**CP Optimizer**			**OR-Tools**		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
la23	379		4	379		3	640	1037	60	640	1183	60
la24	397		1	397		3	704	962	60	704	1071	60
la25	452		1	452		2	723	917	60	723	1048	60
la26	450	500	60	450	508	60	717	1169	60	717	1614	60
la27	474	593	60	474	597	60	686	1188	60	686	1752	60
la28	581	631	60	581	653	60	756	1170	60	756	1748	60
la29	616		1	616		2	723	1132	60	723	1520	60
la30	616	734	60	616	755	60	726	1192	60	726	1755	60
la31	521	849	60	521	876	60	717	1693	60	717	3687	60
la32	457	845	60	457	865	60	756	1823	60	756	3018	60
la33	594	808	60	594	836	60	723	1642	60	723	—	60
la34	622	804	60	622	869	60	656	1676	60	656	3045	60
la35	471	785	60	471	812	60	647	1693	60	647	3020	60
la36	738		2	738		3	948		< 1	948	1769	60
la37	644		3	644		5	986		2	986	1593	60
la38	758		1	758		3	943		< 1	943	1326	60
la39	585		6	585		4	922		4	922	1509	60
la40	784		< 1	784		3	955		< 1	955	1456	60
mt06	30		< 1	30		< 1	47		< 1	47		< 1
mt10	476		< 1	476		< 1	655		< 1	655	697	60
mt20	259	524	60	321	524	60	387	1127	60	446	1198	60
orb1	492		< 1	492		< 1	695		< 1	695		27
orb2	544		< 1	544		< 1	620		< 1	620		46
orb3	557		< 1	557		< 1	648		< 1	648		51
orb4	556		< 1	556		< 1	753		< 1	753		36
orb5	419		< 1	419		< 1	584		< 1	584	609	60
orb6	503		< 1	503		< 1	715		< 1	715	747	60
orb7	197		< 1	197		< 1	275		< 1	275	302	60
orb8	445		< 1	445		< 1	573		< 1	573		47
orb9	526		< 1	526		< 1	659		< 1	659		17
orb10	518		< 1	518		< 1	681		< 1	681	688	60
						Kacem						
1	8		< 1	8		< 1	11		< 1	11		< 1
2	8		< 1	8		< 1	11		< 1	11		1
3	3		< 1	3		< 1	7		< 1	7		< 1
4	7		< 1	7		< 1	10	12	60	10	12	60

B.4. FJSP — Multivalued Decision Diagrams

In Table B.6, we present the results of DDO for all classic datasets (see Section 6.1.1). The maximum allowed runtime was set to 60 seconds. The lower-bound values as reported by DDO are incorrect due to the time-limit cutoff. Instances which are ‘solved to optimality’ are solved to optimality with respect to their allowed width, but may not have reached the optimal value for that instance.

Table B.6.: DDO results for all classic datasets.

	Restricted						**Relaxed**					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
						Barnes						
mt10c1	1296		< 1	1207		< 1	216	1017	60	219	1006	61
mt10cc	1306		< 1	1207		< 1	216	1003	60	219	997	60
mt10x	1173		< 1	1118		< 1	216	1021	61	261	1008	61
mt10xx	1173		< 1	1058		< 1	216	1025	60	261	1008	60
mt10xxx	1173		< 1	1129		< 1	216	1025	60	261	1016	61
mt10xy	1243		< 1	1166		< 1	216	1027	60	249	995	60
mt10xyz	1060		< 1	1044		< 1	216	936	60	246	907	61
setb4c9	1367		< 1	1305		< 1	750	1138	60	342	1048	60
setb4cc	1298		< 1	1229		< 1	750	1179	60	284	1055	60
setb4x	1359		< 1	1254		< 1	750	1147	60	369	1044	60
setb4xx	1288		< 1	1383		< 1	750	1181	60	369	1073	60
setb4xxx	1205		< 1	1288		< 1	750	1205	60	369	1091	60
setb4xy	1076		< 1	1112		< 1	750	1073	60	300	1033	60
setb4xyz	1134		< 1	1113		< 1	750	1032	60	308	1019	60
seti5c12	1612		< 1	1461		< 1	1140	1349	60	501	1281	60
seti5cc	1401		< 1	1408		< 1	1140	1265	60	501	1218	60
seti5x	1523		< 1	1533		< 1	1140	1363	60	501	1298	60
seti5xx	1581		< 1	1368		< 1	1140	1368	60	501	1297	60
seti5xxx	1570		< 1	1579		< 1	1140	1363	60	501	1314	60
seti5xy	1566		< 1	1502		< 1	1140	1265	60	501	1207	60
seti5xyz	1562		< 1	1521		< 1	1140	1282	60	449	1253	60
						Brandimarte						
Mk01	51		< 1	61		< 1	51		< 1	26	42	61
Mk02	53		< 1	50		< 1	53		< 1	28	32	62
Mk03	302		< 1	293		< 1	150	268	61	77	236	60
Mk04	83		< 1	83		< 1	83		< 1	39	70	60
Mk05	248		< 1	207		< 1	248		< 1	180		1
Mk06	76		< 1	73		< 1	76		< 1	64	66	60
Mk07	234		< 1	209		< 1	100	180	64	46	170	61
Mk08	711		< 1	572		< 1	711		< 1	333	543	60
Mk09	439		< 1	463		< 1	439		< 1	332	382	60
Mk10	409		< 1	380		< 1	409		< 1	321		12
						Dauzère-Paulli						
01a	3202		< 1	2918		< 1	2027	2879	60	1202	2748	60
02a	2806		< 1	2657		< 1	2027	2672	60	1107	2505	60
03a	2846		< 1	2969		< 1	2027	2599	60	1061	2474	60

Table B.6.: DDO results for all classic datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
04a	3104		< 1	2911		< 1	2885		< 1	1339	2708	60
05a	3057		< 1	2917		< 1	2260	2653	60	1310	2440	60
06a	2947		< 1	2523		< 1	2218	2481	60	1256	2459	61
07a	3703		< 1	3316		< 1	2957	3129	60	855	2745	60
08a	3081		< 1	2708		< 1	2818		2	823	2400	60
09a	2672		< 1	2682		< 1	2672		< 1	776	2417	60
10a	3799		< 1	2816		< 1	3453		< 1	1411	2774	60
11a	2704		< 1	2662		< 1	2704		< 1	1133	2336	60
12a	2500		< 1	2471		< 1	2500		< 1	1249	2291	61
13a	3616		< 1	3105		< 1	3616		< 1	1335	2913	61
14a	3119		< 1	2769		< 1	3119		2	1134	2644	61
15a	3026		< 1	2821		< 1	3026		2	1111	2752	61
16a	3282		< 1	3251		< 1	3282		< 1	1493	2909	60
17a	3023		< 1	2676		< 1	3023		2	1450	2468	63
18a	2735		< 1	2954		< 1	2735		2	1500	2586	62
Fattahi												
SFJS1	91		< 1	66		< 1	66		< 1	66		< 1
SFJS2	128		< 1	107		< 1	107		< 1	107		< 1
SFJS3	298		< 1	221		< 1	221		< 1	221		< 1
SFJS4	531		< 1	367		< 1	355		< 1	355		< 1
SFJS5	144		< 1	119		< 1	119		< 1	119		< 1
SFJS6	330		< 1	320		< 1	320		< 1	320		< 1
SFJS7	397		< 1	397		< 1	397		< 1	397		< 1
SFJS8	273		< 1	256		< 1	253		< 1	253		< 1
SFJS9	257		< 1	215		< 1	210		< 1	210		< 1
SFJS10	608		< 1	533		< 1	608		< 1	516		< 1
MFJS1	610		< 1	601		< 1	610		< 1	491		< 1
MFJS2	601		< 1	601		< 1	601		< 1	508		< 1
MFJS3	761		< 1	503		< 1	761		< 1	498		< 1
MFJS4	745		< 1	623		< 1	745		< 1	512	565	61
MFJS5	878		< 1	593		< 1	878		< 1	593		< 1
MFJS6	1102		< 1	995		< 1	1102		< 1	614	647	61
MFJS7	1428		< 1	1199		< 1	1428		< 1	905		14
MFJS8	1562		< 1	1312		< 1	1332		< 1	843	905	61
MFJS9	1806		< 1	1534		< 1	1575		< 1	827	1249	61
MFJS10	1887		< 1	1630		< 1	1887		< 1	854	1446	61
Hurink edata												
abz5	1493		< 1	1349		< 1	1493		< 1	1249		< 1
abz6	1145		< 1	1219		< 1	1145		< 1	656	991	60
abz7	968		< 1	849		< 1	968		< 1	699	763	61
abz8	902		< 1	827		< 1	902		< 1	724	798	60
abz9	959		< 1	904		< 1	959		< 1	747	821	60

Table B.6.: DDO results for all classic datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
car1	8786		< 1	7144		< 1	1305	6844	63	1190	6481	61
car2	7776		< 1	7476		< 1	874	6814	63	1003	6532	61
car3	9480		< 1	8188		< 1	6047	7926	62	2667	7514	61
car4	9213		< 1	9013		< 1	995	8227	63	918	7958	61
car5	8366		< 1	8267		< 1	1477	7632	62	2180	7591	61
car6	9950		< 1	9064		< 1	2154	8421	61	1998	8245	61
car7	6766		< 1	6664		< 1	2288	6393	62	2041	6123	62
car8	9453		< 1	8662		< 1	2022	8352	61	1899	7689	61
la01	843		< 1	761		< 1	605	665	63	381	639	61
la02	875		< 1	813		< 1	596	690	63	375	658	61
la03	734		< 1	649		< 1	359	608	63	220	595	62
la04	817		< 1	735		< 1	254	626	63	179	599	62
la05	678		< 1	655		< 1	257	552	63	218	520	61
la06	1277		< 1	1133		< 1	544	932	61	362	918	61
la07	1044		< 1	913		< 1	613	840	61	424	820	61
la08	1080		< 1	979		< 1	374	935	61	209	915	61
la09	1115		< 1	1040		< 1	525	927	61	353	907	61
la10	1238		< 1	1281		< 1	413	1003	61	216	960	61
la11	1382		< 1	1336		< 1	698	1284	61	409	1220	61
la12	1280		< 1	1184		< 1	500	1096	61	264	1016	61
la13	1280		< 1	1287		< 1	512	1216	61	320	1157	60
la14	1411		< 1	1529		< 1	500	1344	61	276	1272	61
la15	1349		< 1	1284		< 1	595	1262	61	397	1204	61
la16	1160		< 1	1041		< 1	725	1012	60	402	949	61
la17	859		< 1	915		< 1	510	781	60	301	777	60
la18	1064		< 1	1060		< 1	531	923	60	347	904	61
la19	1271		< 1	1018		< 1	526	860	60	292	831	60
la20	1217		< 1	1050		< 1	603	995	60	311	912	61
la21	1325		< 1	1509		< 1	1054	1280	60	387	1200	60
la22	1342		< 1	1126		< 1	767	1095	60	351	1045	60
la23	1288		< 1	1456		< 1	762	1131	60	260	1070	60
la24	1455		< 1	1285		< 1	750	1168	60	294	1055	60
la25	1330		< 1	1430		< 1	749	1262	60	284	1112	60
la26	1690		< 1	1546		< 1	1395	1464	60	562	1408	60
la27	1856		< 1	1841		< 1	1009	1660	60	406	1549	60
la28	1698		< 1	1753		< 1	1013	1555	60	349	1388	60
la29	1645		< 1	1646		< 1	999	1514	60	331	1453	60
la30	1777		< 1	1635		< 1	1012	1628	60	429	1506	60
la31	2119		< 1	2007		< 1	1502	1940	60	590	1912	60
la32	2401		< 1	2401		< 1	1504	2241	60	489	2115	61
la33	2246		< 1	2131		< 1	1497	2039	60	352	1958	60
la34	2206		< 1	2067		< 1	1512	2043	60	574	1959	60
la35	2338		< 1	2222		< 1	1502	2181	60	457	2048	60

Table B.6.: DDO results for all classic datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
la36	1656		< 1	1696		< 1	1535		23	601	1426	60
la37	2048		< 1	1607		< 1	1150	1637	60	458	1537	60
la38	1588		< 1	1626		< 1	1126	1465	60	479	1370	60
la39	1600		< 1	1555		< 1	1140	1465	60	466	1359	60
la40	1461		< 1	1452		< 1	1140	1341	60	464	1258	60
mt06	72		< 1	66		< 1	37	55	67	37	55	73
mt10	1162		< 1	1054		< 1	216	1005	60	255	955	61
mt20	1424		< 1	1337		< 1	202	1281	61	196	1203	61
orb1	1225		< 1	1207		< 1	521	1115	60	307	1053	60
orb2	1168		< 1	1055		< 1	610	935	60	366	941	60
orb3	1169		< 1	1115		< 1	516	1084	60	357	1038	61
orb4	1274		< 1	1384		< 1	508	1149	60	321	1118	60
orb5	1098		< 1	1011		< 1	501	966	60	244	925	61
orb6	1210		< 1	1123		< 1	620	1053	60	323	990	60
orb7	524		< 1	486		< 1	51	422	60	198	413	60
orb8	1215		< 1	1082		< 1	497	943	60	265	929	61
orb9	1254		< 1	1199		< 1	520	1064	60	243	1000	61
orb10	1095		< 1	1251		< 1	502	1042	60	326	974	60
Hurink rdata												
abz5	1310		< 1	1192		< 1	1310		< 1	1110		< 1
abz6	1065		< 1	946		< 1	1065		< 1	628	864	60
abz7	898		< 1	764		< 1	898		< 1	647	662	61
abz8	837		< 1	930		< 1	837		< 1	687		9
abz9	825		< 1	829		< 1	825		< 1	614	698	61
car1	6932		< 1	6596		< 1	758	5771	62	753	5421	61
car2	7315		< 1	7081		< 1	696	6376	64	614	6168	61
car3	7437		< 1	7461		< 1	5924	6429	61	2608	6167	61
car4	9363		< 1	8558		< 1	852	7167	63	753	6888	61
car5	8562		< 1	7463		< 1	1245	6677	61	1134	6334	61
car6	7855		< 1	7258		< 1	1716	6700	61	1334	6299	61
car7	6210		< 1	5961		< 1	1167	4938	61	1129	4550	62
car8	8311		< 1	6809		< 1	1425	6308	61	2008	5989	61
la01	735		< 1	749		< 1	605	627	62	344	605	61
la02	732		< 1	710		< 1	596	611	62	309	590	61
la03	723		< 1	586		< 1	359	546	62	203	519	61
la04	681		< 1	701		< 1	254	585	62	173	528	61
la05	691		< 1	728		< 1	252	507	63	161	501	61
la06	1081		< 1	1142		< 1	544	893	61	278	858	61
la07	1029		< 1	856		< 1	613	836	61	394	787	61
la08	1158		< 1	908		< 1	374	910	61	184	849	61
la09	1148		< 1	971		< 1	525	944	61	319	894	61
la10	1178		< 1	1096		< 1	399	931	61	223	895	61

Table B.6.: DDO results for all classic datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
la11	1366		< 1	1309		< 1	696	1213	62	379	1152	61
la12	1166		< 1	1110		< 1	500	1070	62	188	1015	61
la13	1311		< 1	1263		< 1	512	1162	61	287	1137	61
la14	1369		< 1	1244		< 1	500	1201	61	259	1141	60
la15	1326		< 1	1236		< 1	595	1216	61	363	1168	61
la16	1029		< 1	953		< 1	725	865	60	409	793	60
la17	787		< 1	914		< 1	510	737	60	293	701	60
la18	1127		< 1	1044		< 1	531	742	60	266	730	60
la19	1032		< 1	919		< 1	526	821	60	203	792	60
la20	1187		< 1	994		< 1	603	919	60	264	826	60
la21	1187		< 1	1258		< 1	1054	1071	60	427	993	60
la22	1108		< 1	1201		< 1	767	1021	60	285	932	60
la23	1483		< 1	1114		< 1	758	1058	60	262	985	60
la24	1177		< 1	1232		< 1	750	1044	60	267	1012	60
la25	1307		< 1	1177		< 1	749	1087	60	276	1013	60
la26	1690		< 1	1402		< 1	1394		14	506	1305	60
la27	1554		< 1	1520		< 1	1009	1448	60	403	1370	60
la28	1710		< 1	1678		< 1	1013	1538	60	455	1454	60
la29	1614		< 1	1430		< 1	999	1346	61	333	1291	60
la30	1741		< 1	1566		< 1	1012	1473	60	480	1409	60
la31	2121		< 1	1940		< 1	1502	1893	60	572	1823	60
la32	2336		< 1	2160		< 1	1504	2119	60	476	1993	61
la33	2142		< 1	1941		< 1	1497	1864	60	437	1857	61
la34	2127		< 1	1921		< 1	1512	1947	60	553	1869	61
la35	2199		< 1	2187		< 1	1502	1955	60	503	1876	60
la36	1545		< 1	1697		< 1	1545		< 1	746	1275	60
la37	1656		< 1	1346		< 1	1150	1317	60	452	1230	60
la38	1369		< 1	1507		< 1	1126	1245	60	424	1180	60
la39	1586		< 1	1457		< 1	1140	1323	60	366	1201	60
la40	1396		< 1	1227		< 1	1140	1217	60	367	1148	60
mt06	61		< 1	53		< 1	37	50	62	26	47	62
mt10	1021		< 1	941		< 1	216	840	60	191	759	61
mt20	1379		< 1	1252		< 1	202	1219	61	198	1129	61
orb1	1027		< 1	1053		< 1	521	911	60	194	849	60
orb2	910		< 1	915		< 1	610	798	61	316	757	60
orb3	931		< 1	840		< 1	516	857	61	325	810	60
orb4	1104		< 1	944		< 1	508	879	61	263	823	60
orb5	1179		< 1	853		< 1	501	791	60	234	725	60
orb6	961		< 1	1010		< 1	620	920	60	308	849	60
orb7	415		< 1	387		< 1	45	343	61	205	322	60
orb8	787		< 1	860		< 1	497	723	61	213	683	61
orb9	967		< 1	875		< 1	509	854	60	281	760	60
orb10	1160		< 1	1054		< 1	502	898	60	286	845	60

Table B.6.: DDO results for all classic datasets (continued).

	Restricted						**Relaxed**					
	$w=1$			$w=5$			$w=1$			$w=5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
						Hurink vdata						
abz5	993		< 1	945		< 1	993		< 1	945		< 1
abz6	861		< 1	902		< 1	861		< 1	579	764	60
abz7	773		< 1	711		< 1	773		2	534	618	61
abz8	753		< 1	723		< 1	753		2	545	628	61
abz9	740		< 1	781		< 1	740		2	569	651	61
car1	6644		< 1	6176		< 1	650	5755	62	511	5412	61
car2	6877		< 1	6906		< 1	625	6410	62	461	6080	61
car3	9448		< 1	6746		< 1	5924	6465	62	2393	6276	61
car4	7660		< 1	7805		< 1	852	7010	62	595	6740	61
car5	7051		< 1	7076		< 1	1245	5924	62	1132	5558	61
car6	7469		< 1	7469		< 1	1716	5486	61	843	5486	60
car7	5372		< 1	4800		< 1	796	4367	62	747	4326	61
car8	5811		< 1	6124		< 1	1425	4968	62	1199	4707	61
la01	698		< 1	654		< 1	605	607	62	310	591	61
la02	676		< 1	593		< 1	593		5	254	565	61
la03	667		< 1	603		< 1	359	552	62	204	535	61
la04	713		< 1	726		< 1	254	606	62	133	549	61
la05	625		< 1	616		< 1	252	522	63	194	508	61
la06	1117		< 1	991		< 1	544	916	62	317	903	61
la07	1029		< 1	894		< 1	613	836	62	360	804	60
la08	1083		< 1	977		< 1	374	891	61	222	844	61
la09	1055		< 1	1048		< 1	525	916	62	295	887	61
la10	1098		< 1	1000		< 1	399	939	62	211	874	61
la11	1299		< 1	1324		< 1	696	1170	62	334	1138	61
la12	1145		< 1	1101		< 1	500	1031	62	255	995	61
la13	1255		< 1	1232		< 1	512	1143	62	303	1091	61
la14	1351		< 1	1374		< 1	500	1227	62	242	1159	61
la15	1303		< 1	1226		< 1	595	1186	62	325	1132	61
la16	886		< 1	790		< 1	725	739	60	339	734	60
la17	764		< 1	752		< 1	510	680	61	219	664	60
la18	804		< 1	754		< 1	531	663	61	259	663	60
la19	812		< 1	782		< 1	526	693	61	222	672	60
la20	1066		< 1	821		< 1	603	790	60	180	761	60
la21	1089		< 1	1072		< 1	1039		2	413	975	60
la22	1086		< 1	941		< 1	767	936	61	272	872	60
la23	1241		< 1	1250		< 1	758	1034	61	269	979	61
la24	1275		< 1	1086		< 1	750	1003	61	218	988	60
la25	1243		< 1	1097		< 1	749	1006	61	233	951	61
la26	1436		< 1	1357		< 1	1385		3	498	1249	61
la27	1666		< 1	1549		< 1	1009	1383	61	431	1336	61
la28	1554		< 1	1460		< 1	1013	1458	61	392	1395	60
la29	1584		< 1	1482		< 1	999	1277	61	343	1292	60

Table B.6.: DDO results for all classic datasets (continued).

<table>
<tr><th></th><th colspan="6">Restricted</th><th colspan="6">Relaxed</th></tr>
<tr><th></th><th colspan="3">$w = 1$</th><th colspan="3">$w = 5$</th><th colspan="3">$w = 1$</th><th colspan="3">$w = 5$</th></tr>
<tr><th>Name</th><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th></tr>
<tr><td>la30</td><td colspan="2">1623</td><td>< 1</td><td colspan="2">1538</td><td>< 1</td><td>1012</td><td>1425</td><td>61</td><td>323</td><td>1407</td><td>60</td></tr>
<tr><td>la31</td><td colspan="2">2022</td><td>< 1</td><td colspan="2">2130</td><td>< 1</td><td>1502</td><td>1901</td><td>61</td><td>555</td><td>1777</td><td>62</td></tr>
<tr><td>la32</td><td colspan="2">2266</td><td>< 1</td><td colspan="2">2162</td><td>< 1</td><td>1504</td><td>2094</td><td>61</td><td>464</td><td>2022</td><td>61</td></tr>
<tr><td>la33</td><td colspan="2">1974</td><td>< 1</td><td colspan="2">1850</td><td>< 1</td><td>1497</td><td>1849</td><td>61</td><td>537</td><td>1775</td><td>60</td></tr>
<tr><td>la34</td><td colspan="2">2067</td><td>< 1</td><td colspan="2">1940</td><td>< 1</td><td>1512</td><td>1887</td><td>62</td><td>540</td><td>1814</td><td>61</td></tr>
<tr><td>la35</td><td colspan="2">2043</td><td>< 1</td><td colspan="2">1983</td><td>< 1</td><td>1502</td><td>1901</td><td>60</td><td>477</td><td>1828</td><td>60</td></tr>
<tr><td>la36</td><td colspan="2">1263</td><td>< 1</td><td colspan="2">1217</td><td>< 1</td><td colspan="2">1263</td><td>1</td><td>396</td><td>1044</td><td>61</td></tr>
<tr><td>la37</td><td colspan="2">1301</td><td>< 1</td><td colspan="2">1138</td><td>< 1</td><td>1150</td><td>1183</td><td>63</td><td>318</td><td>1098</td><td>61</td></tr>
<tr><td>la38</td><td colspan="2">1332</td><td>< 1</td><td colspan="2">1143</td><td>< 1</td><td colspan="2">1124</td><td>3</td><td>242</td><td>1043</td><td>61</td></tr>
<tr><td>la39</td><td colspan="2">1610</td><td>< 1</td><td colspan="2">1078</td><td>< 1</td><td colspan="2">1129</td><td>6</td><td>356</td><td>1008</td><td>60</td></tr>
<tr><td>la40</td><td colspan="2">1502</td><td>< 1</td><td colspan="2">1004</td><td>< 1</td><td colspan="2">1060</td><td>3</td><td>367</td><td>1004</td><td>61</td></tr>
<tr><td>mt06</td><td colspan="2">54</td><td>< 1</td><td colspan="2">60</td><td>< 1</td><td>37</td><td>47</td><td>62</td><td>17</td><td>47</td><td>61</td></tr>
<tr><td>mt10</td><td colspan="2">748</td><td>< 1</td><td colspan="2">795</td><td>< 1</td><td>216</td><td>693</td><td>60</td><td>187</td><td>684</td><td>60</td></tr>
<tr><td>mt20</td><td colspan="2">1272</td><td>< 1</td><td colspan="2">1215</td><td>< 1</td><td>202</td><td>1178</td><td>62</td><td>188</td><td>1100</td><td>61</td></tr>
<tr><td>orb1</td><td colspan="2">834</td><td>< 1</td><td colspan="2">813</td><td>< 1</td><td>521</td><td>759</td><td>61</td><td>261</td><td>721</td><td>60</td></tr>
<tr><td>orb2</td><td colspan="2">933</td><td>< 1</td><td colspan="2">899</td><td>< 1</td><td>610</td><td>724</td><td>61</td><td>290</td><td>682</td><td>60</td></tr>
<tr><td>orb3</td><td colspan="2">920</td><td>< 1</td><td colspan="2">823</td><td>< 1</td><td>516</td><td>696</td><td>61</td><td>304</td><td>662</td><td>60</td></tr>
<tr><td>orb4</td><td colspan="2">919</td><td>< 1</td><td colspan="2">948</td><td>< 1</td><td>508</td><td>771</td><td>61</td><td>235</td><td>753</td><td>60</td></tr>
<tr><td>orb5</td><td colspan="2">890</td><td>< 1</td><td colspan="2">706</td><td>< 1</td><td>501</td><td>648</td><td>60</td><td>231</td><td>602</td><td>60</td></tr>
<tr><td>orb6</td><td colspan="2">798</td><td>< 1</td><td colspan="2">829</td><td>< 1</td><td>620</td><td>718</td><td>61</td><td>229</td><td>715</td><td>60</td></tr>
<tr><td>orb7</td><td colspan="2">449</td><td>< 1</td><td colspan="2">354</td><td>< 1</td><td>45</td><td>337</td><td>61</td><td>112</td><td>312</td><td>60</td></tr>
<tr><td>orb8</td><td colspan="2">821</td><td>< 1</td><td colspan="2">814</td><td>< 1</td><td>497</td><td>614</td><td>61</td><td>162</td><td>574</td><td>60</td></tr>
<tr><td>orb9</td><td colspan="2">916</td><td>< 1</td><td colspan="2">731</td><td>< 1</td><td>509</td><td>698</td><td>60</td><td>253</td><td>702</td><td>60</td></tr>
<tr><td>orb10</td><td colspan="2">1208</td><td>< 1</td><td colspan="2">809</td><td>< 1</td><td>502</td><td>739</td><td>61</td><td>209</td><td>712</td><td>60</td></tr>
<tr><th colspan="13">Kacem</th></tr>
<tr><td>1</td><td colspan="2">12</td><td>< 1</td><td colspan="2">12</td><td>< 1</td><td colspan="2">11</td><td>< 1</td><td colspan="2">11</td><td>16</td></tr>
<tr><td>2</td><td colspan="2">16</td><td>< 1</td><td colspan="2">16</td><td>< 1</td><td colspan="2">16</td><td>< 1</td><td colspan="2">14</td><td>< 1</td></tr>
<tr><td>3</td><td colspan="2">8</td><td>< 1</td><td colspan="2">8</td><td>< 1</td><td colspan="2">8</td><td>< 1</td><td colspan="2">8</td><td>< 1</td></tr>
<tr><td>4</td><td colspan="2">17</td><td>< 1</td><td colspan="2">14</td><td>< 1</td><td colspan="2">17</td><td>< 1</td><td colspan="2">14</td><td>< 1</td></tr>
</table>

In Table B.7, we present the results of both CP solvers for the modern datasets (see Section 6.1.1). The maximum allowed runtime was set to 60 seconds. The lower-bound values as reported by DDO are incorrect due to the time-limit cutoff. Instances which are 'solved to optimality' are solved to optimality with respect to their allowed width, but may not have reached the optimal value for that instance.

Table B.7.: Results of DDO for modern datasets.

	Restricted						Relaxed					
	$w=1$			$w=5$			$w=1$			$w=5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
						Behnke						
1	103		< 1	97		< 1	103		< 1	99		< 1
2	111		< 1	106		< 1	111		< 1	99		< 1
3	106		< 1	105		< 1	106		< 1	105		< 1
4	106		< 1	109		< 1	106		< 1	113		< 1
5	100		< 1	105		< 1	100		< 1	105		< 1
6	147		< 1	143		< 1	147		< 1	138		< 1
7	152		< 1	140		< 1	152		< 1	141		< 1
8	146		< 1	141		< 1	146		< 1	143		< 1
9	134		< 1	146		< 1	134		< 1	143		< 1
10	150		< 1	152		< 1	150		< 1	139		< 1
11	261		< 1	258		< 1	261		2	252		4
12	249		< 1	240		< 1	249		2	246		4
13	266		< 1	268		< 1	266		2	265		4
14	271		< 1	259		< 1	271		2	257		45
15	268		< 1	262		< 1	268		2	261		4
16	452		< 1	450		5	452		13	450		23
17	447		< 1	446		5	447		12	433		23
18	439		< 1	434		5	439		12	442		22
19	436		< 1	439		5	436		13	436		23
20	445		< 1	435		5	445		12	424		22
21	103		< 1	106		< 1	103		< 1	103		< 1
22	104		< 1	97		< 1	104		< 1	93		2
23	104		< 1	90		< 1	104		< 1	90		< 1
24	98		< 1	99		< 1	98		< 1	99		< 1
25	104		< 1	103		< 1	104		< 1	103		< 1
26	145		< 1	129		< 1	145		< 1	127		1
27	138		< 1	140		< 1	138		< 1	137		23
28	127		< 1	130		< 1	127		< 1	128		1
29	140		< 1	137		< 1	140		< 1	133		2
30	137		< 1	138		< 1	137		< 1	124	134	61
31	257		< 1	252		2	257		5	249		8
32	248		< 1	246		2	248		4	239		58
33	251		< 1	247		2	251		5	219	242	61
34	243		< 1	242		2	243		4	235		8
35	232		< 1	235		2	232		5	225		60
36	417		1	415		7	417		27	412	405	61
37	411		1	421		7	411		26	426		43
38	427		1	421		7	427		27	423		45
39	423		1	418		7	423		28	419		44
40	454		1	445		7	454		26	448		43
41	96		< 1	96		< 1	96		< 1	96		1
42	101		< 1	101		< 1	101		< 1	93	96	61

Table B.7.: Results of DDO for modern datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
43	96		< 1	99		< 1	96		< 1	99		< 1
44	95		< 1	95		< 1	95		< 1	95		< 1
45	102		< 1	92		< 1	102		< 1	92		1
46	127		< 1	127		< 1	127		< 1	131		2
47	138		< 1	138		< 1	138		< 1	133		3
48	132		< 1	141		< 1	132		< 1	132		2
49	125		< 1	119		< 1	125		< 1	124		2
50	137		< 1	135		< 1	137		1	137		3
51	247		< 1	247		3	247		8	246		14
52	234		< 1	232		3	234		8	238		14
53	243		< 1	233		3	243		8	232		14
54	247		< 1	244		3	247		8	241		13
55	244		< 1	239		3	244		8	235		13
56	421		2	413		11	421		46	414		64
57	423		2	422		11	423		46	417		64
58	429		2	421		11	429		48	424		64
59	432		2	422		11	432		45	420		64
60	435		2	435		10	435		43	427		64
Naderi												
1	1365		< 1	1234		< 1	1365		< 1	575	1190	61
2	1473		< 1	1442		< 1	1473		< 1	618	1331	60
3	1367		< 1	1323		< 1	1367		< 1	570	1184	61
4	1388		< 1	1363		< 1	1388		< 1	642	1293	61
5	2935		< 1	2650		< 1	2935		2	911	2663	62
6	3487		< 1	3174		< 1	3487		2	1208	2942	62
7	2882		< 1	2818		< 1	2882		4	896	2511	62
8	3049		< 1	2897		< 1	3049		4	1318	2772	63
9	921		< 1	888		< 1	921		< 1	455	868	61
10	1041		< 1	1057		< 1	1041		< 1	493	879	61
11	942		< 1	840		< 1	942		2	553	827	61
12	1069		< 1	994		< 1	1069		1	486	972	64
13	1859		< 1	2043		< 1	1859		3	888	1736	62
14	2504		< 1	2084		< 1	2504		3	1092	2036	66
15	2126		< 1	2099		1	2126		7	903	1987	67
16	2733		< 1	2705		1	2733		6	1033	2570	69
17	839		< 1	868		< 1	839		1	446	773	61
18	948		< 1	960		< 1	948		1	493	827	62
19	834		< 1	804		< 1	834		2	528	696	65
20	974		< 1	991		< 1	974		2	475	858	65
21	1692		< 1	1936		1	1692		5	930	1632	63
22	2250		< 1	2118		1	2250		5	1109	1971	64
23	1534		< 1	1671		2	1534		8	882	1436	70

Table B.7.: Results of DDO for modern datasets (continued).

<table>
<tr><th></th><th colspan="6">Restricted</th><th colspan="6">Relaxed</th></tr>
<tr><th></th><th colspan="3">$w = 1$</th><th colspan="3">$w = 5$</th><th colspan="3">$w = 1$</th><th colspan="3">$w = 5$</th></tr>
<tr><th>Name</th><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th></tr>
<tr><td>24</td><td colspan="2">1857</td><td>< 1</td><td colspan="2">1822</td><td>2</td><td colspan="2">1857</td><td>8</td><td>931</td><td>1782</td><td>71</td></tr>
<tr><td>25</td><td colspan="2">2265</td><td>< 1</td><td colspan="2">2168</td><td>< 1</td><td colspan="2">2265</td><td>2</td><td>757</td><td>2035</td><td>61</td></tr>
<tr><td>26</td><td colspan="2">2291</td><td>< 1</td><td colspan="2">2212</td><td>< 1</td><td colspan="2">2291</td><td>2</td><td>869</td><td>2070</td><td>61</td></tr>
<tr><td>27</td><td colspan="2">2307</td><td>< 1</td><td colspan="2">2131</td><td>1</td><td colspan="2">2307</td><td>3</td><td>731</td><td>2158</td><td>62</td></tr>
<tr><td>28</td><td colspan="2">2392</td><td>< 1</td><td colspan="2">2212</td><td>1</td><td colspan="2">2392</td><td>3</td><td>846</td><td>2224</td><td>63</td></tr>
<tr><td>29</td><td colspan="2">4597</td><td>< 1</td><td colspan="2">4591</td><td>2</td><td colspan="2">4597</td><td>7</td><td>1498</td><td>4227</td><td>70</td></tr>
<tr><td>30</td><td colspan="2">4909</td><td>< 1</td><td colspan="2">5046</td><td>2</td><td colspan="2">4909</td><td>8</td><td>1860</td><td>4703</td><td>66</td></tr>
<tr><td>31</td><td colspan="2">4354</td><td>< 1</td><td colspan="2">4095</td><td>2</td><td colspan="2">4354</td><td>13</td><td>1411</td><td>3940</td><td>64</td></tr>
<tr><td>32</td><td colspan="2">5437</td><td>< 1</td><td colspan="2">5106</td><td>3</td><td colspan="2">5437</td><td>13</td><td>1456</td><td>4946</td><td>63</td></tr>
<tr><td>33</td><td colspan="2">1497</td><td>< 1</td><td colspan="2">1468</td><td>< 1</td><td colspan="2">1497</td><td>3</td><td>575</td><td>1287</td><td>61</td></tr>
<tr><td>34</td><td colspan="2">1849</td><td>< 1</td><td colspan="2">1668</td><td>1</td><td colspan="2">1849</td><td>3</td><td>913</td><td>1583</td><td>61</td></tr>
<tr><td>35</td><td colspan="2">1637</td><td>< 1</td><td colspan="2">1501</td><td>2</td><td colspan="2">1637</td><td>6</td><td>753</td><td>1430</td><td>65</td></tr>
<tr><td>36</td><td colspan="2">1803</td><td>< 1</td><td colspan="2">1766</td><td>2</td><td colspan="2">1803</td><td>6</td><td>899</td><td>1651</td><td>65</td></tr>
<tr><td>37</td><td colspan="2">3073</td><td>< 1</td><td colspan="2">2979</td><td>2</td><td colspan="2">3073</td><td>12</td><td>1447</td><td>2996</td><td>69</td></tr>
<tr><td>38</td><td colspan="2">4000</td><td>< 1</td><td colspan="2">3389</td><td>2</td><td colspan="2">4000</td><td>12</td><td>1558</td><td>3424</td><td>67</td></tr>
<tr><td>39</td><td colspan="2">2889</td><td>< 1</td><td colspan="2">2987</td><td>4</td><td colspan="2">2889</td><td>21</td><td>989</td><td>3035</td><td>67</td></tr>
<tr><td>40</td><td colspan="2">3508</td><td>< 1</td><td colspan="2">3386</td><td>4</td><td colspan="2">3508</td><td>20</td><td>1464</td><td>3303</td><td>60</td></tr>
<tr><td>41</td><td colspan="2">1403</td><td>< 1</td><td colspan="2">1305</td><td>2</td><td colspan="2">1403</td><td>4</td><td>777</td><td>1167</td><td>61</td></tr>
<tr><td>42</td><td colspan="2">1379</td><td>< 1</td><td colspan="2">1325</td><td>2</td><td colspan="2">1379</td><td>4</td><td>689</td><td>1307</td><td>61</td></tr>
<tr><td>43</td><td colspan="2">1117</td><td>< 1</td><td colspan="2">1113</td><td>3</td><td colspan="2">1117</td><td>9</td><td>646</td><td>1023</td><td>69</td></tr>
<tr><td>44</td><td colspan="2">1243</td><td>< 1</td><td colspan="2">1189</td><td>3</td><td colspan="2">1243</td><td>9</td><td>666</td><td>1186</td><td>68</td></tr>
<tr><td>45</td><td colspan="2">2714</td><td>< 1</td><td colspan="2">2508</td><td>4</td><td colspan="2">2714</td><td>18</td><td>1138</td><td>2478</td><td>62</td></tr>
<tr><td>46</td><td colspan="2">2708</td><td>< 1</td><td colspan="2">2798</td><td>4</td><td colspan="2">2708</td><td>17</td><td>1443</td><td>2527</td><td>64</td></tr>
<tr><td>47</td><td colspan="2">2300</td><td>1</td><td colspan="2">2319</td><td>6</td><td colspan="2">2300</td><td>31</td><td colspan="2">2343</td><td>68</td></tr>
<tr><td>48</td><td colspan="2">2718</td><td>1</td><td colspan="2">2655</td><td>6</td><td colspan="2">2718</td><td>27</td><td>1479</td><td>2676</td><td>60</td></tr>
<tr><td>49</td><td colspan="2">3132</td><td>< 1</td><td colspan="2">2981</td><td>2</td><td colspan="2">3132</td><td>5</td><td>1056</td><td>2832</td><td>64</td></tr>
<tr><td>50</td><td colspan="2">3479</td><td>< 1</td><td colspan="2">3281</td><td>2</td><td colspan="2">3479</td><td>5</td><td>1185</td><td>3031</td><td>63</td></tr>
<tr><td>51</td><td colspan="2">2885</td><td>< 1</td><td colspan="2">2843</td><td>3</td><td colspan="2">2885</td><td>7</td><td>1070</td><td>2602</td><td>69</td></tr>
<tr><td>52</td><td colspan="2">3176</td><td>< 1</td><td colspan="2">3208</td><td>3</td><td colspan="2">3176</td><td>8</td><td>1213</td><td>3027</td><td>66</td></tr>
<tr><td>53</td><td colspan="2">6387</td><td>< 1</td><td colspan="2">6103</td><td>3</td><td colspan="2">6387</td><td>18</td><td>1863</td><td>5833</td><td>64</td></tr>
<tr><td>54</td><td colspan="2">7027</td><td>< 1</td><td colspan="2">6982</td><td>3</td><td colspan="2">7027</td><td>17</td><td>2103</td><td>6439</td><td>61</td></tr>
<tr><td>55</td><td colspan="2">6121</td><td>1</td><td colspan="2">5657</td><td>6</td><td colspan="2">6121</td><td>33</td><td colspan="2">5641</td><td>70</td></tr>
<tr><td>56</td><td colspan="2">6703</td><td>1</td><td colspan="2">6501</td><td>6</td><td colspan="2">6703</td><td>32</td><td colspan="2">6600</td><td>70</td></tr>
<tr><td>57</td><td colspan="2">2156</td><td>< 1</td><td colspan="2">1925</td><td>2</td><td colspan="2">2156</td><td>7</td><td>964</td><td>1888</td><td>68</td></tr>
<tr><td>58</td><td colspan="2">2491</td><td>< 1</td><td colspan="2">2326</td><td>2</td><td colspan="2">2491</td><td>7</td><td>1016</td><td>2170</td><td>69</td></tr>
<tr><td>59</td><td colspan="2">2044</td><td>< 1</td><td colspan="2">1885</td><td>4</td><td colspan="2">2044</td><td>14</td><td>908</td><td>1862</td><td>63</td></tr>
<tr><td>60</td><td colspan="2">2531</td><td>< 1</td><td colspan="2">2392</td><td>5</td><td colspan="2">2531</td><td>15</td><td>1263</td><td>2277</td><td>63</td></tr>
<tr><td>61</td><td colspan="2">4697</td><td>< 1</td><td colspan="2">4336</td><td>5</td><td colspan="2">4697</td><td>31</td><td colspan="2">4362</td><td>72</td></tr>
<tr><td>62</td><td colspan="2">4719</td><td>1</td><td colspan="2">4384</td><td>6</td><td colspan="2">4719</td><td>31</td><td colspan="2">4331</td><td>71</td></tr>
<tr><td>63</td><td colspan="2">3916</td><td>2</td><td colspan="2">3878</td><td>9</td><td colspan="2">3916</td><td>50</td><td colspan="2">3748</td><td>69</td></tr>
<tr><td>64</td><td colspan="2">4687</td><td>2</td><td colspan="2">4285</td><td>10</td><td colspan="2">4687</td><td>51</td><td colspan="2">4318</td><td>68</td></tr>
<tr><td>65</td><td colspan="2">1670</td><td>< 1</td><td colspan="2">1640</td><td>4</td><td colspan="2">1670</td><td>10</td><td>850</td><td>1502</td><td>68</td></tr>
<tr><td>66</td><td colspan="2">1978</td><td>< 1</td><td colspan="2">1836</td><td>4</td><td colspan="2">1978</td><td>11</td><td>864</td><td>1783</td><td>66</td></tr>
</table>

Table B.7.: Results of DDO for modern datasets (continued).

<table>
<tr><th rowspan="3">Name</th><th colspan="6">Restricted</th><th colspan="6">Relaxed</th></tr>
<tr><th colspan="3">$w = 1$</th><th colspan="3">$w = 5$</th><th colspan="3">$w = 1$</th><th colspan="3">$w = 5$</th></tr>
<tr><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th></tr>
<tr><td>67</td><td colspan="2">1490</td><td>1</td><td colspan="2">1546</td><td>7</td><td colspan="2">1490</td><td>20</td><td>798</td><td>1497</td><td>61</td></tr>
<tr><td>68</td><td colspan="2">1854</td><td>1</td><td colspan="2">1763</td><td>7</td><td colspan="2">1854</td><td>21</td><td>952</td><td>1844</td><td>60</td></tr>
<tr><td>69</td><td colspan="2">3566</td><td>1</td><td colspan="2">3498</td><td>7</td><td colspan="2">3566</td><td>44</td><td colspan="2">3418</td><td>69</td></tr>
<tr><td>70</td><td colspan="2">3595</td><td>1</td><td colspan="2">3396</td><td>8</td><td colspan="2">3595</td><td>38</td><td colspan="2">3602</td><td>69</td></tr>
<tr><td>71</td><td colspan="2">3022</td><td>2</td><td colspan="2">2929</td><td>13</td><td colspan="2">3022</td><td>69</td><td colspan="2">2933</td><td>66</td></tr>
<tr><td>72</td><td colspan="2">3623</td><td>3</td><td colspan="2">3477</td><td>14</td><td colspan="2">3623</td><td>68</td><td colspan="2">3496</td><td>66</td></tr>
<tr><td>73</td><td colspan="2">4358</td><td>< 1</td><td colspan="2">4173</td><td>4</td><td colspan="2">4358</td><td>13</td><td>1426</td><td>4109</td><td>62</td></tr>
<tr><td>74</td><td colspan="2">4474</td><td>< 1</td><td colspan="2">4367</td><td>5</td><td colspan="2">4474</td><td>12</td><td>1529</td><td>4231</td><td>64</td></tr>
<tr><td>75</td><td colspan="2">4013</td><td>1</td><td colspan="2">3711</td><td>8</td><td colspan="2">4013</td><td>21</td><td>1255</td><td>3765</td><td>68</td></tr>
<tr><td>76</td><td colspan="2">4512</td><td>1</td><td colspan="2">4303</td><td>8</td><td colspan="2">4512</td><td>21</td><td colspan="2">4380</td><td>68</td></tr>
<tr><td>77</td><td colspan="2">8610</td><td>2</td><td colspan="2">8447</td><td>8</td><td colspan="2">8610</td><td>48</td><td colspan="2">8254</td><td>69</td></tr>
<tr><td>78</td><td colspan="2">9160</td><td>2</td><td colspan="2">8901</td><td>9</td><td colspan="2">9160</td><td>47</td><td colspan="2">8754</td><td>69</td></tr>
<tr><td>79</td><td colspan="2">8497</td><td>3</td><td colspan="2">8340</td><td>15</td><td colspan="2">8497</td><td>70</td><td colspan="2">8204</td><td>66</td></tr>
<tr><td>80</td><td colspan="2">9449</td><td>3</td><td colspan="2">8970</td><td>16</td><td colspan="2">9449</td><td>70</td><td colspan="2">8964</td><td>67</td></tr>
<tr><td>81</td><td colspan="2">2751</td><td>1</td><td colspan="2">2748</td><td>6</td><td colspan="2">2751</td><td>19</td><td>1560</td><td>2665</td><td>61</td></tr>
<tr><td>82</td><td colspan="2">3026</td><td>1</td><td colspan="2">3047</td><td>6</td><td colspan="2">3026</td><td>18</td><td>1185</td><td>3002</td><td>62</td></tr>
<tr><td>83</td><td colspan="2">2687</td><td>2</td><td colspan="2">2558</td><td>12</td><td colspan="2">2687</td><td>35</td><td colspan="2">2619</td><td>67</td></tr>
<tr><td>84</td><td colspan="2">3202</td><td>2</td><td colspan="2">3103</td><td>12</td><td colspan="2">3202</td><td>36</td><td colspan="2">3059</td><td>65</td></tr>
<tr><td>85</td><td colspan="2">5871</td><td>3</td><td colspan="2">5691</td><td>13</td><td colspan="2">5871</td><td>70</td><td colspan="2">5922</td><td>67</td></tr>
<tr><td>86</td><td colspan="2">6688</td><td>3</td><td colspan="2">6471</td><td>15</td><td colspan="2">6688</td><td>70</td><td colspan="2">6309</td><td>67</td></tr>
<tr><td>87</td><td colspan="2">6058</td><td>4</td><td colspan="2">5850</td><td>24</td><td colspan="2">6058</td><td>69</td><td colspan="2">5711</td><td>66</td></tr>
<tr><td>88</td><td colspan="2">6299</td><td>5</td><td colspan="2">6052</td><td>24</td><td colspan="2">6299</td><td>70</td><td colspan="2">6094</td><td>63</td></tr>
<tr><td>89</td><td colspan="2">2204</td><td>2</td><td colspan="2">2176</td><td>10</td><td colspan="2">2204</td><td>27</td><td colspan="2">2148</td><td>66</td></tr>
<tr><td>90</td><td colspan="2">2243</td><td>2</td><td colspan="2">2069</td><td>9</td><td colspan="2">2243</td><td>24</td><td>1366</td><td>2236</td><td>60</td></tr>
<tr><td>91</td><td colspan="2">2129</td><td>3</td><td colspan="2">2106</td><td>17</td><td colspan="2">2129</td><td>53</td><td colspan="2">2111</td><td>63</td></tr>
<tr><td>92</td><td colspan="2">2479</td><td>3</td><td colspan="2">2317</td><td>17</td><td colspan="2">2479</td><td>52</td><td colspan="2">2419</td><td>64</td></tr>
<tr><td>93</td><td colspan="2">4333</td><td>4</td><td colspan="2">4300</td><td>19</td><td colspan="2">4333</td><td>68</td><td colspan="2">4351</td><td>65</td></tr>
<tr><td>94</td><td colspan="2">4989</td><td>4</td><td colspan="2">4733</td><td>20</td><td colspan="2">4989</td><td>68</td><td colspan="2">4894</td><td>65</td></tr>
<tr><td>95</td><td colspan="2">4302</td><td>7</td><td colspan="2">4121</td><td>37</td><td colspan="2">4302</td><td>66</td><td colspan="2">4192</td><td>61</td></tr>
<tr><td>96</td><td colspan="2">4950</td><td>7</td><td colspan="2">4828</td><td>36</td><td colspan="2">4950</td><td>66</td><td colspan="2">4978</td><td>61</td></tr>
</table>

B.5. SDST — Multivalued Decision Diagrams

In Table B.8, we present the results of DDO for all SDST datasets (see Section 6.1.2). The maximum allowed runtime was set to 60 seconds: any instance for which this runtime is reached is not solved to optimality.

Table B.8.: Results of DDO for SDST datasets.

	Restricted						**Relaxed**					
	$w = 1$			$w = 100$			$w = 1$			$w = 100$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
							Fattahi					
SFJS1	99		< 1	70		< 1	70		< 1	70		< 1
SFJS2	138		< 1	112		< 1	112		< 1	112		< 1
SFJS3	324		< 1	233		< 1	233		< 1	233		< 1
SFJS4	562		< 1	374		< 1	389		< 1	374		< 1
SFJS5	151		< 1	126		< 1	126		< 1	126		< 1
SFJS6	337		< 1	334		< 1	334		< 1	334		< 1
SFJS7	397		< 1	397		< 1	397		< 1	397		< 1
SFJS8	336		< 1	262		< 1	262		< 1	262		< 1
SFJS9	259		< 1	220		< 1	221		< 1	220		< 1
SFJS10	633		< 1	541		< 1	633		< 1	541		< 1
MFJS1	686		< 1	522		< 1	686		< 1	522		< 1
MFJS2	627		< 1	485		< 1	627		< 1	485		< 1
MFJS3	698		< 1	544		< 1	698		< 1	544		< 1
MFJS4	775		< 1	655		< 1	775		< 1	655		< 1
MFJS5	680		< 1	621		< 1	680		< 1	621		< 1
MFJS6	1148		< 1	744		< 1	1148		< 1	744		< 1
MFJS7	1530		< 1	1077		< 1	1530		< 1	926	990	62
MFJS8	1646		< 1	1121		< 1	1646		< 1	973		1
MFJS9	1811		< 1	1529		< 1	1811		< 1	1024	1297	62
MFJS10	1912		< 1	1594		< 1	1912		< 1	1156	1483	62
							Hurink edata					
la21	843		< 1	695		< 1	605	665	62	377	623	65
la22	875		< 1	702		< 1	596	690	63	367	655	65
la23	734		< 1	664		< 1	359	607	63	226	577	69
la24	817		< 1	663		< 1	254	620	63	206	599	68
la25	678		< 1	586		< 1	257	553	63	243	518	64
la26	1277		< 1	980		< 1	544	938	61	363	870	62
la27	1044		< 1	835		< 1	613	846	61	396	810	61
la28	1080		< 1	995		< 1	374	935	61	229	883	62
la29	1115		< 1	952		< 1	525	927	61	346	899	61
la30	1238		< 1	996		< 1	413	1001	61	290	938	61
la31	1382		< 1	1262		< 1	698	1284	61	404	1181	62
la32	1280		< 1	1015		< 1	500	1070	61	266	1003	61
la33	1280		< 1	1186		< 1	512	1216	61	359	1144	61
la34	1411		< 1	1393		< 1	500	1344	61	275	1213	61
la35	1349		< 1	1252		< 1	595	1262	61	346	1179	61
la36	1160		< 1	982		< 1	725	1013	60	429	917	61
la37	859		< 1	846		< 1	510	782	60	306	760	62
la38	1064		< 1	963		< 1	531	923	60	358	913	61
la39	1271		< 1	962		< 1	526	860	60	292	801	61

Table B.8.: Results of DDO for SDST datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 100$			$w = 1$			$w = 100$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
la40	1217		< 1	1025		< 1	603	995	60	381	913	61

B.6. Blocking — Multivalued Decision Diagrams

In Table B.9, we present the results of DDO for all Blocking datasets (see Section 6.1.3). The maximum allowed runtime was set to 60 seconds: any instance for which this runtime is reached is not solved to optimality.

Table B.9.: Results of DDO for Blocking datasets.

	Restricted						Relaxed					
	$w = 1$			$w = 100$			$w = 1$			$w = 100$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
						Barnes						
mt10c1	1178	2378	61	1332	2277	63			< 1			< 1
mt10cc	1029	2380	61	1083	2264	62			< 1			< 1
mt10x	1568	2251	61	2198		50			< 1			< 1
mt10xx	1243	2351	61	1247	2266	62			< 1			< 1
mt10xxx	1092	2513	61	1101	2269	62			< 1			< 1
mt10xy	931	2432	61	960	2214	62			< 1			< 1
mt10xyz	588		61	601	2158	61			< 1			< 1
setb4c9	750		60	649		61			< 1			< 1
setb4cc	750		61	510		61			< 1			< 1
setb4x	750		60	576		61			< 1			< 1
setb4xx	750		60	622		61			< 1			< 1
setb4xxx	750		60	611		60			< 1			< 1
setb4xy	750		60	441		60			< 1			< 1
setb4xyz	750		60	489		60			< 1			< 1
seti5c12	1352		60	951		60			< 1			< 1
seti5cc	1352		60	725		60			< 1			< 1
seti5x	1352		60	863		60			< 1			< 1
seti5xx	1352		60	837		60			< 1			< 1
seti5xxx	1352		60	756		60			< 1			< 1
seti5xy	1352		60	718		60			< 1			< 1
seti5xyz	1352		60	717		60			< 1			< 1
						Brandimarte						
Mk01	54	68	62	33	61	64			< 1			< 1
Mk02	57	68	63	39	53	63			< 1			< 1
Mk03	150		62	124		61			< 1			< 1
Mk04	90		61	55		61			< 1			< 1

Table B.9.: Results of DDO for Blocking datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 100$			$w = 1$			$w = 100$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
Mk05	525		61	211	412	61			< 1			< 1
Mk06	149		61	72	398	61			< 1			< 1
Mk07	100		62	76	349	61			< 1			< 1
Mk08	1120		60	375		61			< 1			< 1
Mk09	1195		61	453		61			< 1			< 1
Mk10	1195		61	342		61			< 1			< 1
						Dauzère-Paulli						
01a	3402		60	8794		40			< 1			< 1
02a	3351		60	4217		61			< 1			< 1
03a	2223		60	2192		61			< 1			< 1
04a	3570		60	8716		38			< 1			< 1
05a	3557		61	4313		61			< 1			< 1
06a	3534		60	2207		61			< 1			< 1
07a	3253		60	2599		60			< 1			< 1
08a	2953		61	1733		60			< 1			< 1
09a	2939		61	1717		61			< 1			< 1
10a	4701		60	2868		61			< 1			< 1
11a	4409		60	1780		60			< 1			< 1
12a	3243		61	1594		60			< 1			< 1
13a	3861		60	1188		60			< 1			< 1
14a	3861		61	797		61			< 1			< 1
15a	3861		61	1315		61			< 1			< 1
16a	5406		60	2659		60			< 1			< 1
17a	4250		61	1676		61			< 1			< 1
18a	4249		61	1140		61			< 1			< 1
						Fattahi						
SFJS1	66		< 1	66		< 1	91		< 1	66		< 1
SFJS2	107		< 1	107		< 1	128		< 1	107		< 1
SFJS3	261		< 1	256		< 1	298		< 1	256		< 1
SFJS4	396		< 1	396		< 1	396		< 1	396		< 1
SFJS5	128		< 1	128		< 1	128		< 1	128		< 1
SFJS6	320		< 1	320		< 1	320		< 1	320		< 1
SFJS7	397		< 1	397		< 1	397		< 1	397		< 1
SFJS8	253		< 1	253		< 1	345		< 1	253		< 1
SFJS9	220		< 1	215		< 1	317		< 1	275		< 1
SFJS10	555		< 1	533		< 1	555		< 1	533		< 1
MFJS1	551		< 1	601		< 1	551		< 1	601		< 1
MFJS2	677		< 1	504		< 1	677		< 1	610		< 1
MFJS3	643		< 1	537		< 1	643		< 1	718		< 1
MFJS4	710		< 1	656		< 1	710		< 1	656		< 1
MFJS5	686		< 1	612		< 1	686		< 1	745		< 1
MFJS6	1055		< 1	801		< 1	1055		< 1	941		< 1

Table B.9.: Results of DDO for Blocking datasets (continued).

<table>
<tr><th></th><th colspan="6">Restricted</th><th colspan="6">Relaxed</th></tr>
<tr><th></th><th colspan="3">$w = 1$</th><th colspan="3">$w = 100$</th><th colspan="3">$w = 1$</th><th colspan="3">$w = 100$</th></tr>
<tr><th>Name</th><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th><th>UB</th><th>LB</th><th>t (s)</th></tr>
<tr><td>MFJS7</td><td colspan="2">1308</td><td>< 1</td><td>1003</td><td>1097</td><td>61</td><td colspan="2">1575</td><td>< 1</td><td colspan="2">1480</td><td>< 1</td></tr>
<tr><td>MFJS8</td><td colspan="2">1419</td><td>< 1</td><td>916</td><td>1046</td><td>61</td><td colspan="2">1419</td><td>< 1</td><td colspan="2">1472</td><td>< 1</td></tr>
<tr><td>MFJS9</td><td colspan="2">1671</td><td>< 1</td><td colspan="2">1355</td><td>1</td><td colspan="2">1856</td><td>< 1</td><td colspan="2">1837</td><td>< 1</td></tr>
<tr><td>MFJS10</td><td colspan="2">1863</td><td>< 1</td><td>1163</td><td>1506</td><td>61</td><td colspan="2">2106</td><td>< 1</td><td colspan="2">2088</td><td>< 1</td></tr>
<tr><th colspan="13">Hurink edata</th></tr>
<tr><td>abz5</td><td>4950</td><td></td><td>60</td><td>2203</td><td></td><td>61</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>abz6</td><td>1990</td><td></td><td>61</td><td>1340</td><td></td><td>62</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>abz7</td><td>3296</td><td></td><td>60</td><td>875</td><td></td><td>60</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>abz8</td><td>3289</td><td></td><td>60</td><td>990</td><td></td><td>60</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>abz9</td><td>3294</td><td></td><td>60</td><td>1399</td><td></td><td>60</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>car1</td><td colspan="2">7252</td><td>56</td><td colspan="2">7252</td><td>28</td><td colspan="2">9326</td><td>< 1</td><td colspan="2">9918</td><td>< 1</td></tr>
<tr><td>car2</td><td colspan="2">9031</td><td>8</td><td>6110</td><td>7667</td><td>67</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>car3</td><td>6155</td><td>8321</td><td>63</td><td colspan="2">8055</td><td>66</td><td colspan="2">11214</td><td>< 1</td><td colspan="2">9653</td><td>< 1</td></tr>
<tr><td>car4</td><td colspan="2">9149</td><td>9</td><td>5246</td><td>9134</td><td>65</td><td></td><td></td><td>< 1</td><td colspan="2">10191</td><td>< 1</td></tr>
<tr><td>car5</td><td colspan="2">8078</td><td>10</td><td colspan="2">8066</td><td>13</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>car6</td><td colspan="2">9201</td><td>2</td><td colspan="2">8873</td><td>2</td><td></td><td></td><td>< 1</td><td colspan="2">10336</td><td>< 1</td></tr>
<tr><td>car7</td><td colspan="2">6788</td><td>< 1</td><td colspan="2">6788</td><td>< 1</td><td colspan="2">8000</td><td>< 1</td><td colspan="2">7626</td><td>< 1</td></tr>
<tr><td>car8</td><td colspan="2">8473</td><td>3</td><td colspan="2">8473</td><td>4</td><td colspan="2">13200</td><td>< 1</td><td colspan="2">9427</td><td>< 1</td></tr>
<tr><td>la01</td><td colspan="2">1310</td><td>21</td><td colspan="2">1380</td><td>4</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la02</td><td colspan="2">1222</td><td>14</td><td colspan="2">1186</td><td>9</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la03</td><td colspan="2">1001</td><td>8</td><td colspan="2">1001</td><td>3</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la04</td><td colspan="2">1229</td><td>4</td><td colspan="2">1156</td><td>3</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la05</td><td colspan="2">994</td><td>18</td><td colspan="2">1031</td><td>5</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la06</td><td>767</td><td></td><td>62</td><td>1164</td><td>1999</td><td>64</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la07</td><td>1110</td><td></td><td>62</td><td>1354</td><td>1836</td><td>64</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la08</td><td>602</td><td></td><td>61</td><td>1189</td><td>1888</td><td>63</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la09</td><td>715</td><td></td><td>62</td><td colspan="2">2107</td><td>30</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la10</td><td>783</td><td></td><td>62</td><td>949</td><td>2101</td><td>64</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la11</td><td>698</td><td></td><td>62</td><td>827</td><td></td><td>62</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la12</td><td>694</td><td></td><td>62</td><td>962</td><td></td><td>62</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la13</td><td>992</td><td></td><td>62</td><td>1051</td><td></td><td>63</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la14</td><td>526</td><td></td><td>62</td><td></td><td></td><td>63</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la15</td><td>609</td><td></td><td>62</td><td>1052</td><td></td><td>62</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la16</td><td>1589</td><td></td><td>60</td><td>1311</td><td>2997</td><td>62</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la17</td><td>931</td><td></td><td>60</td><td>1196</td><td>2692</td><td>61</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la18</td><td>1064</td><td></td><td>60</td><td>1333</td><td>3115</td><td>62</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la19</td><td></td><td></td><td>60</td><td>1240</td><td></td><td>61</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la20</td><td>1124</td><td></td><td>60</td><td>1442</td><td>3175</td><td>62</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la21</td><td>1199</td><td></td><td>60</td><td>603</td><td></td><td>61</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la22</td><td>767</td><td></td><td>61</td><td>684</td><td></td><td>61</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la23</td><td>906</td><td></td><td>60</td><td>558</td><td></td><td>61</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
<tr><td>la24</td><td>750</td><td></td><td>61</td><td>729</td><td></td><td>61</td><td></td><td></td><td>< 1</td><td></td><td></td><td>< 1</td></tr>
</table>

Table B.9.: Results of DDO for Blocking datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 100$			$w = 1$			$w = 100$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
la25	749		61	400		61			< 1			< 1
la26	1395		60	782		61			< 1			< 1
la27	1399		60	1048		61			< 1			< 1
la28	1209		60	741		60			< 1			< 1
la29	999		60	418		60			< 1			< 1
la30	1012		60	925		61			< 1			< 1
la31	1796		60	1101		60			< 1			< 1
la32	1799		60	1167		60			< 1			< 1
la33	1497		60	568		61			< 1			< 1
la34	1512		60	1068		60			< 1			< 1
la35	1502		60	640		61			< 1			< 1
la36	1590		60	955		60			< 1			< 1
la37	1586		60	1196		60			< 1			< 1
la38	1346		60	644		60			< 1			< 1
la39	1140		60	774		60			< 1			< 1
la40	1573		60	610		60			< 1			< 1
mt06	107		< 1	104		< 1			< 1			< 1
mt10	1294	2294	61	1713	2172	64			< 1			< 1
mt20	500		61	557		62			< 1			< 1
orb1	1324	2198	61	2073		46			< 1			< 1
orb2	1304		60	1898	2742	63			< 1			< 1
orb3	1184	1784	61	1408	1648	63			< 1			< 1
orb4	2017	3123	61	2992		40			< 1			< 1
orb5	1139	2551	60	1473	2047	63			< 1			< 1
orb6	1288	2728	61	2289		53			< 1			< 1
orb7	623	1495	60	896	1242	63			< 1			< 1
orb8	1610		61	1199	1525	63			< 1			< 1
orb9	1790	2704	61	2580		24			< 1			< 1
orb10	3367		60	1753	2709	63			< 1			< 1
Hurink rdata												
abz5	4950		60	1440		62			< 1			< 1
abz6	1990		61	944		61			< 1			< 1
abz7	3296		60	794		60			< 1			< 1
abz8	3289		60	1010		60			< 1			< 1
abz9	3293		60	690		60			< 1			< 1
car1	2624	8242	62	2575	7291	63			< 1			< 1
car2	2471	7947	64	2626	7624	65			< 1	9303		< 1
car3	5924	8128	62	4597	7887	63			< 1			< 1
car4	2383	9113	64	2382	8838	63			< 1			< 1
car5	3446	9082	62	3432	8298	63			< 1			< 1
car6	4404	10456	61	4401	9027	62			< 1			< 1
car7	4272	6680	62	4166	6608	64			< 1			< 1

Table B.9.: Results of DDO for Blocking datasets (continued).

	Restricted						**Relaxed**					
	$w = 1$			$w = 100$			$w = 1$			$w = 100$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
car8	4353	9227	62	6984	9195	63			< 1			< 1
la01	605	991	63	532	927	64			< 1			< 1
la02	596	930	63	362	819	64			< 1			< 1
la03	945		63	556	756	64			< 1			< 1
la04	316	992	63	314	844	64			< 1			< 1
la05	838		64	285	795	64			< 1			< 1
la06	686	1612	63	708	1369	62			< 1			< 1
la07	613		63	515	1399	62			< 1			< 1
la08	374		63	350	1375	63			< 1			< 1
la09	525		63	323		63			< 1			< 1
la10	399		62	573	1327	62			< 1			< 1
la11	696		62	404		62			< 1			< 1
la12	500		62	475		62			< 1			< 1
la13	512		62	670		62			< 1			< 1
la14	500		62	372		62			< 1			< 1
la15	595		62	515		62			< 1			< 1
la16	817		61	552	2385	62			< 1			< 1
la17	510		61	363		61			< 1			< 1
la18	994		60	412		61			< 1			< 1
la19	526		61	429		61			< 1			< 1
la20	603		61	302		61			< 1			< 1
la21	1199		61	645		60			< 1			< 1
la22	916		61	751		61			< 1			< 1
la23	760		60	450		60			< 1			< 1
la24	750		61	434		61			< 1			< 1
la25	749		61	383		60			< 1			< 1
la26	1395		61	591		61			< 1			< 1
la27	1399		61	559		61			< 1			< 1
la28	1013		61	958		61			< 1			< 1
la29	999		61	455		61			< 1			< 1
la30	1013		61	614		61			< 1			< 1
la31	1796		60	795		61			< 1			< 1
la32	1504		61	603		61			< 1			< 1
la33	1497		60	542		60			< 1			< 1
la34	1512		61	776		61			< 1			< 1
la35	1502		61	729		61			< 1			< 1
la36	1823		60	925		60			< 1			< 1
la37	1150		60	587		60			< 1			< 1
la38	1132		60	559		60			< 1			< 1
la39	1140		60	535		60			< 1			< 1
la40	1352		60	535		60			< 1			< 1
mt06	68		34	68		26			< 1			< 1
mt10	225		60			61			< 1			< 1

Table B.9.: Results of DDO for Blocking datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 100$			$w = 1$			$w = 100$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
mt20	230		62	359	1985	62			< 1			< 1
orb1	522		60	459		61			< 1			< 1
orb2	610		61	410		61			< 1			< 1
orb3	516	2109	61	604	1898	63			< 1			< 1
orb4	508		61	425	2854	62			< 1			< 1
orb5	501		61	374		62			< 1			< 1
orb6	620		61	590		61			< 1			< 1
orb7	51		61	303		61			< 1			< 1
orb8	497	2371	61	372	1708	62			< 1			< 1
orb9	509		61	348	2468	62			< 1			< 1
orb10			61	444		61			< 1			< 1
Hurink vdata												
abz5	4950		62	1355	2315	61			< 1			< 1
abz6	1850		7	691	1479	61			< 1			< 1
abz7	3296		63	706		61			< 1			< 1
abz8	3289		62	836		61			< 1			< 1
abz9	3293		61	995		61			< 1			< 1
car1	1680	8242	62	1680	7051	62			< 1			< 1
car2	1938	8691	64	1980	7303	63			< 1	9649		< 1
car3	5924	8565	64	3458	7630	63			< 1			< 1
car4	2174	8889	63	3068	8121	62			< 1			< 1
car5	2099	8040	62	2625	6777	62			< 1			< 1
car6	1716	8971	62	999	7782	61			< 1			< 1
car7	2688	6287	62	2123	5893	62			< 1	8802		< 1
car8	1425	8095	62	2029	7341	61			< 1			< 1
la01	605	899	63	500	767	63			< 1			< 1
la02	596	778	63	382	719	63			< 1	1214		< 1
la03	359	904	63	245	698	63			< 1			< 1
la04	254	845	64	240	790	64			< 1			< 1
la05	268	735	64	329	674	65			< 1			< 1
la06	686		62	1259		62			< 1			< 1
la07	1110		63	722	1337	62			< 1			< 1
la08	374	1676	62	261	1098	62			< 1			< 1
la09	525		62	334	1350	62			< 1			< 1
la10	399		61	713	1182	62			< 1			< 1
la11	696		61	394	1738	61			< 1			< 1
la12	500		62	391		62			< 1			< 1
la13	992		61	593	2145	61			< 1			< 1
la14	500		62	375		61			< 1			< 1
la15	595		61	301	1884	61			< 1			< 1
la16	725	1425	61	302	1274	61			< 1			< 1
la17	510	1109	61	247	975	61			< 1			< 1

Table B.9.: Results of DDO for Blocking datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 100$			$w = 1$			$w = 100$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
la18	531	2136	61	262	1323	61			< 1			< 1
la19	526	1521	61	248	1401	61			< 1			< 1
la20	603	1570	61	345	1197	61			< 1			< 1
la21	1054		61	535		61			< 1			< 1
la22	767	2171	62	465	2053	61			< 1			< 1
la23	758		61	582		61			< 1			< 1
la24	750		61	318		61			< 1			< 1
la25	749		61	252	2532	61			< 1			< 1
la26	1395		61	741		61			< 1			< 1
la27	1009		61	464		61			< 1			< 1
la28	1013		61	563		61			< 1			< 1
la29	999		61	417		61			< 1			< 1
la30	1012		62	422		61			< 1			< 1
la31	1502		61	599		61			< 1			< 1
la32	1504		61	516		61			< 1			< 1
la33	1497		61	761		61			< 1			< 1
la34	1512		61	622		61			< 1			< 1
la35	1502		61	616		61			< 1			< 1
la36	1590		60	457		61			< 1			< 1
la37	1150		60	568		61			< 1			< 1
la38	1126		61	351		62			< 1			< 1
la39	1140		61	373		61			< 1			< 1
la40	1140		60	577		60			< 1			< 1
mt06	37	54	64	32	53	64			< 1			< 1
mt10	216	1452	61	187	1212	61			< 1	1934		< 1
mt20	202		61	356	1491	61			< 1			< 1
orb1	521		61	274	1591	61			< 1			< 1
orb2	610	1796	62	323	1247	61			< 1			< 1
orb3	516	1432	61	359	1217	61			< 1			< 1
orb4	508	1440	62	235	1200	61			< 1			< 1
orb5	501	1392	61	241	1027	61			< 1	1919		< 1
orb6	620	1370	61	311	1085	61	2367		< 1			< 1
orb7	45	919	62	245	666	61			< 1	1078		< 1
orb8	497	1442	61	165	1288	61			< 1			< 1
orb9	509		62	202	1425	61			< 1			< 1
orb10	502		61	333	1921	61			< 1			< 1
					Kacem							
1	11		< 1	11		< 1	12		< 1	12		< 1
2	16		< 1	14		< 1	16		< 1	15		< 1
3	10		< 1	9		< 1	10		< 1	9		< 1
4	25		< 1	18		< 1	25		< 1	19		< 1

B.7. APP — Multivalued Decision Diagrams

In Table B.10, we present the results of DDO for all APP datasets (see Section 6.1.4). The maximum allowed runtime was set to 60 seconds. The lower-bound values as reported by DDO are incorrect due to the time-limit cutoff. Instances which are 'solved to optimality' are solved to optimality with respect to their allowed width, but may not have reached the optimal value for that instance.

Table B.10.: DDO results for all APP datasets.

	Restricted						Relaxed					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
Barnes												
mt10c1	76	631	60	78	620	60	880		< 1	816		< 1
mt10cc	73	673	60	80	599	60	838		< 1	869		< 1
mt10x	118	671	60	124	650	61	825		< 1	737		< 1
mt10xx	84	593	60	86	562	60	848		< 1	714		< 1
mt10xxx	74	723	60	75	611	60	946		< 1	879		< 1
mt10xy	49	772	60	61	690	60	843		< 1	870		< 1
mt10xyz	81	698	60	69	629	60	786		< 1	768		< 1
setb4c9	86	797	60	56	733	60	1137		< 1	1124		< 1
setb4cc	89	862	60	66	738	60	960		< 1	1005		< 1
setb4x	49	802	60	53	753	60	942		< 1	842		< 1
setb4xx	52	799	60	50	751	60	880		< 1	832		< 1
setb4xxx	48	922	60	46	909	60	994		< 1	1159		< 1
setb4xy	43	754	60	45	706	60	811		< 1	879		< 1
setb4xyz	50	683	60	52	645	60	844		< 1	880		< 1
seti5c12	71	911	60	48	850	60	1028		< 1	975		< 1
seti5cc	123	908	60	92	828	60	1027		< 1	961		< 1
seti5x	81	874	60	54	928	60	903		< 1	1057		< 1
seti5xx	149	870	60	106	812	60	885		< 1	982		< 1
seti5xxx	129	863	60	99	824	60	941		< 1	933		< 1
seti5xy	104	829	60	67	825	60	904		< 1	833		< 1
seti5xyz	100	937	60	59	861	60	1249		< 1	1020		< 1
Brandimarte												
Mk01	6	20	61	5	19	63	20		< 1	26		< 1
Mk02	17		60	3	16	63	35		< 1	28		< 1
Mk03	10	141	60	6	138	60	190		< 1	147		< 1
Mk04	6	31	60	4	28	61	34		< 1	45		< 1
Mk05	31	110	60	16	102	61	134		< 1	113		< 1
Mk06	10	35	60	5	35	60	37		< 1	42		< 1
Mk07	5	96	60	4	91	61	116		< 1	113		< 1
Mk08	50	317	60	15	301	60	391		< 1	348		< 1
Mk09	55	266	60	17	238	61	319		< 1	260		< 1
Mk10	55	144	60	18	165	61	287		< 1	191		< 1
Dauzère-Paulli												

Table B.10.: DDO results for all APP datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
01a	261	1589	60	178	1510	60	2057		< 1	1696		< 1
02a	182	1262	60	154	1185	60	1655		< 1	1352		< 1
03a	212	1943	60	124	1747	60	2437		< 1	1906		< 1
04a	282	1422	60	196	1352	60	1428		< 1	1446		< 1
05a	197	1505	60	121	1412	60	1758		< 1	1654		< 1
06a	322	1622	60	221	1532	60	2022		< 1	1727		< 1
07a	166	1950	60	85	1724	60	2391		< 1	1960		< 1
08a	173	1677	60	105	1550	61	2228		< 1	1942		< 1
09a	227	1387	60	129	1365	61	1587		< 1	1484		< 1
10a	266	1685	60	154	1663	60	2161		< 1	2070		< 1
11a	305	1556	60	161	1521	61	1869		< 1	1613		< 1
12a	261	1405	60	136	1410	61	1617		< 1	1511		< 1
13a	184	1512	60	111	1635	61	1783		< 1	1903		< 1
14a	242	1709	61	116	1670	60	1991		< 1	1820		< 1
15a	236	1922	61	2007		63	2120		< 1	2007		< 1
16a	317	1856	60	133	1780	61	2098		< 1	1886		< 1
17a	173	1611	60	1716		62	1616		< 1	1716		< 1
18a	222	1421	61	1625		63	1604		< 1	1625		< 1
Fattahi												
SFJS1	24		< 1	24		< 1	24		< 1	24		< 1
SFJS2	43		< 1	43		< 1	64		< 1	43		< 1
SFJS3	106		< 1	106		< 1	130		< 1	106		< 1
SFJS4	179		< 1	179		< 1	179		< 1	179		< 1
SFJS5	73		< 1	73		< 1	73		< 1	73		< 1
SFJS6	160		< 1	160		< 1	197		< 1	197		< 1
SFJS7	247		< 1	247		< 1	247		< 1	247		< 1
SFJS8	146		< 1	146		< 1	162		< 1	152		< 1
SFJS9	80		< 1	80		< 1	117		< 1	97		< 1
SFJS10	238		< 1	238		< 1	384		< 1	260		< 1
MFJS1	285		4	285		2	378		< 1	345		< 1
MFJS2	273		2	273		< 1	290		< 1	283		< 1
MFJS3	160		2	160		< 1	246		< 1	246		< 1
MFJS4	202	269	60	269		39	332		< 1	269		< 1
MFJS5	165	223	60	192	223	69	337		< 1	248		< 1
MFJS6	170	215	60	193	215	67	300		< 1	283		< 1
MFJS7	215	607	60	223	560	63	880		< 1	880		< 1
MFJS8	150	567	60	154	544	62	712		< 1	712		< 1
MFJS9	165	624	60	167	570	61	790		< 1	954		< 1
MFJS10	180	803	60	192	773	62	1109		< 1	863		< 1
Hurink edata												
abz5	400	990	60	149	856	60	1202		< 1	1069		< 1

Table B.10.: DDO results for all APP datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
abz6	146	621	60	94	547	60	918		< 1	905		< 1
abz7	122	524	60	45	501	60	626		< 1	518		< 1
abz8	147	492	60	54	465	61	669		< 1	488		< 1
abz9	154	569	60	58	514	60	587		< 1	600		< 1
car1	551	3159	60	717	2933	61	3620		< 1	3982		< 1
car2	490	4070	60	520	3587	61	5243		< 1	4012		< 1
car3	537	3223	60	571	2930	61	3850		< 1	3184		< 1
car4	428	4051	60	440	3837	61	4928		< 1	4457		< 1
car5	876	3814	60	890	3570	61	5582		< 1	6148		< 1
car6	948	5353	60	973	4690	61	5489		< 1	5489		< 1
car7	915	3460	60	1040	3038	61	4409		< 1	3460		< 1
car8	1034	4597	60	1210	4378	61	6234		< 1	6167		< 1
la01	75	421	60	77	394	61	516		< 1	457		< 1
la02	95	351	60	95	326	62	485		< 1	416		< 1
la03	100	308	60	113	279	63	333		< 1	331		< 1
la04	75	375	60	78	341	62	449		< 1	435		< 1
la05	67	353	60	78	303	62	443		< 1	368		< 1
la06	76	499	60	73	473	61	564		< 1	539		< 1
la07	52	500	60	58	474	61	580		< 1	540		< 1
la08	53	529	60	57	454	61	546		< 1	513		< 1
la09	57	558	60	59	528	60	699		< 1	575		< 1
la10	67	543	60	74	500	60	739		< 1	640		< 1
la11	36	581	60	35	576	60	714		< 1	581		< 1
la12	26	651	60	29	612	61	722		< 1	710		< 1
la13	52	783	60	54	720	60	895		< 1	753		< 1
la14	32	662	60	36	639	60	727		< 1	753		< 1
la15	45	706	60	45	639	60	891		< 1	687		< 1
la16	86	669	60	61	605	60	916		< 1	696		< 1
la17	63	529	60	66	525	60	686		< 1	611		< 1
la18	118	676	60	120	617	60	779		< 1	660		< 1
la19	71	598	60	82	550	60	763		< 1	737		< 1
la20	122	793	60	77	695	60	937		< 1	889		< 1
la21	60	820	60	47	807	60	903		< 1	883		< 1
la22	43	533	60	35	547	60	821		< 1	805		< 1
la23	46	836	60	43	767	60	929		< 1	988		< 1
la24	66	784	60	53	688	60	983		< 1	814		< 1
la25	101	736	60	85	724	60	801		< 1	814		< 1
la26	66	967	60	50	890	60	1103		< 1	985		< 1
la27	79	1043	60	61	896	60	1335		< 1	1028		< 1
la28	49	870	60	35	884	60	1045		< 1	1008		< 1
la29	88	960	60	59	803	60	1045		< 1	963		< 1
la30	52	1124	60	44	917	60	1388		< 1	1236		< 1
la31	56	1420	60	30	1307	61	1440		< 1	1430		< 1

Table B.10.: DDO results for all APP datasets (continued).

	Restricted						Relaxed					
	$w=1$			$w=5$			$w=1$			$w=5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
la32	49	1599	60	30	1416	60	1646		< 1	1704		< 1
la33	47	1474	60	25	1402	61	1474		< 1	1653		< 1
la34	128	1398	60	51	1338	61	1483		< 1	1517		< 1
la35	41	1393	60	29	1389	60	1393		< 1	1473		< 1
la36	101	1001	60	82	872	60	1348		< 1	1032		< 1
la37	106	1039	60	71	966	60	1341		< 1	1064		< 1
la38	62	732	60	41	677	60	732		< 1	834		< 1
la39	69	962	60	53	881	60	1138		< 1	1031		< 1
la40	84	1035	60	47	965	60	1215		< 1	1085		< 1
mt06	19	34	60	27	34	71	43		< 1	39		< 1
mt10	76	661	60	78	553	60	721		< 1	665		< 1
mt20	24	707	60	30	664	60	812		< 1	744		< 1
orb1	83	631	60	94	617	60	676		< 1	731		< 1
orb2	58	737	60	63	713	60	924		< 1	814		< 1
orb3	50	663	60	50	656	60	791		< 1	821		< 1
orb4	96	541	60	80	522	61	783		< 1	617		< 1
orb5	87	675	60	75	559	60	787		< 1	697		< 1
orb6	97	677	60	75	647	60	868		< 1	847		< 1
orb7	28	289	60	27	252	60	307		< 1	304		< 1
orb8	66	713	60	71	665	60	799		< 1	760		< 1
orb9	63	740	60	57	732	60	841		< 1	786		< 1
orb10	73	712	60	75	650	60	850		< 1	774		< 1
Hurink rdata												
abz5	338	672	60	175	640	60	868		< 1	761		< 1
abz6	121	626	60	67	572	60	794		< 1	781		< 1
abz7	128	412	60	44	403	61	459		< 1	451		< 1
abz8	121	434	60	31	451	61	532		< 1	465		< 1
abz9	123	390	60	41	375	60	424		< 1	425		< 1
car1	672	3254	60	485	3178	62	4038		< 1	3459		< 1
car2	302	3870	60	345	3548	61	4354		< 1	3994		< 1
car3	1100	2982	60	680	2769	61	3342		< 1	3101		< 1
car4	251	3973	60	333	3712	61	5119		< 1	4228		< 1
car5	558	3936	60	588	3740	61	4872		< 1	4429		< 1
car6	561	4391	60	625	3625	61	6048		< 1	5026		< 1
car7	785	2998	60	874	2985	61	4762		< 1	3622		< 1
car8	852	4291	60	777	3511	61	5717		< 1	5152		< 1
la01	77	396	60	83	364	62	522		< 1	460		< 1
la02	67	366	60	67	327	62	461		< 1	444		< 1
la03	41	326	60	42	292	61	411		< 1	359		< 1
la04	45	347	60	56	309	62	512		< 1	378		< 1
la05	52	394	60	54	362	62	505		< 1	443		< 1
la06	54	630	60	55	571	61	655		< 1	663		< 1

Table B.10.: DDO results for all APP datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
la07	39	435	60	44	404	61	464		< 1	494		< 1
la08	27	541	60	27	509	60	664		< 1	580		< 1
la09	46	552	60	51	538	60	697		< 1	657		< 1
la10	48	554	60	51	515	61	736		< 1	654		< 1
la11	31	646	60	28	624	60	680		< 1	667		< 1
la12	22	571	60	22	542	60	721		< 1	674		< 1
la13	27	604	60	28	606	60	841		< 1	693		< 1
la14	25	697	60	25	647	60	932		< 1	715		< 1
la15	38	701	60	38	648	60	857		< 1	741		< 1
la16	133	595	60	109	572	60	751		< 1	636		< 1
la17	68	495	60	62	433	60	601		< 1	547		< 1
la18	53	492	60	49	483	60	697		< 1	565		< 1
la19	72	702	60	82	622	60	869		< 1	740		< 1
la20	59	587	60	55	548	60	659		< 1	605		< 1
la21	75	746	60	48	689	60	871		< 1	925		< 1
la22	53	688	60	38	642	60	804		< 1	765		< 1
la23	61	757	60	50	709	60	1011		< 1	925		< 1
la24	101	675	60	87	642	60	800		< 1	970		< 1
la25	47	757	60	22	687	60	880		< 1	896		< 1
la26	67	833	60	40	755	60	871		< 1	904		< 1
la27	40	880	60	29	882	61	1149		< 1	1105		< 1
la28	97	1019	60	79	953	61	1091		< 1	1147		< 1
la29	47	900	60	26	894	61	1057		< 1	1056		< 1
la30	129	962	60	75	865	60	1043		< 1	1065		< 1
la31	97	1153	60	42	1123	61	1359		< 1	1128		< 1
la32	43	1470	61	28	1327	61	1516		< 1	1479		< 1
la33	47	1395	60	27	1325	61	1420		< 1	1350		< 1
la34	52	1355	60	26	1205	63	1475		< 1	1301		< 1
la35	49	1238	60	26	1167	61	1418		< 1	1251		< 1
la36	93	764	60	63	802	60	1042		< 1	880		< 1
la37	140	879	60	108	873	61	987		< 1	1057		< 1
la38	67	821	60	38	775	60	970		< 1	798		< 1
la39	69	876	60	43	851	60	1066		< 1	923		< 1
la40	118	936	60	54	878	61	1053		< 1	1009		< 1
mt06	10	29	60	11	28	62	53		< 1	42		< 1
mt10	94	586	60	69	520	60	888		< 1	720		< 1
mt20	31	634	60	31	592	60	794		< 1	704		< 1
orb1	66	597	60	55	525	60	825		< 1	699		< 1
orb2	97	544	60	58	470	60	741		< 1	665		< 1
orb3	53	536	60	53	471	60	701		< 1	701		< 1
orb4	47	556	60	51	532	60	753		< 1	695		< 1
orb5	409		60	66	388	60	609		< 1	404		< 1
orb6	113	549	60	55	567	60	847		< 1	715		< 1

Table B.10.: DDO results for all APP datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
orb7	38	290	60	25	256	60	344		< 1	292		< 1
orb8	129	568	60	75	519	60	673		< 1	663		< 1
orb9	115	501	60	114	477	60	728		< 1	669		< 1
orb10	101	622	60	113	488	60	786		< 1	702		< 1
Hurink vdata												
abz5	313	596	61	132	596	60	628		< 1	677		< 1
abz6	236	592	60	148	600	60	667		< 1	755		< 1
abz7	144	446	61	38	368	61	537		< 1	368		< 1
abz8	126	376	62	33	421	63	511		< 1	511		< 1
abz9	159	478	61	62	472	65	494		< 1	549		< 1
car1	590	2745	60	514	2662	61	4587		< 1	2969		< 1
car2	239	3970	60	302	3598	61	4530		< 1	4507		< 1
car3	443	4133	60	359	3792	61	5267		< 1	4965		< 1
car4	321	4704	60	321	4518	61	5133		< 1	5126		< 1
car5	731	3661	60	720	3543	61	4557		< 1	4448		< 1
car6	392	3660	60	483	3703	61	4296		< 1	4246		< 1
car7	483	2883	60	565	2883	61	3588		< 1	3904		< 1
car8	1186	3837	60	1262	3837	61	4617		< 1	3976		< 1
la01	69	389	60	66	389	61	532		< 1	425		< 1
la02	87	371	60	81	325	61	454		< 1	413		< 1
la03	41	319	60	41	291	61	446		< 1	372		< 1
la04	22	315	60	19	300	61	391		< 1	372		< 1
la05	65	301	60	72	248	61	522		< 1	336		< 1
la06	52	470	60	49	434	61	513		< 1	511		< 1
la07	84	448	60	65	425	61	512		< 1	525		< 1
la08	34	421	60	35	375	61	551		< 1	469		< 1
la09	34	554	60	37	532	61	629		< 1	656		< 1
la10	96	524	60	51	490	61	615		< 1	516		< 1
la11	60	645	60	30	617	61	750		< 1	645		< 1
la12	26	571	60	26	563	60	741		< 1	717		< 1
la13	38	667	60	38	619	60	762		< 1	701		< 1
la14	25	732	60	20	660	60	874		< 1	707		< 1
la15	39	696	60	36	652	60	742		< 1	741		< 1
la16	148	523	61	108	503	60	679		< 1	555		< 1
la17	84	449	60	64	449	60	478		< 1	459		< 1
la18	142	478	60	62	449	61	614		< 1	543		< 1
la19	53	478	60	38	367	60	502		< 1	554		< 1
la20	48	517	60	23	502	61	780		< 1	569		< 1
la21	133	652	61	104	636	60	733		< 1	666		< 1
la22	55	598	60	37	557	61	661		< 1	732		< 1
la23	44	584	60	33	571	61	787		< 1	707		< 1
la24	48	594	60	23	609	61	783		< 1	835		< 1

Table B.10.: DDO results for all APP datasets (continued).

	Restricted						Relaxed					
	$w = 1$			$w = 5$			$w = 1$			$w = 5$		
Name	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)	UB	LB	t (s)
la25	98	584	60	62	590	61	739		< 1	763		< 1
la26	60	842	61	36	846	61	922		< 1	945		< 1
la27	121	909	60	48	935	61	1036		< 1	964		< 1
la28	60	1026	60	38	949	61	1135		< 1	1079		< 1
la29	44	800	60	33	803	60	1077		< 1	902		< 1
la30	57	1064	60	38	1007	61	1076		< 1	1056		< 1
la31	56	1321	61	1326		62	1371		< 1	1326		< 1
la32	1418		61	1486		63	1418		< 1	1486		< 1
la33	1345		61	1257		63	1345		< 1	1257		< 1
la34	1234		61	1304		63	1234		< 1	1304		< 1
la35	1279		61	1172		62	1279		< 1	1172		< 1
la36	171	764	61	105	797	62	1140		< 1	936		< 1
la37	71	813	61	39	771	64	813		< 1	839		< 1
la38	114	783	61	83	758	62	863		< 1	786		< 1
la39	100	654	69	58	682	62	883		< 1	710		< 1
la40	78	784	60	32	784	62	825		< 1	784		< 1
mt06	7	30	60	8	30	62	37		< 1	32		< 1
mt10	29	525	60	29	510	60	599		< 1	548		< 1
mt20	25	726	60	22	642	60	861		< 1	775		< 1
orb1	67	492	60	42	492	60	565		< 1	646		< 1
orb2	64	560	60	47	544	60	621		< 1	544		< 1
orb3	64	574	60	48	574	60	654		< 1	604		< 1
orb4	106	556	60	43	556	60	764		< 1	573		< 1
orb5	133	441	60	86	419	60	583		< 1	499		< 1
orb6	73	503	61	45	503	61	759		< 1	620		< 1
orb7	15	197	60	15	197	60	209		< 1	206		< 1
orb8	45	447	60	31	445	60	528		< 1	499		< 1
orb9	56	535	60	33	526	60	725		< 1	666		< 1
orb10	47	551	60	34	539	60	647		< 1	628		< 1
Kacem												
1	8		2	8		1	9		< 1	8		< 1
2	2	9	61	2	8	71	10		< 1	10		< 1
3	2	4	60	2	3	67	5		< 1	5		< 1
4	3	9	60	2	9	67	11		< 1	9		< 1

Bibliography

L. R. Abreu and M. S. Nagano (2022). "A new hybridization of adaptive large neighborhood search with constraint programming for open shop scheduling with sequence-dependent setup times". In: *Computers & Industrial Engineering* 168, p. 108128. DOI: 10.1016/j.cie.2022.108128.

A. Ali, H. Elaswad, and A. Jabuda (Apr. 2025). "Optimizing Job Shop Scheduling with Alternative Routes: A Metaheuristic Approach". In: *African Journal of Advanced Pure and Applied Sciences* 4, pp. 146–151.

A. Azzouz, A. Chaabani, M. Ennigrou, and L. Ben Said (2020). "Handling Sequence-dependent Setup Time Flexible Job Shop Problem with Learning and Deterioration Considerations using Evolutionary Bi-level Optimization". In: *Applied Artificial Intelligence* 34.6, pp. 433–455. DOI: 10.1080/08839514.2020.1723871.

A. Azzouz, M. Ennigrou, and L. Ben Said (2016). "Flexible Job-shop Scheduling Problem with Sequence-dependent Setup Times using Genetic Algorithm". In: *International Conference on Enterprise Information Systems.* URL: 10.5220/0005821900470053.

A. Azzouz, M. Ennigrou, and L. Ben Said (Jan. 2017). "A hybrid algorithm for flexible job-shop scheduling problem with setup times". In: *International Journal of Production Management and Engineering* 5, p. 23. DOI: 10.4995/ijpme.2017.6618.

J. W. Barnes and J. B. Chambers (1995). "Solving the job shop scheduling problem with tabu search". In: *IIE Transactions* 27.2, pp. 257–263. DOI: 10.1080/07408179508936739.

D. Behnke and M. J. Geiger (Jan. 2012). *Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers.* Arbeitspapier RR-12-01-01. Hamburg: Helmut-Schmidt-Universität, Universität der Bundeswehr. URL: https://d-nb.info/1023241773/34.

K. Ben Ali, H. Louati, and S. Bechikh (Aug. 2024). "A Self-learning Particle Swarm Optimization Algorithm for Dynamic Job Shop Scheduling Problem with New Jobs Insertion". In: Singapore: Springer, pp. 70–84. DOI: 10.1007/978-981-97-7181-3_6.

D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. Hooker (Oct. 2016). *Decision Diagrams for Optimization.* Cham: Springer. DOI: 10.1007/978-3-319-42849-9.

J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan (1983). "Scheduling subject to resource constraints: classification and complexity". In: *Discrete Applied Mathematics* 5.1, pp. 11–24. DOI: 10.1016/0166-218X(83)90012-4.

F. Boutet and G. Motet (1998). "Use of constraints in Petri nets for modelling and solving scheduling problems in preliminary system design". In: *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics.* Vol. 1, pp. 576–581. DOI: 10.1109/ICSMC.1998.725474.

P. Brandimarte (Sept. 1993). "Routing and scheduling in a flexible job shop by tabu search". In: *Annals of Operations Research* 41, pp. 157–183. DOI: 10.1007/BF02023073.

P. Brucker and R. Schlie (1991). "Job-shop scheduling with multi-purpose machines". In: *Computing* 45, pp. 369–375. URL: https://api.semanticscholar.org/CorpusID:13262880.

R. E. Bryant (1986). "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* C-35.8, pp. 677–691. DOI: 10.1109/TC.1986.1676819.

R. Čapek, P. Šůcha, and Z. Hanzálek (2012). "Production scheduling with alternative process plans". In: *European Journal of Operational Research* 217.2, pp. 300–311. DOI: 10.1016/j.ejor.2011.09.018.

R. Chen, B. Yang, S. Li, and S. Wang (2020). "A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem". In: *Computers & Industrial Engineering* 149, p. 106778. DOI: 10.1016/j.cie.2020.106778.

A. A. Cire and W.-J. van Hoeve (Jan. 2012). "MDD Propagation for Disjunctive Scheduling". In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*, pp. 11–19.

V. Coppé, X. Gillard, and P. Schaus (Nov. 2024). "Decision Diagram-Based Branch-and-Bound with Caching for Dominance and Suboptimality Detection". In: *INFORMS Journal on Computing* 36.6, pp. 1522–1542. DOI: 10.1287/ijoc.2022.0340.

S. Dauzère-Pérès, J. Ding, L. Shen, and K. Tamssaouet (2024). "The flexible job shop scheduling problem: A review". In: *European Journal of Operational Research* 314.2, pp. 409–432. DOI: 10.1016/j.ejor.2023.05.017.

S. Dauzère-Pérès and J. Paulli (1994). *Solving the General Multiprocessor Job-shop Scheduling Problem.*

T.L. Dean and M.S. Boddy (1988). "An Analysis of Time-Dependent Planning". In: *AAAI*. Vol. 88, pp. 49–54.

P. Fattahi, M. Saidi-Mehrabad, and F. Jolai (July 2007). "Mathematical modeling and heuristic approaches to flexible job shop scheduling problems". In: *Journal of Intelligent Manufacturing* 18, pp. 331–342. DOI: 10.1007/s10845-007-0026-8.

G. Fernández, M. Ángel, C. Vela, and R. Arias (June 2013). "An Efficient Memetic Algorithm for the Flexible Job Shop with Setup Times". In: *ICAPS 2013 - Proceedings of the 23rd International Conference on Automated Planning and Scheduling*, pp. 91–99. DOI: 10.1609/icaps.v23i1.13542.

R. Fontaine, J. Dibangoye, and C. Solnon (2023). "Exact and anytime approach for solving the time dependent traveling salesman problem with time windows". In: *European Journal of Operational Research* 311.3, pp. 833–844. DOI: 10.1016/j.ejor.2023.06.001.

Z.L. Gan, S.M. Musa, and H.J. Yap (2023). "A Review of the High-Mix, Low-Volume Manufacturing Industry". In: *Applied Sciences* 13.3. DOI: 10.3390/app13031687.

M. R. Garey, D. S. Johnson, and R. Sethi (1976). "The Complexity of Flowshop and Jobshop Scheduling". In: *Mathematics of Operations Research* 1, pp. 117–129. URL: https://www.jstor.org/stable/3689278.

X. Gillard, P. Schaus, and V. Coppé (2020). "DDO, a generic and efficient framework for MDD-based optimization". In: *IJCAI-20 - Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. Available from https://github.com/xgillard/ddo, Pages 5243–5245. DOI: 10.24963/ijcai.2020/757.

R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan (1979). "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey". In: *Discrete Optimization II*. Vol. 5. Annals of Discrete Mathematics. Elsevier, pp. 287–326. DOI: 10.1016/S0167-5060(08)70356-X.

P. Hart, N. Nilsson, and B. Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

V. A. Hauder, A. Beham, S. Raggl, S. N. Parragh, and M. Affenzeller (2020). "Resource-constrained multi-project scheduling with activity and time flexibility". In: *Computers & Industrial Engineering* 150, p. 106857. DOI: 10.1016/j.cie.2020.106857.

V. Heinz, A. Novák, M. Vlk, and Z. Hanzálek (2022). "Constraint Programming and constructive heuristics for parallel machine scheduling with sequence-dependent setups and common servers". In: *Computers & Industrial Engineering* 172, p. 108586. DOI: 10.1016/j.cie.2022.108586.

M. Horn, J. Maschler, G. R. Raidl, and E. Rönnberg (2021). "A*-based construction of decision diagrams for a prize-collecting scheduling problem". In: *Computers & Operations Research* 126, p. 105125. DOI: 10.1016/j.cor.2020.105125.

J. Hurink, B. Jurisch, and M. Thole (Dec. 1994). "Tabu search for the job-shop scheduling problem with multi-purpose machines". In: *OR Spektrum* 15, pp. 205–215. DOI: 10.1007/BF01719451.

S. M. Johnson (1954). "Optimal two- and three-stage production schedules with setup times included". In: *Naval Research Logistics Quarterly* 1, pp. 61–68. URL: https://api.semanticscholar.org/CorpusID:62713652.

B. Jurish (Dec. 1992). "Scheduling jobs in shops with multi-purpose machines". PhD thesis. Osnabrück: Fachbereich Mathematik/Informatik, Universität Osnabrück.

I. Kacem, S. Hammadi, and P. Borne (2002). "Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic". In: *Mathematics and Computers in Simulation* 60.3, pp. 245–276. DOI: 10.1016/S0378-4754(02)00019-8.

T. Kis (2003). "Job-shop scheduling with processing alternatives". In: *European Journal of Operational Research* 151.2, pp. 307–332. DOI: 10.1016/S0377-2217(02)00828-7.

D. Kowalczyk and R. Leus (Dec. 2018). "A branch-and-price algorithm for parallel machine scheduling using ZDDs and generic branching". In: *INFORMS Journal on Computing.* Vol. 30, pp. 768–782. DOI: 10.1287/ijoc.2018.0809.

A. Kusiak and G. Finke (1988). "Selection of process plans in automated manufacturing systems". In: *IEEE Journal on Robotics and Automation* 4.4, pp. 397–402. DOI: 10.1109/56.803.

P. Laborie (2018). "An Update on the Comparison of MIP, CP and Hybrid Approaches for Mixed Resource Allocation and Scheduling". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research.* Springer International Publishing, pp. 403–411. ISBN: 978-3-319-93031-2.

L. Lan and J. Berkhout (2025). *PyJobShop: Solving scheduling problems with constraint programming in Python.* arXiv: 2502.13483. URL: https://arxiv.org/abs/2502.13483.

K. Lei, P. Guo, W. Zhao, Y. Wang, L. Qian, X. Meng, and L. Tang (2022). "A multi-action deep reinforcement learning framework for flexible Job-shop scheduling problem". In: *Expert Systems with Applications* 205, p. 117796. DOI: 10.1016/j.eswa.2022.117796.

H. Li, Y. Zheng, B. Sun, and B. Du (2024). "A Job Sequence Optimization Approach for Parallel Machine Scheduling Problem in Printing Manufacturing Systems". In: *IEEE Access* 12, pp. 63462–63476. DOI: 10.1109/ACCESS.2024.3396455.

X. Li and L. Gao (2016). "An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem". In: *International Journal of Production Economics* 174, pp. 93–110. DOI: 10.1016/j.ijpe.2016.01.016.

Chi-Shiuan Lin, Pei-Yi Li, Jun-Min Wei, and Muh-Cherng Wu (2020). "Integration of process planning and scheduling for distributed flexible job shops". In: *Computers & Operations Research* 124, p. 105053. DOI: https://doi.org/10.1016/j.cor.2020.105053.

W. T. Lunardi, E. G. Birgin, P. Laborie, D. P. Ronconi, and H. Voos (2020). "Mixed Integer linear programming and constraint programming models for the online printing shop scheduling problem". In: *Computers & Operations Research* 123, p. 105020. DOI: https://doi.org/10.1016/j.cor.2020.105020.

K. Matsumoto, K. Hatano, and E. Takimoto (2018). "Decision Diagrams for Solving a Job Scheduling Problem Under Precedence Constraints". In: *17th International Symposium on Experimental Algorithms (SEA 2018).* Vol. 103. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 1–12. DOI: 10.4230/LIPIcs.SEA.2018.5.

B. Naderi and V. Roshanaei (Aug. 2021). "Critical-Path-Search Logic-Based Benders Decomposition Approaches for Flexible Job Shop Scheduling". In: *INFORMS Journal on Optimization* 4. DOI: 10.1287/ijoo.2021.0056.

B. Naderi, R. Ruiz, and V. Roshanaei (2023). "Mixed-Integer Programming vs. Constraint Programming for Shop Scheduling Problems: New Results and Outlook". In: *INFORMS Journal on Computing* 35.4, pp. 817–843. DOI: 10.1287/ijoc.2023.1287.

N. M. Najid, S. Dauzère-Pérès, and A. Zaidat (2002). "A modified simulated annealing method for flexible job shop scheduling problem". In: *IEEE International Conference on Systems, Man and Cybernetics*. Vol. 5, p. 6. DOI: 10.1109/ICSMC.2002.1176334.

A. Oddi, R. Rasconi, A. Cesta, and S. Smith (Jan. 2011). "Applying iterative flattening search to the job shop scheduling problem with alternative resources and sequence dependent setup times". In: *COPLAS 2011 - Proceedings of the Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems*, pp. 15–22. DOI: 10.5555/2283696.2283733.

C. Özgüven, Y. Yavuz, and L. Özbakır (2012). "Mixed integer goal programming models for the flexible job-shop scheduling problems with separable and non-separable sequence dependent setup times". In: *Applied Mathematical Modelling* 36.2, pp. 846–858. DOI: 10.1016/j.apm.2011.07.037.

M. Pinedo (Jan. 2022). *Scheduling: Theory, Algorithms, and Systems*. Cham: Springer. DOI: 10.1007/978-3-031-05921-6.

R. Čapek, P. Šůcha, and Z. Hanzálek (2015). "Scheduling of Production with Alternative Process Plans". In: *Handbook on Project Management and Scheduling Vol. 2*. Springer International Publishing, pp. 1187–1204. DOI: 10.1007/978-3-319-05915-0_23.

R. Reijnen, I. G. Smit, H. Zhang, Y. Wu, Z. Bukhsh, and Y. Zhang (2025). *Job Shop Scheduling Benchmark: Environments and Instances for Learning and Non-learning Methods*. arXiv: 2308.12794 [cs.AI]. URL: https://arxiv.org/abs/2308.12794.

P. Richard, N. Jacquet, C. Cavalier, and C. Proust (Nov. 1995). "Solving scheduling problems using Petri nets and constraint logic programming". In: *Emerging Technologies and Factory Automation, 1995. ETFA '95, Proceedings., 1995 INRIA/IEEE Symposium*. Vol. 1, pp. 59–67. DOI: 10.1109/ETFA.1995.496763.

A. Rossi (2014). "Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships". In: *International Journal of Production Economics* 153, pp. 253–267. DOI: 10.1016/j.ijpe.2014.03.006.

T. Servranckx, J. Coelho, and M. Vanhoucke (2024). "A genetic algorithm for the Resource-Constrained Project Scheduling Problem with Alternative Subgraphs using a boolean satisfiability solver". In: *European Journal of Operational Research* 316.3, pp. 815–827. DOI: 10.1016/j.ejor.2024.02.041.

L. Shen, S. Dauzère-Pérès, and J. S. Neufeld (2018). "Solving the flexible job shop scheduling problem with sequence-dependent setup times". In: *European Journal of Operational Research* 265.2, pp. 503–516. DOI: 10.1016/j.ejor.2017.08.021.

R. L. Sisson (1959). "Methods of Sequencing in Job Shops - A Review". In: *Operations Research* 7.1, pp. 10–29. ISSN: 0030364X, 15265463. URL: http://www.jstor.org/stable/167590 (visited on 01/14/2025).

S. Tao and Z. S. Dong (2017). "Scheduling resource-constrained project problem with alternative activity chains". In: *Computers & Industrial Engineering* 114, pp. 288–296. DOI: 10.1016/j.cie.2017.10.027.

S. Tao and Z. S. Dong (2018). "Multi-mode resource-constrained project scheduling problem with alternative project structures". In: *Computers & Industrial Engineering* 125, pp. 333–347. DOI: 10.1016/j.cie.2018.08.027.

M. E. Tayebi-Araghi, F. Jolai, and M. Rabiee (2014). "Incorporating learning effect and deterioration for solving a SDST flexible job-shop scheduling problem with a hybrid meta-heuristic approach". In: *International Journal of Computer Integrated Manufacturing* 27.8, pp. 733–746. DOI: `10.1080/0951192X.2013.834465`. eprint: `https://doi.org/10.1080/0951192X.2013.834465`.

P. van den Bogaerdt (2018). "Multi-machine scheduling lower bounds using decision diagrams". MA thesis. Delft University of Technology, Department of Software Technology. URL: `https://repository.tudelft.nl/file/File_8ac51167-90e9-411d-9663-b6c18dc3d7f9`.

P. van den Bogaerdt and M. de Weerdt (2019). "Lower Bounds for Uniform Machine Scheduling Using Decision Diagrams". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research.* Vol. 11494. Lecture Notes in Computer Science. Springer, pp. 565–580. DOI: `10.1007/978-3-030-19212-9_38`.

W.-J. van Hoeve (Oct. 2024). "An Introduction to Decision Diagrams for Optimization". In: *Tutorials in Operations Research: Smarter Decisions for a Better World.* INFORMS. Chap. 4, pp. 117–145. DOI: `10.1287/educ.2024.0276`.

L. Wan, L. Fu, C. Li, and K. Li (2024). "Flexible job shop scheduling via deep reinforcement learning with meta-path-based heterogeneous graph neural network". In: *Knowledge-Based Systems* 296, p. 111940. DOI: `10.1016/j.knosys.2024.111940`.

H. Xiong, S. Shi, D. Ren, and J. Hu (2022). "A survey of job shop scheduling problem: The types and models". In: *Computers & Operations Research* 142, p. 105731. DOI: `10.1016/j.cor.2022.105731`.