Master Thesis : Optimization Strategies for Industrial-size Job Shop Scheduling

Auteur : Boveroux, Laurie
Promoteur(s) : Louveaux, Quentin
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en science des données, à finalité spécialisée
Année académique : 2022-2023
URI/URL : http://hdl.handle.net/2268.2/17710

UNIVERSITY OF LIÈGE - FACULTY OF APPLIED SCIENCES

Optimization Strategies for Industrial-size Job Shop Scheduling

---

Master's thesis completed in order to obtain the degree of
Master of Science in Data Science and Engineering
by

Boveroux Laurie

Supervisor
Louveaux Quentin

Jury members
Derval Guillaume
Fontaine Pascal
Louveaux Quentin

Academic Year
2022-2023

Abstract

This thesis investigates the optimization of a large-scale job shop scheduling process. Scheduling plays a crucial role in resource allocation and productivity maximization for companies. We evaluate the performance of established optimization techniques, including simulated annealing, branch and bound, dive, and a relaxation with a linear program we call the ranking method. We use three instances from the database of TOOWHE Enterprise Resource Planning (ERP) software, developed by Hi-pass. Through extensive experimentation on three instances of industrial-scale job shop problems, we assess the effectiveness and limitations of each algorithm. The simulated annealing algorithm shows promise by achieving significant objective value improvements within a reasonable execution time. However, the ranking method quickly outperforms it, providing equivalent solutions in a fraction of a second. By solving a relaxation of the problem with a linear program, the ranking method provides lower bounds and generates efficient operation schedules. To overcome the limitations of the branch and bound method, we propose the dive approach, which allows deeper exploration of the solution space while maintaining model consistency. By incorporating different strategies for operation selection, the dive approach achieves high-quality schedules within a limited number of dives. Our results highlight the limited scalability of the branch and bound method and the effectiveness of the simulated annealing algorithm, ranking method, and dive approach in generating reliable initial schedules. We recommend using the ranking method or dive approach to obtain an initial schedule and applying the simulated annealing algorithm for further refinement. This research contributes to the advancement of scheduling optimization in ERP systems, providing insights into algorithm performance and practical recommendations for improving scheduling efficiency and resource utilization in industrial contexts.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Q. Louveaux, for his guidance and mentoring throughout this research project. His expertise and feedback have been crucial in shaping the direction and quality of this thesis.

I would also like to thank Hi-Pass for providing me with the subject of this thesis and access to the real-world data and resources necessary to conduct this study. Their cooperation and collaboration were critical to gaining practical insight into industrial job shop scheduling problems and conducting meaningful research.

Finally, I would like to thank my family and friends for their support and encouragement throughout this journey. Their belief in my abilities has been a constant source of motivation and inspiration.

Without the collective support and encouragement of these individuals and organizations, this research would not have been possible.

Author's Note

Please note that for grammatical and syntactical reasons, some sections of this thesis have been rewritten using chatGPT, an AI language model developed by OpenAI. All content reviewed with the help of chatGPT was revised and edited to align with my own ideas and understanding. ChatGPT was used as a reformulation and grammar tool. Furthermore, all content submitted to chatGPT is my own. Therefore, while the language model was used as a tool, the intellectual contributions and ownership of the content remain solely mine.

Additionally, all the files used to conduct this thesis are available in the GitHub repository:
`https://github.com/LaurieBvrx/Optimization-Strategies-for-Industrial-size-Job-Shop-Scheduling.git`.

Table of Contents

Acronyms

BFS	Breadth-first Search.
BnB	Branch and Bound.
BNS	Best-node search.
CM-2LV	Critical machine and the 2 largest violations.
CM-5LV	Critical machine and one of the 5 largest violations.
CM-FO	Critical machine and first operations.
CM-LV	Critical machine and largest violation.
GA	Genetic Algorithm.
GS	Greedy search.
LJ	Largest job.
LP	Linear Program.
LS	Lukewarm start.
SA	Simulated Annealing.

1. Introduction

1.1. Context

TOOWHE is an Enterprise Resource Planning software developed by Hi-Pass. This software provides solutions to manage various operating processes of companies by integrating multiple management functions. Its features enable enterprises to reduce and automate administrative tasks and ensure the traceability of all activities.

Among the management functions provided by TOOWHE, scheduling is a crucial process. Scheduling involves efficiently allocating a company's available resources, including employees and machines, to maximize productivity. Scheduling is a process that plays an important role in the industrial sector. It allows companies to determine their billing rate and set delivery times for their clients. Having an optimal schedule allows companies to invoice faster the clients and optimize the use of machines and personnel. In other terms, it affects a company's overall success.

The scheduling algorithm in TOOWHE offers some optimization features. The optimization is done with a pure analysis approach. This involves thoroughly examining the existing schedule, identifying any gaps or inefficiencies, and making appropriate adjustments to improve its quality. An acceptable schedule can be produced, one that is both applicable and suitable for companies with their required timeframe.

In addition to the pure analysis approach, a global optimization perspective can be adopted and that is precisely the objective of this thesis. The objective is to design a process that goes beyond an acceptable schedule to achieve optimized scheduling, although not necessarily optimal. By testing different optimization algorithms and

techniques, we aim to develop a high-quality scheduling process, particularly when dealing with large-scale problems.

In summary, while the TOOWHE software optimizes the schedule through a pure analysis approach, this thesis aims to extend the optimization efforts by incorporating different optimization algorithms. The objective is to achieve a high-quality scheduling process that maximizes resource utilization and contributes to companies' overall efficiency in industrial contexts.

1.2. Field of Scheduling

Scheduling is the process of determining the order and timing of tasks or activities in a way that maximizes efficiency, meets deadlines, and optimizes resource utilization. This process involves allocating resources, such as time, people, and equipment, to specific tasks in a coordinated manner.

One effective way to visualize and understand scheduling is through the use of a Gantt chart. A Gantt chart provides a comprehensive timeline view, serving as a graphical representation of tasks or activities. Along the chart's x-axis, tasks are displayed as horizontal bars. The x-axis represents time and the y-axis represents machines. Each bar corresponds to a specific task, and its length illustrates the duration of that task. By observing the Gantt chart, we can easily grasp the temporal relationships between tasks and their respective timelines. An example from the TOOWHE software is given in Figure 1.1.

The field of scheduling has been extensively studied for several decades in the literature, with numerous books and papers covering various aspects of the topic. The papers range from elementary to very advanced, with some focusing on deterministic scheduling [5, 19], while others cover stochastic scheduling [36, 34, 37, 9] or online scheduling [4, 17, 24, 26]. A stochastic schedule is a type of schedule that takes into account the various sources of uncertainty or randomness that can occur in a production environment. These sources of uncertainty may include machine breakdowns, unexpected high-priority jobs, and imprecise processing times [31]. Online scheduling, on the other hand, focuses on scheduling

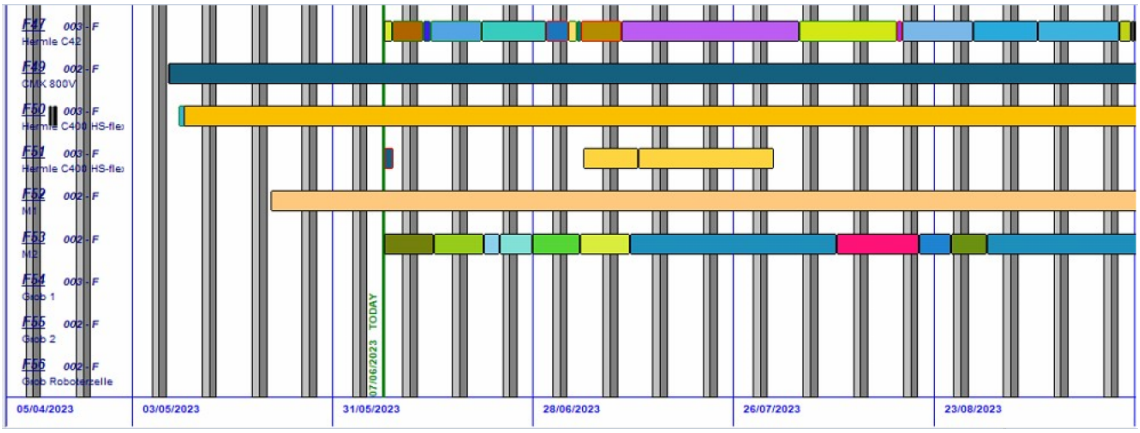


Figure 1.1.: Extract of a Gantt chart from TOOWHE

problems where jobs arrive dynamically over time. The scheduler does not have complete information about the tasks that will be encountered in the future. Some of the papers also deal with computational complexity issues [12], and others are more applied and focus on real-world problems [30]. Scheduling problems can be found in a wide range of applications across various fields such as transportation, manufacturing [18] and healthcare [21]. To sum up, the field of scheduling is a rich area of research that has generated a wealth of knowledge and practical applications.

The complexity of scheduling problems comes from the fact that they can take many different forms with different characteristics. To describe a deterministic schedule problem accurately, the standard notation in literature is the triplet $\alpha|\beta|\gamma$ where α represents the machine environment, β the processing characteristics and the constraints and γ the objective. The term job is a unit of work to be completed. For example, in a manufacturing area, a job is the production of a specific product on one or several machines.

1.2.1. Machine environment

Machine environment α refers to the characteristics of the machines that are involved in the process. These characteristics may include factors such as the number of machines available, their processing speeds and capacities. Some possible machine environments specified in the α field are listed below. These definitions come from

the book of reference of Pinedo [31].

Single machine (1). *The case of a single machine is the simplest of all possible machine environments and is a special case of all other more complicated machine environments.*

Flow shop (Fm). *There are m machines in series. Each job has to be processed on each one of the m machines. All jobs have to follow the same route, i.e., they have to be processed first on machine 1, then on machine 2, and so on.*

Job shop (Jm). *In a job shop with m machines, each job has its own predetermined route to follow.*

Open shop (Om). *There are m machines and there are no restrictions with regard to the routing of each job through the machine environment.*

1.2.2. Processing characteristics

Some examples of processing characteristics and constraints that are specified in the β field are listed below. These definitions come from the book of reference of Pinedo [31].

Preemptions ($prmp$). *Preemptions imply that it is not necessary to keep a job on a machine, once started, until its completion. The scheduler is allowed to interrupt the processing of a job (preempt) at any point in time and put a different job on the machine instead.*

Precedence ($prec$). *Precedence constraints may appear in a single machine or in a parallel machine environment, requiring that one or more jobs may have to be completed before another job is allowed to start its processing. There are several special forms of precedence constraints: if each job has at most one predecessor and at most one successor, the constraints are referred to as chains.*

Recirculation ($rcrc$). *Recirculation may occur in a job shop or flexible job shop when a job may visit a machine more than once.*

1.2.3. Objective

The objective to minimize γ is always a function of the completion times of the jobs. A common objective is the **makespan** (C_{max}), defined as the maximum completion time of jobs. It is equivalent to the completion time of the last job to leave the system. Another common objective is the **total weighted completion time** ($\sum w_j C_j$), defined as the sum of weighted completion times of all jobs. The weight w_j denotes the importance of job j relative to the other jobs in the system. There exist other objectives relative to the lateness of a job. The lateness is defined as the difference between its completion time and its due date. The due date of a job is the deadline for delivering a job to the customer.

1.3. Problem statement

In this thesis, we will focus on a specific machine environment, the *Job Shop* (Jm). A job shop is composed of a set of n jobs $J_1, ..., J_n$ and m different machines $M_1, ..., M_m$. Each job J_i follows a predetermined route on machines. A route consists of a number n_i of operations $O_{i,1}, ..., O_{i,n_i}$ which have to be processed in this particular order. The Gantt chart in Figure 1.2 illustrates these relations.

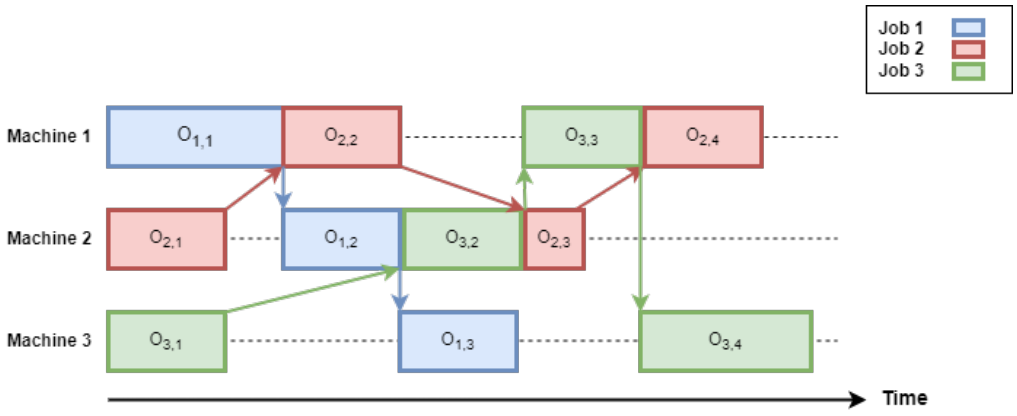


Figure 1.2.: Gantt chart of a job shop problem

This ordered list of operations is called a machine sequence. Each operation O_{ik} is assigned to a machine and must be processed during a processing time p_{ik} without interruption. There is no preemption ($prmp$), the operations cannot be interrupted

at any point in time once it is processed. Each job may contain operations that are processed on the same machine, known as re-circulation ($rcrc$). We assume that each machine can process only one operation at a time and that each operation can be processed only on a unique machine. There is no symmetry. The scheduling problem is subject to precedence constraints ($prec$) in chain form. In addition to the order on the operations, there can be orders on jobs. A job can have to be completed before another job is allowed to start its machine sequence. The objective one considers is the total weighted completion time, denoted by the sum of the weighted completion time of each job. The problem one is concerned with can be characterized by the triplet $Jm \mid prec,\ rcrc \mid \sum w_j C_j$. This triplet refers to a job shop with m machines. There is recirculation, so a job may visit a machine more than once and there are precedence constraints. The objective is the minimization of the total weighted completion times.

Even if some aspects of the environment of TOOWHE are not covered (e.g. personnel, availability of raw materials, etc.), an effective model for the considered Job Shop problem could also be useful to solve problems with more complex requirements.

1.4. Related work

The job shop problem is an NP-hard problem [25], meaning that there is no known algorithm that can solve it in polynomial time. It has also proven to be computationally challenging and it is among the most studied combinatorial problems [13]. The work of Fisher and Thompson [16] has contributed to the fame of this problem. They described two small-size problem instances, one with 20 jobs to be scheduled on 5 machines and another one with 10 jobs on 10 machines. The first one was solved ten years later and the other took more than 25 years to be solved optimally in 1989 [11]. They solved the problem with a *Branch and Bound* method based on the disjunctive graph model.
Branch and bound methods [8, 7] are exact methods. It is based on a complete enumeration of the possible schedules organized in a tree structure with a pruning technique. At each branch, a subset of possible schedules is defined using different

strategies. For example, we can choose an operation of a job to be scheduled or add a constraint on the order of the operations. To evaluate each subset, a lower bound of the solution is calculated by analyzing the choices made in a specific branch. This lower bound provides an optimistic estimate of the potential solutions that can be generated by the branch. If the lower bound for a branch exceeds the current best solution, the entire branch can be pruned from the tree.

The Branch and Bound methods can take a very long time to find an optimal schedule even for small problems like 20 machines and 20 jobs [31]. According to Zhang et al. [42], they do not solve instances larger than 250 operations in a reasonable time.

To deal with larger problems, it was, therefore, necessary to develop heuristic methods that concentrate the search for promising branches. One of the most famous heuristic procedures for scheduling is the *Shifting Bottleneck Heuristic* [1]. This procedure solves the problem by iteratively identifying the machine that creates a bottleneck and solving the schedule optimally on that unique machine using the one-machine schedule method by Carlier [10].

There exist other methods that are useful in dealing with scheduling problems in practice such as local search methods. These are heuristic methods that can find a reasonably good solution in a relatively short time [31].
A first example is *Simulated annealing* [22]. It is an iterative optimization algorithm, inspired by the physical annealing process of solids [33]. It works by starting with an initial schedule and iteratively modifying it by making small random changes. The algorithm evaluates the new solution and accepts it if the objective decreases. It also accepts solutions that increase the objective with a certain probability. This probability decreases over time. By accepting these solutions with a probability, it allows the algorithm to escape local optima.
A second example is the *Tabu search* [14, 28, 29, 39]. It is also an iterative optimization algorithm based on an acceptance-rejection criterion. This criterion is no longer based on a probabilistic process but rather on a deterministic one. A "tabu list" is used to avoid visiting states that have already been viewed and undoing changes that have already been made. It also searches for better solutions in the neighborhood of the current one by making small moves and may accept

schedules that are possibly worse than the previous solution. The new contribution of this algorithm is that the moves that lead to previously visited states are forbidden and kept in the tabu list. Tabu search can escape local optima and converge to a global optimum by allowing the algorithm to explore solutions that may initially seem worse.

Another type of local search algorithm is *genetic algorithm* (GA) [31]. This type is more general and abstract than simulated annealing and tabu-search methods. Genetic algorithms are methods that mimic natural evolution. A GA considers a population of individuals, each of which encodes a potential solution to the problem. The quality of an individual is defined by the fitness function. In the context of scheduling, the fitness function measures the objective. The search progress is achieved by changing the population, generation after generation. A new population is produced by selecting the parents based on some characteristics (selection), combining the two parents to produce a new offspring (crossover), and modifying the newly generated offspring to set up unexplored genetic materials (mutation). GA differs from simulating annealing and tabu-search methods in that at each step, several schedules are generated and passed to the next step. However, simulating annealing and tabu-search can be viewed as a specific case of GA with a population of one individual.

1.5. Benchmarks from the literature

Benchmarks are used to evaluate in a standardized way the performance of different approaches to solve a particular problem. Benchmarks typically consist of a set of instances that represent different job shop problems. Each instance specifies the number of jobs, the number of machines, and the processing times for each job on each machine. These instances differ in size and difficulty.
The benchmark introduced by Fisher and Thompson [16] was seen as a challenge and contributed to the fame of the Job Shop problem [13]. After this first challenging benchmark, the scientific community created many other benchmarks [2, 3, 15, 23, 27, 38, 40, 41]. Although not strictly required, the scheduling community conventionally creates rectangular instances for jobs and machines.

This implies that each job assigns exactly one operation for each machine, and each machine has exactly one operation for each job assigned to it. The largest instances are created by Taillard [40]. They consist of 100 jobs and 20 machines, for a total of 2000 operations.

However, our scheduling problems are different from all the instances of these available benchmarks. They do not follow a rectangular form. In our model, a job can have a number of operations different from the number of machines and a job can be processed more than once on the same machine (recirculation). Due to these variations in problem structure and size, we cannot directly compare the results obtained from our instances with those of the existing benchmarks.

1.6. Contributions

The challenge of this thesis lies in tackling the size of the job shop problems at hand. Table 1.1 provides an overview of the problem sizes for three instances, with the number of jobs, operations (including those on machines with infinite capacity), and machines (including machines with infinite capacity).

In our research, we aim to push the limits of established optimization techniques, including simulated annealing, branch and bound, and dives. By subjecting these methods to industrial-size problems, we can evaluate their performance and identify their strengths and limitations.

Moreover, we use an algorithm we call the ranking method. It involves solving a relaxation of the problem with a linear program. The solution gives a lower bound and an order in which we can schedule the operations. Through extensive experimentation, we determine the feasibility and effectiveness of this approach in dealing with large job shop problems.

We performed the initial testing of our algorithms on the first instance of the job shop problem, and the detailed results and analysis of these tests can be found in the main text of the thesis. To further validate our results, we performed additional tests on the second and third instances. The comprehensive results for these instances are presented in the appendix of the thesis. By testing and evaluating our algorithms

on multiple instances, we gain a deeper understanding of their performance and limitations across a range of problem sizes and complexities.

Firstly, we develop two algorithms that compute the completion times of the operations based on a given order. This can be challenging, as there are many constraints to consider. Each of the methods we evaluate (except the ranking method) tries to find a better schedule by modifying the order of the operations. It is so necessary to compute the completion times of the operations efficiently to evaluate each of the orders generated all along the algorithms.

Secondly, we evaluate the simulated annealing algorithm. It starts with a streamlined schedule where each job is organized one after the other in the order given in the data. The objective value of this initial schedule is 8,286,958. We explore three different neighborhoods, starting from simpler to more complex strategies. Among them, only the first neighborhood exhibits low objective schedules without significant variability in the tests. The algorithm with an appropriate neighborhood demonstrates promise, achieving a best objective value of 3,185,492 within a seven-minute execution time.

However, the ranking method quickly outperforms the simulated annealing algorithm, achieving an objective value of 2,984,850 in less than a second. When one family of constraints in the linear program is applied to each set of operations, the ranking method can find a solution that is less than two times the optimal solution. However, due to the problem's size, the family of constraints can only be applied to partial sets of operations. After studying several specific sets, we find that the ranking method consistently produces schedules with objectives of about 3,000,000 within seconds, regardless of the set chosen. The lower bounds, though, are really distinct and the higher ones reach values of approximately 800,000.

The high lower bound obtained from the ranking method serves as a baseline for the branch and bound algorithm. In this algorithm, the branching is done by selecting two operations and imposing their order. After each branching, we are thus in a subproblem where two additional operations are ordered. Despite testing various strategies to select operations and expand nodes, we were unable to improve the pruning process. The pruning process happens when the lower bound exceeds the current best objective, meaning that no better schedule can be found in this branch.

The lack of pruning reduces the branch and bound algorithm to a simple exact method where all the solution space is explored. The branch and bound method without pruning cannot handle the complexity and size of industrial-scale problems effectively. Additionally, the need to establish in totality and solve a linear program at each node further contributes to the algorithm's limitation by slowing it down.

To address these limitations, we introduce the dive approach, which incorporates the concept of a warm start. In the dive approach, the same model is maintained between nodes within the same dive. At each node, a new constraint that imposes a random order on two selected operations is added to the model. The dive approach's efficiency enables deeper exploration of the search tree. By analyzing different strategies for operation selection, we achieve good schedules in less than 100 dives within an hour. This makes the dive approach a promising alternative. By starting the search with a good schedule obtained from the ranking method in less than a second, we found a schedule with the best objective value of 2,400,752.

To compare the simulated annealing algorithm with the dive approach, we perform the simulated annealing algorithm using the initial schedule obtained from the ranking method. The ranking method provides an initial schedule with an objective value of 2,984,830, which improves to 2,092,161 after applying the simulated annealing algorithm. Furthermore, we investigate refining the dive approach schedule using the simulated annealing algorithm. The dive approach yields an initial schedule with an objective value of 2,400,752, which is further optimized to 2,009,964 through the application of simulated annealing. The simulated annealing algorithm is thus a valuable tool for refining and enhancing schedules with a low initial objective. Mixing the methods leads to better schedules than applying a method alone, and this for the three instances of the problem.

In conclusion, our findings highlight the limited performance of the branch and bound method when confronted with the size of the problems considered. Conversely, the simulated annealing algorithm, ranking method, and dive approach demonstrate their effectiveness in generating reliable initial schedules. For optimal results, we recommend using one of the initial schedules obtained from the ranking method or dive approach, followed by applying the simulated annealing algorithm

to further refine and improve the schedules.

Instance	Number of		
	jobs	operations	machines
1	828	6057 (4392)	51 (17)
2	866	6174 (4514)	48 (16)
3	268	1049 (515)	35 (1)

Table 1.1.: Problem Sizes for Three Instances: Number of Jobs, Operations (Including Operations on Machines with Infinite Capacity), and Machines (Including Machines with Infinite Capacity)

1.7. Outline of the work

The structure of the report is as follows:

- **Chapter 1 - Introduction**
- **Chapter 2 - Methodology:** This chapter covers the methods developed for this thesis and describes the structure and implementation of different algorithms to solve the problem.
- **Chapter 3 - Evaluation:** Demonstration and discussion of the results of the different methods.
- **Chapter 4 - Conclusion:** Brief summary and conclusion of what was done in this thesis, and quick overview of some potential future work from this thesis.

2. Methodology

2.1. Data

The data delivered by TOOWHE about the supply chain of one of their client are given in CSV files. The first CSV file contains all the operations for each job. The j^{th} operation $O_{i,j}$ of the job J_i is defined by the machine $M_{i,j}$ on which it has to be processed, along with its preparation time and its realization time. The processing time $p_{i,j}$ is defined by the sum of the last two. The j^{th} operation of the job J_i is then the tuple $(i, j, m_{i,j}, p_{i,j})$.
The second CSV file contains the names of all the machines involved in the process and their corresponding capacity. The capacity of a machine, which can either be 1 or infinity, is the maximum number of operations that it can handle at the same time. The machines with infinite capacity, such as SST (sous-traitance) and MAG (magasin) do not require scheduling. But the operations on these machines still impact the timing of the operations of the same job.
To deal with the infinite capacity, the j^{th} operation of the job J_i can be expressed as the tuple $(i, j, m_{i,j}, p_{i,j}, a_{i,j})$, where $a_{i,j}$ is the sum of processing times of operations on a machine with infinite capacity that come after the j^{th} operation of the job J_i and before the next operation of the same job on a machine with a finite capacity.
In some cases, a job may start with an operation on a machine with infinite capacity. In such cases, the j^{th} operation of the job J_i can be expressed as a tuple $(i, j, m_{i,j}, b_{i,j}, p_{i,j}, a_{i,j})$, where $b_{i,j}$ is the sum of the processing times of operations on a machine with infinite capacity before the operation of the same job on a machine with a finite capacity. This value can be different from zero only for the first operation of a job.

2.2. Definition of a Schedule

A schedule consists of two parts: the order in which operations are processed on each machine and the corresponding completion time for each operation. While a schedule is crucial for effective resource allocation and productivity, it is essential to have an initial schedule to begin the planning process. The initial schedule serves as a starting point for further optimization and improvement. It provides a baseline from which companies can evaluate and refine their scheduling decisions. Therefore, for all algorithms we will test, we require an initial order of operations and an efficient algorithm to accurately calculate the completion times whenever the order is altered.

2.2.1. Order of the operations

Finding a first order can be easily done by sequentially examining each operation of each job. We iterate over all the jobs and add the j^{th} operation of the job J_i, denoted by $(i, j, b_{ij}, p_{ij}, a_{ij})$, to the list of operations of the corresponding machine. This amounts to programming one job after the other.
This deterministic approach ensures that the schedule is feasible, i.e. the order of the sequence machine is respected. However, some algorithms like simulated annealing do not require a feasible schedule initially. Therefore, a random schedule can be generated by shuffling the list of operations of each machine, which can be used as the initial schedule for the simulated annealing algorithm.

2.2.2. Completion times of the operations

Although the order of the operations is sufficient to define a schedule, it is necessary to compute the completion times to calculate the objective, assess the schedule's quality and evaluate its effectiveness. While establishing the order of operations is a straightforward task, the computation of completion times requires additional considerations and cannot be derived trivially from the order of operations.

Two approaches were developed to get the completion times of the operations, one involving a linear program (LP) and the other without solving a LP. This step

must be done as fast as possible to not limit the number of solutions that can be explored within a given time.

When using a linear program, a model is created with two variables: *completion* and *end*. The variable *completion*, notated $C_{M_i,j}$, represents the completion time of each operation j on the specific machine M_i. If the operation j of the job J_i on machine m_{ij} is followed by other operations on machines with infinite capacity, i.e. $a_{ij} \neq 0$, the processing times of these next operations do not affect the completion time of the machine m_{ij}. These operations are not carried out on it. The variable *end* of size 1xn represents the n completion times of the n jobs. end_i, the completion time of the job J_i, is determined by summing the completion time of the last operation k of the job J_i and the value a_{ik}.
The objective is defined by the weighted sum of the values of the variable *end* where all the weights are equal to 1:

$$\sum_{i=1}^{n} w_i * end_i \quad \text{where } w_i = 1 \ \ \forall i.$$

The model has two types of constraints:

- The precedence constraints of the operations of a single job (illustrated in Figure 2.1):

$$C_{i,j+1} \geq C_{i,j} + a_{i,j} + b_{i,j+1} + p_{i,j+1} \quad \forall j \in \{1, ..., n_i - 1\} \quad \forall i \tag{2.1}$$

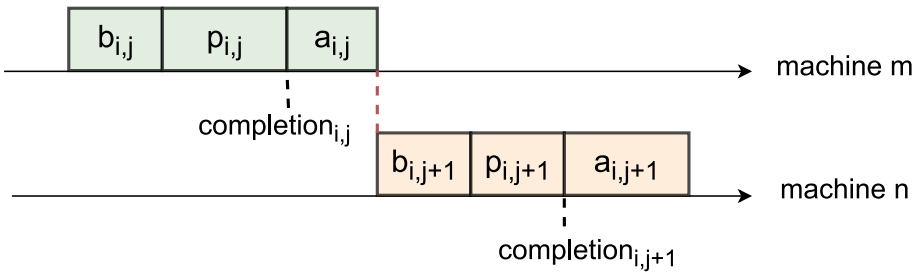


Figure 2.1.: Illustration of a precedence constraint of the job J_i

- The constraints on the capacity of the machines (illustrated in Figure 2.2):

If the operation l of the job J_k follows the operation i of the job J_i in the schedule of the machine $m_{k,l}$ $(= m_{i,j})$,

$$C_{k,l} \geq C_{i,j} + p_{k,l} \quad if m_{k,l} = m_{i,j} \tag{2.2}$$

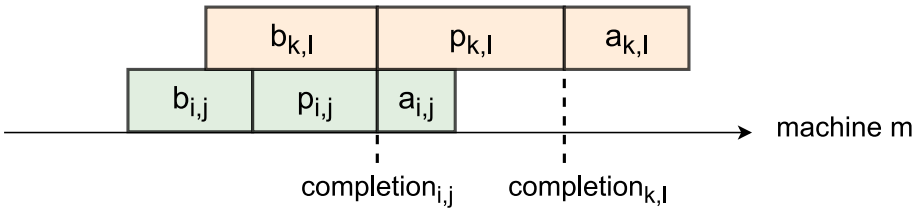


Figure 2.2.: Illustration of a constraint on the capacity of the machine m

When not using a linear program, the completion time of each operation can be determined by sequentially examining the schedule and noting the completion time of the first available operation encountered. An operation is considered available if it is either the first operation of a job or if the previous operation in the sequence has already been scheduled.
In other words, we have two counters: one for the jobs and one for the machines. While all the operations of all the jobs are not scheduled, we iterate over the machines. At each current operation $(i, j, b_{i,j}, p_{ij}, a_{i,j})$ of the machine (determined by the counter), if the number of the operation j corresponds to the counter of the job i, the operation can be scheduled. This means that all the previous operations of the job and on the machine have been scheduled and the two counters are incremented.

The two methods produce the same completion times, which validates the results. However, we consider the second technique as better because it is faster.
The two methods were tested on n different orders, where n represents the total number of jobs. To generate these n distinct orders, the deterministic method outlined in subsection 2.2.1 is initiated with a different job as the starting point for each schedule. For instance, the first schedule begins with job 1 and concludes with job n, while the tenth schedule starts with job 10, processes job n, and then proceeds from job 1 to job 9, adhering to a circular order. The results of this test

are reported in Table 2.1.

	With LP	Without LP
Time for the 828 schedules (in s)	72.7613	4.7271
Average time for 1 schedule (in s)	0.0879	0.0057

Table 2.1.: Comparison of Execution Time for the 2 Methods to Get the Completion Times

The second method of computing completion times is not only more efficient but also more adaptable compared to the linear program approach. This adaptability becomes particularly useful when we are confronted with infeasible schedules that may arise during the simulated annealing (SA) algorithm.

When generating a neighborhood in the SA algorithm, it is possible to encounter new schedules that are infeasible, meaning they violate certain constraints and cannot be executed. This poses a challenge because the linear program, used in the first method, fails to find a solution to the completion times when it is confronted with such infeasible schedules. Consequently, the objective cannot be computed, and the algorithm becomes effectively blocked, hindering further progress.

However, the second technique can be modified to address this issue. Suppose no more available operations can be scheduled. In that case, we can schedule the next operation for a particular machine and add a penalty to the job and the machine by increasing the processing time of this operation. To determine the machine for scheduling the next operation, a criterion based on the mean order of operations is used. The mean order is computed as the average of the order of operations assigned to the specific machine. For instance, consider a scenario where a machine processes the fourth operation of one job and the second operation of another job. In this case, the mean order would be 3. By selecting the machine with the lowest mean order, we can expect to schedule an operation that disrupts the existing blockage.

This relaxation of the scheduling process within the simulated annealing algorithm proves highly advantageous. The ability to generate neighboring solutions becomes more effective as it acknowledges the potential occurrence of infeasible schedules. From such infeasible schedules, feasible neighbors can be derived, allowing a larger exploration of the solution space.

2.3. Simulated Annealing

Simulated annealing [22] is a widely used stochastic algorithm for solving hard combinatorial optimization problems such as scheduling. The algorithm is inspired by the process of annealing in metallurgy, where *a material is heated to a temperature that permits many atomic rearrangements and then slowly cooled down to decrease defects until it freezes into a good crystal* [33].

In simulated annealing, the algorithm starts with an initial schedule/solution and an initial temperature, the maximum temperature. We iterate while the temperature is larger than the minimum temperature and while the maximum number of iterations is not reached. At each iteration, the algorithm searches for better solutions by making random changes in the order of the operations of the current schedule. These random changes define the neighbor of the current solution.

The algorithm always accepts changes that decrease the objective. To avoid being trapped in local minima, the algorithm accepts changes that raise the objective with a certain probability. This probability decreases gradually during the search. The probability that the schedule is accepted is $p = exp(\frac{\Delta}{T})$, where T is the current temperature and Δ is the difference between the objective of the previous schedule and the objective of the new one. The difference is positive since the new schedule raises the objective.

To introduce randomness into the algorithm, random numbers uniformly distributed between 0 and 1 are commonly employed. One such number is selected and compared with the probability value p. If the random number is lower than p, the algorithm keeps the new solution; otherwise, the previous solution is used to start the next step.

Moreover, at each iteration, if the new schedule has a lower objective than the best schedule we currently have, the new schedule becomes the best one. Once all the iterations are done or once the temperature is low enough, the algorithm returns the best schedule encountered during the search. The pseudo-code of this algorithm is provided in Algorithm 1.

Algorithm 1 Simulated Annealing

$T \leftarrow T_{max}$
$i \leftarrow 0$
$curr \leftarrow$ INIT()
$obj\_curr \leftarrow$ OBJECTIVE($curr$)
$best \leftarrow curr$
$obj\_best \leftarrow obj\_curr$
while $T > T_{min}$ **and** $i \leq nb\_iter$ **do**
    $next \leftarrow$ NEIGHBOUR($curr$)
    $obj\_next \leftarrow$ OBJECTIVE($next$)
    if $obj\_next < obj\_curr$ **then**
        $curr \leftarrow next$
        if $obj\_next < obj\_best$ **and** ISFEASIBLE($next$) **then**
            $best \leftarrow next$
    else if RANDOM() $< exp(\frac{obj\_curr - obj\_next}{T})$ **then**
        $curr \leftarrow next$
    $T \leftarrow T * cooling\_rate$
    $i \leftarrow i + 1$
return $best$

2.3.1. Design of the neighborhood

The effectiveness of simulated annealing depends on the neighborhood structure of a schedule. Two schedules are neighbors if one can be obtained through a well-defined modification of the other by definition [31].

First neighborhood

A first simple neighbor of a schedule can be designed by interchanging a pair of adjacent operations within a window of a certain size $2 * w$. To begin, we randomly select a machine that has more than one operation. Next, we choose a starting operation k at random from all the operations on that machine. Assuming that the machine has a total of t operations, the second operation is then randomly selected from the range $[max(1, k - w),\ min(t, k + w)]$, where $2 * w$ is a predefined window size. This ensures that the second operation is adjacent to the first one within the specified window, where the window spans from $k - w$ to $k + w$ and is constrained by the total number of operations on the selected machine.

After swapping the two operations, it is not guaranteed that the resulting schedule

will satisfy all the precedence constraints of the job shop problem. To address this issue, the second method described in subsection 2.2.2 can be used, in its modified version which involves introducing a penalty. If the penalty is sufficiently large, the objective of the new schedule will be higher than the objective of the previous one since it violates a constraint. The new schedule will be accepted according to the previously defined probability. However, even if the objective of the new schedule may be lower than the best objective, it cannot be considered as the new best solution because it is infeasible, as shown in pseudo-code 1.

Second neighborhood

The second neighborhood is an extension of the first neighborhood, where instead of swapping two operations, we swap three operations within a window of a certain size. This neighborhood is created by modifying the procedure described previously.

Similar to the first neighborhood, we begin by randomly selecting a machine that has more than two operations. However, if a machine has exactly two operations, we make an exception and swap these two operations directly. This ensures that even machines with only two operations are considered in the swapping process.

For machines with more than two operations, we proceed as follows: we choose a starting operation, denoted as k, at random from all the operations on that machine. The second operation is randomly selected from the range $[max(1, k - w),\ min(t, k + w)]$, where w represents the window size. In the second neighborhood, we introduce a third operation, which is also randomly selected from the range $[max(1, k - w),\ min(t, k + w)]$.

By swapping three operations instead of two, the second neighborhood allows for a more extensive exploration of the solution space. It increases the difference between two neighboring schedules that can be obtained from a given schedule. This enhanced exploration can be beneficial in finding better solutions or escaping from local optima in the optimization process.

While increasing the number of swapped operations expands the search space, it also introduces a higher risk of violating precedence constraints. Therefore, similar to the first neighborhood, it is necessary to apply a mechanism to handle infeasible

schedules, such as the penalty-based method mentioned in Subsection 2.2.2.

In summary, the second neighborhood in simulated annealing swaps three operations within a window, providing a neighbor more different than the initial one. This extended neighborhood can be advantageous in improving the optimization process by allowing the algorithm to discover new promising solutions.

Third neighborhood

The third neighborhood expands upon the previous neighborhoods by considering a larger number of operations to swap. This neighborhood is the most complex out of the three because we select the neighbor based on an optimization problem. In this case, we select four operations from the schedule and evaluate all possible combinations of these four operations. The combination that yields the lowest objective value, considering the objective is to be minimized, is chosen as the new schedule.

By increasing the number of operations involved in the swapping process, the third neighborhood allows for even greater exploration of the solution space compared to the previous neighborhoods. Considering all 24 possible combinations of four operations provides a more comprehensive search, potentially leading to the discovery of better solutions.

More complex neighborhoods can be useful for several reasons. First, they promote intensification in the search process. Intensification refers to focusing on promising areas of the solution space where better solutions are more likely to be found. By evaluating all possible combinations of four operations, the algorithm systematically assesses various combinations to identify the one with the lowest objective value. This intensification approach can lead to faster convergence towards better solutions.

Second, more complex neighborhoods can be particularly beneficial for highly complex optimization problems or instances where the solution space is vast and challenging to explore thoroughly. By considering a larger neighborhood, the algorithm can cover a broader range of potential solutions, increasing the chances of finding optimal or near-optimal solutions.

However, it is worth noting that the more complex neighborhood also comes with

increased computational costs due to the evaluation of multiple combinations. Therefore, its use should be carefully considered, taking into account the problem's complexity and available computational resources.

In summary, the third neighborhood selects four operations and evaluates all possible combinations to identify the one with the lowest objective value. This approach allows us to make an extensive exploration of the solution space and can be beneficial for complex optimization problems or instances with vast solution spaces even though it requires high computation costs.

2.4. Ranking Method

In this section, we present a method we call the "Ranking Method" for solving our job shop scheduling problem. It provides a systematic approach to determine the sequence in which the operations of each job should be processed.

To begin, we develop an equivalent problem for the single-machine scheduling problem. Next, we consider a relaxation problem derived from the single-machine scheduling problem with precedence constraints. By solving this relaxation problem, we obtain completion times for the operations that respect the precedence constraints. However, these completion times do not account for the capacity constraints of the machines. Consequently, the obtained completion times serve as a lower bound on the objective value. We then explore the relationship between the optimal value and the lower bound obtained from the relaxation problem.

Subsequently, we adapt the relaxation of the single-machine scheduling problem to our job shop scheduling problem. This adaptation enables us to leverage the insights and techniques from the single-machine problem and apply them to the more complex job shop scenario.

The completion times obtained from the relaxation can be interpreted as a ranking or order in which the operations can be processed. By sorting the completion times in increasing order, we can create a feasible schedule for the job shop problem. Finally, the real completion times, taking into consideration the capacity constraints

of the machines, can then be computed using one of the two approaches developed previously in subsection 2.2.2.

2.4.1. A single-machine scheduling problem

According to Bertsimas and Weismantel [6, pp. 95-98], let's consider the single machine scheduling problem with a set $V = \{1, ..., n\}$ and follow their developments. Each job $i \in V$ has its processing time p_i. The objective is to minimize the total weighted completion time $\sum_{i \in V} w_i C_i$ where $w_i \geq 0, i \in V$ are given weights. The schedule needs to satisfy the following constraints to characterize the completion times:

- Constraints on the processing time:

$$C_j \geq p_j, \quad j \in V, \tag{2.3}$$

- Constraints stating that in every schedule, either job k is processed before job j, or job j is processed before job k:

$$C_j \geq C_k + p_j \quad \text{or} \quad C_k \geq C_j + p_k \quad j, k \in V,\ j \neq k.$$

A schedule is nonidling if the machine is used without any idle time. This means that the machine is processing a job at all times during the schedule. In a nonidling schedule, the completion time of a job i is $C_i^* = \sum_{k=1}^{i} p_k$.
Knowing that all nonidling schedules are optimal [6], the completion times $C_1, ..., C_n$ of the n jobs satisfy:

$$\begin{aligned}
\sum_{i=1}^{n} p_i C_i &\geq \sum_{i=1}^{n} p_i C_i^* \\
&= \sum_{i=1}^{n} p_i \sum_{i=1}^{n} p_k \\
&= \frac{1}{2} \sum_{i=1}^{n} p_i^2 + \frac{1}{2} \Big(\sum_{i=1}^{n} p_i \Big)^2
\end{aligned}$$

We can consider any set $S = \{i_1, ..., i_s\} \subset V$ of jobs and the objective function where

$$w_i = p_i \quad \forall i \in S,$$
$$w_i = 0 \quad \forall i \notin S.$$

The schedule in which we first process the job according to the ordering $i_1, ..., i_s$ of the element of S and then the jobs not in S is optimal from the following statement:

If 1,2,...,n is the order according to which the jobs in an optimal feasible schedule are processed. Then,

$$\frac{w_1}{p_1} \geq \frac{w_2}{p_2} \geq ... \geq \frac{w_n}{p_n}.$$

Therefore, for all nonidling schedules, the completion times satisfy

$$\begin{aligned} \sum_{i=1}^{n} w_i C_i &\geq \sum_{i=1}^{n} p_i C_i \\ &= \sum_{j=1}^{s} p_{i_j} \sum_{k=1}^{j} p_{i_k} \\ &\geq \frac{1}{2} \sum_{i \in S} p_i^2 + \frac{1}{2} \Big(\sum_{i \in S} p_i \Big)^2, \quad \forall S \subset V. \end{aligned} \tag{2.4}$$

The completion times are given exactly by the inequalities (2.4) [32] and the single machine scheduling problem is therefore equivalent to

$$\begin{aligned} min \quad & \sum_{i=1}^{n} w_i C_i \\ s.t. \quad & \sum_{i \in S} p_i C_i \geq \frac{1}{2} \sum_{i \in S} p_i^2 + \frac{1}{2} \Big(\sum_{i \in S} p_i \Big)^2, \qquad S \subset V. \end{aligned} \tag{2.5}$$

We can now consider the single machine scheduling problem with precedence constraints according to Bertsimas and Weismantel [6, pp. 452-53]. The following linear optimization problem (2.6) of the single machine scheduling problem with

precedence constraints is a relaxation of the problem and provides a lower bound on the optimal objective function value:

$$\begin{aligned} min \quad & \sum_{i=1}^{n} w_i C_i \\ s.t. \quad & \sum_{i \in S} p_i C_i \geq \frac{1}{2} \sum_{i \in S} p_i^2 + \frac{1}{2} \Big(\sum_{i \in S} p_i \Big)^2, && S \subset V, \\ & C_j \geq C_i + p_j, && (i,j) \in \mathcal{A}. \end{aligned} \tag{2.6}$$

Where the precedence constraints among jobs are described by a directed graph $G = (V, \mathcal{A})$, where

$$\mathcal{A} = \big\{ (i,j) \mid \text{job } i \text{ must be processed before job } j \big\},$$

We can sort the optimal completion time C_i^* given by the relaxation (2.6) and create a feasible schedule by processing the jobs in the same order. The value of the objective, the weighted sum, is Z_H.
If the inequality (2.4) is applied on all possible sets S, the value Z_H is lower or equal to two times the value of the relaxation Z_{LP} [35]:

$$\frac{Z_H}{Z_{LP}} \leq 2 \tag{2.7}$$

In summary, the relaxation problem provides a lower bound Z_{LP} on the optimal objective value Z_{opti}, and the feasible schedule generated from the optimal completion times serves as an upper bound Z_H:

$$Z_{LP} \leq Z_{opti} \leq Z_H \tag{2.8}$$

The relationship between these bounds highlights the potential optimality gap in the solution space and guides us in developing effective scheduling strategies for the job shop problem. Moreover, the relaxation problem allows us to find a solution that is less than two times the optimal solution when the inequality (2.4) is applied

on all possible sets S. Indeed, from the inequalities 2.7 and 2.8 we can write:

$$Z_H \leq 2 * Z_{opti}. \tag{2.9}$$

2.4.2. A single machine scheduling problem to our job shop problem

The single machine scheduling problem described in the previous section with its lower bound can be adapted to our job shop problem.
Let's consider the linear optimization problem that minimizes the total weighted completion time with the following constraints, inspired by the previous section 2.4.1:

- Constraints on the processing time for each operation j of each job i (see 2.3):

$$C_{i,j} \geq b_{i,j} + p_{i,j},$$

- Precedence constraints (see 2.1):

$$C_{i,j+1} \geq C_{i,j} + a_{i,j} + b_{i,j+1} + p_{i,j+1} \quad \forall j \in \{1, ..., n_i - 1\} \quad \forall i$$

- Constraints on the set $\mathcal{S}$ of operations that are processed on the same machine (see 2.6):

$$\sum_{i \in S} p_i C_i \geq \frac{1}{2} \sum_{i \in S} p_i^2 + \frac{1}{2} \Big(\sum_{i \in S} p_i \Big)^2$$

  Imposing this constraint on every set $\mathcal{S}$ of operations is intractable if we want to solve the problem with a linear program. Considering only pairs of operations on each machine already result in 91,417 constraints, and including all possible triples would introduce 4,590,413 additional constraints.
  A study of the set of operations that are in $\mathcal{S}$ will be done in chapter 3.

Solving this linear optimization problem gives completion times of the operations that respect the precedence constraints but not the capacity of the machine. There

is no constraint on the machines such that (2.2). Therefore, the completion times provide a lower bound on the objective value since we cannot do better without violating the precedence constraints. The completion times are the solution to the relaxed problem.
In addition, the completion times can be seen as a ranking, an order in which the operations can be processed. Based on this order, the real completion times (that respect the capacity of the machines) can be computed with one of the methods described in section 2.2.2.

To sum up, the linear optimization problem that minimizes the total weighted completion times, subject to the given constraints, can be solved using a linear program. However, this problem is a relaxation of the actual problem, meaning that the obtained solution provides a lower bound on the optimal objective value.

By solving the relaxation problem, we can determine the completion times for each operation. These completion times represent a ranking or order in which the operations can be processed. To obtain a feasible schedule, we can sort the completion times in increasing order and process the operations accordingly.

While the relaxation problem's objective value serves as a lower bound, the real value of the objective can be computed based on the feasible schedule obtained. This real objective value represents an upper bound on the optimal objective value.

Therefore, through the relaxation problem, we establish a lower bound on the objective value, while the feasible schedule derived from it establishes an upper bound. The gap between these two bounds provides insight into the potential optimality of the solution space and guides us in developing strategies to improve scheduling decisions for the job shop problem.

2.5. Branch and Bound Algorithm

Branch and bound methods [8, 7] are exact methods used to solve optimization problems, including scheduling problems. These methods work by breaking down the problem into smaller sub-problems ("branch" step) and using a tree-based search to

explore the space of possible solutions. Each branch is a subset of possible solutions. To evaluate each subset, a lower bound of the solution is calculated by analyzing the choices made in a specific branch ("bound" step). This lower bound provides an optimistic estimate of the potential solutions that can be generated by the branch. If the lower bound of a branch exceeds the current best solution, the entire branch can be pruned from the tree. Indeed, if the best solution we can reach in this branch is the lower bound and if the lower bound is already larger than the current best solution, the best solution cannot be in this branch.

2.5.1. Branching

Our branch and bound algorithm starts with a first feasible schedule, the root. The upper bound of the root is the objective of the schedule and the lower bound is the bound given by the ranking method.
Each node of the tree can have two children, two different subsets of solutions, by imposing an order between two operations on the same machine. The order can be either k before l or l before k. For each child, the new lower bound is computed with the ranking method with the additional constraint. For the first child, the operation k is processed before the operation l on the machine m and the constraint is written such that

$$C_l \geq C_k + p_l.$$

For the second child, the operation l is processed before the operation k on the machine m and the constraint is written such that

$$C_k \geq C_l + p_k.$$

2.5.2. Selection of the constraint

A constraint is identified by the machine and the two operations. To choose the machine and its two operations, we determine three different deterministic and one non-deterministic strategies:

1. Critical machine and first operations (CM-FO):

We determine the critical machine, which is the machine with the highest sum of processing times for its operations. We select the first operation on the critical machine and continue selecting operations until no more can be selected. We then move to the next critical machine and repeat the process. Based on the ranking method's order and completion time values, we select the first successive pair of operations for which their completion times are too close together. So for increasing indices i from 2 to the number of operations scheduled on that machine, if the completion time of operation i is lower than the sum of the completion time of operation $i-1$ and the processing time of the operation i, the pair of operations $i-1$ and i is selected for the constraint.

2. Critical machine and largest violation (CM-LV):
   We choose the critical machine as before and select the two successive operations for which the difference between the completion time of operation i and the sum of the completion time of operation $i-1$ and the processing time of the operation i is the largest.

3. Largest job (LJ):
   The largest job is the one with the most operations. We select the first operation of the largest job for which the completion time is less than the sum of the completion time of the previous operation on the machine and its processing time. The pair of operations is the operation of the largest job and the previous operation on the machine.

4. Critical machine and one of the 5 largest violations (CM-5LV): We choose the critical machine as before and select the five pairs of successive operations for which the difference between the completion time of operation i and the sum of the completion time of operation $i-1$ and the processing time of the operation i are the largest. We then choose randomly among these five pairs.

Each time a new constraint is added, the ranking method returns another ranking and the lower bound increases. The ranking is modified and two successive operations may not be adjacent anymore. To be sure that the indices will not change with the ranking, we use the initial schedule's set of indices to identify the operations and not the set of indices based on the ranking.

2.5.3. Tree structure and search

For the tree-based search, we use a node structure. A node consists in:

- The lower bound given by the ranking method. Precisely, the lower bound is computed by evaluating the objective on the completion times obtained with the ranking method.
- The upper bound is the objective of the current schedule. The order of the schedule is given by the ranking method and the real completion times are obtained with the second method of subsection 2.2.2.
- The constraint of the branch that creates the node.
- The parent of the node, which is itself a node. Knowing the parent allows us to go up the tree and to have the list of constraints to apply.

The search process begins with the root node, which corresponds to the initial schedule that does not impose any constraint on the operation order. The search algorithm recursively explores the child nodes of the current node to identify optimal solutions. Different expansion strategies can be followed to expand the tree. We use four strategies:

1. Breadth-first search (BFS), where we expand the shallowest node in the fringe. The priority is the depth and the fringe is equivalent to a FIFO queue (First In First Out).
2. Greedy search (GS), where we expand the node with the minimum upper bound. The priority is the upper bound of the node and the fringe is equivalent to a priority queue.
3. Best-node search (BNS), where we expand the node with the minimum lower bound. The priority is the lower bound of the node and the fringe is equivalent to a priority queue.

To implement these strategies, a priority queue called the fringe is used, which orders nodes based on their priorities. The node with the lowest priority is selected first. Initially, the fringe contains only the root node. The tree is expanded as long as the fringe is not empty and the desired number of nodes to explore has not been

reached. At each step, the lowest priority node is removed from the fringe and its two children are generated, representing the next possible constraint. It is possible that no further constraints can be added, in which case no child nodes are added to the fringe, and the next node in the queue is considered. Moreover, it is also possible that adding the next constraint to the problem leads to infeasibility. In this case, the search process is halted. If the lower bound of a child node is higher than the current minimum objective, the node is not added to the fringe, and the search process in that direction is halted. The pseudo-code of the branch and bound algorithm is given in 2.

2.5.4. Lukewarm start

The tree-based search requires solving the optimization problem with a linear program at each node. To speed up the time taken to solve the problem at each node, we can use a technique we call *Lukewarm start* (LS).

The Gurobi solver [20] that solves linear programs offers a parameter called "start," which can be used to set the initial value of a variable. By doing so, the solver can initiate the optimization process closer to the optimal solution, which often reduces the computation time required to obtain the optimal solution. Indeed, a node and its children differ by only one constraint and the solver can focus on finding a feasible solution that satisfies the new constraint only. If the solver starts from zero, it has to explore a larger search space, which can be time-consuming and computationally expensive.

We tested both approaches, with and without a warm start. We ran the branch and bound algorithm for ten nodes for all the strategies that choose the constraint for the branching and search strategies. We tested two different sets: S=[0,0,0,0,0,0,0] (1) and S=[0,0,1,0,0,0,0] (17).

Table 2.2 presents the execution time of the entire algorithm, including the sum of the time taken by each node to optimize and create the model. We can observe the optimization time does not follow a consistent pattern: the warm start technique sometimes takes longer, and sometimes shorter, than the non-warm start approach.

Algorithm 2 Branch and Bound

```
fringe ← priority_queue
schedule ← Init()
ranking, _ ← Ranking(schedule)
up_bound ← Objective(schedule)
low_bound ← Objective(ranking)
priority ← 0
root ← Node(up_bound, low_bound)
fringe.push((root, priority))
obj_best ← obj_curr
best ← root
i ← 0
while len(fringe) > 0 and i ≤ nb_nodes do
    priority, item ← fringe.pop()
    if item.upper_bound < obj_best then
        best ← item
        obj_best ← item.upper_bound
    constraint_list ← constraintList(item)
    constraint_next ← constraintNext(item)
    for j = 1 to 2 do
        ranking, feasible ← Ranking(schedule, constraint_list, constraint_next)
        if feasible then
            low_bound ← Objective(ranking)
            if low_bound < obj_best then
                real ← CompletionTimes(curr, ranking)
                up_bound ← Objective(real)
                priority ←Priority()
                child ← Node(up_bound, low_bound, item, constraint_next + j)
                fringe.push((child, priority))
    end for
    i ← i + 1
return best
```

It is worth noting that the optimization time constitutes only a small fraction of the overall time required to run the branch and bound algorithm. The majority of the time is consumed in creating the model, which is reconstructed entirely at each node. This is because creating a copy of the model for each child is a time-consuming task, and sharing the parent's model with its children is not feasible due to the differing modifications made by each child. Therefore, an alternative approach must be found to reduce the time required for model construction.

Constraint strategy	Search strategy	Total time		Optimization time		Model time	
		LS	no LS	LS	no LS	LS	no LS
S=[0,0,0,0,0,0,0]							
CM-FO	BFS	1.2285	1.1767	0.029	0.029	1.0719	0.9967
	GS	1.189	1.1404	0.039	0.031	1.0745	1.0786
	BNS	1.1943	1.1247	0.035	0.016	1.0547	0.9688
CM-LV	BFS	1.2253	1.156	0.052	0.048	1.0888	1.0002
	GS	1.2244	1.1404	0.034	0.048	1.0977	0.9374
	BNS	1.2253	1.156	0.037	0.0	1.1034	1.0001
LJ	BFS	1.2721	1.2028	0.048	0.015	1.0739	0.9845
	GS	1.2725	1.2028	0.033	0.0	1.0388	0.9846
	BNS	1.2865	1.2028	0.019	0.048	1.0706	0.9688
CM-5LV	BFS	1.3462	1.2686	0.032	0.011	1.0973	1.0323
	GS	1.4237	1.2611	0.026	0.023	1.1515	1.0139
	BNS	1.3173	1.3159	0.025	0.013	1.0544	1.0226
S=[0,0,1,0,0,0,0]							
CM-FO	BFS	4.3217	4.2334	1.656	1.701	4.1027	4.0924
	GS	4.3028	4.2334	1.621	1.642	4.1776	4.0614
	BNS	4.303	4.2334	1.679	1.685	4.1437	4.0301
CM-LV	BFS	4.3924	4.3059	1.707	1.717	4.2244	4.1089
	GS	4.3334	4.2856	1.668	1.747	4.1453	4.1261
	BNS	4.322	4.3343	1.662	1.705	4.2039	4.0823
LJ	BFS	4.3969	4.3115	1.694	1.659	4.1121	4.0766
	GS	4.4126	4.3271	1.757	1.734	4.2065	4.061
	BNS	4.3975	4.3115	1.62	1.732	4.1315	4.0924
CM-5LV	BFS	4.5205	4.3987	1.756	1.637	4.2178	4.0711
	GS	4.4577	4.3965	1.746	1.731	4.1403	4.1448
	BNS	4.4298	4.3747	1.735	1.675	4.1527	4.1151

Table 2.2.: Total time to run the algorithm, sum of the times of each node to optimize the model and sum of the times of each node to create and optimize the model, for the branch and bound algorithm for 10 nodes for the different constraint and expansion strategies and two sets S, with and without the warm start technique.

2.6. Dives

In the context of tree-based search, the term "dive" refers to the process of exploring deeper into the tree structure. This process involves exploring a particular branch of the tree until the search reaches a leaf node or a node that meets the termination

condition of the algorithm.

The dive approach provides a solution to avoid building an entirely new model at each node during a partial exploration of the tree. By keeping the same model within each dive, the algorithm can use the information and constraints already established, saving computational resources and improving efficiency. This practice of keeping the same model throughout the dive is commonly referred to as a "warm start" in optimization.

In this subsection, we introduce an implementation of the dive approach and demonstrate its effectiveness in efficiently exploring numerous nodes in a short amount of time. Additionally, we explore how the dive approach can be integrated with the branch and bound method. Furthermore, we discuss the evaluation of different constraint strategies (see subsection 2.5.2) that determine the selection of the constraint added in a new node.

2.6.1. Implementation of a dive

In the context of our job shop problem, a dive refers to the process of initiating the exploration of a tree at the root and progressively moving deeper by adding a new constraint to the model at each node until either the lower bound exceeds the best-known objective or until there is no more constraint to add to the model.

To begin a dive, a model is built with the initial constraints, including the precedence constraints and the constraints on the set of operations $\mathcal{S}$. This model is maintained throughout the process. At each node, a new constraint is added to the model in order to further refine the search. The specific constraint added at each node is determined by the constraint strategy chosen. The dive proceeds by selecting two operations, labeled i and j, for the constraint according to the strategy. The order in which i and j are applied, whether i comes before j or vice versa, is determined randomly. By repeatedly adding new constraints at each node and exploring deeper into the tree, the search continues until either there are no more constraints to add or the lower bound exceeds the best-known upper bound.

In some cases, the specific order in which the i and j operations are applied may result in infeasibility. This means that the optimization problem at the node

cannot be solved using the current constraints. In such cases, a potential solution is to remove the last constraint that was added to the model and apply the same constraint again but with the i and j operations in the opposite order.

The dive approach, unlike the branch and bound method, allows us to add multiple constraints simultaneously. Therefore, one possible strategy is to select two pairs of successive operations that have the largest difference between the completion time of operation i and the sum of the completion time of operation $i-1$ and the processing time of operation i. This strategy is referred to as CM-2LV.
However, when randomly selecting the order for the operations within each pair, infeasibility issues can arise occasionally. Resolving such infeasibility is more complex because it involves dealing with multiple constraints.
To address this, one potential solution is to modify the order of one or both pairs and/or remove a constraint. These adjustments can help alleviate the infeasibility and allow for a valid solution to be found.

To conclude, maintaining the model throughout the dive can potentially reduce processing time compared to the traditional branch and bound algorithm. At each node of the tree, a new constraint is added to the model and the optimization process is repeated. Unlike the branch and bound algorithm, where the entire model is constructed from scratch at each node, this approach builds on the previous model, potentially saving time and computational resources.

2.6.2. Efficiency of a dive: Comparison of the performances with the branch and bound method

In order to evaluate the efficiency of the dive approach, we conducted experiments to compare its performance to the traditional branch and bound algorithm. Specifically, we measured the processing time required for each method to explore a given number of nodes in the search tree. In order to ensure a fair comparison, we always apply the same order on the two operations i and j selected for the constraint: "operation i before operation j". If this order leads to infeasibility, we apply the order "operation j before operation i".

To evaluate the performance of both methods, we conducted two tests for each of the three strategies using three different sets, namely $\mathcal{S} = [1, 1, 1, 1, 1, 1, 1]$, $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$ and $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$, with a total of 1000 nodes explored in each test. The results of our experiments are presented in Table 2.3, which reports the average processing times for each method.

Upon comparing the processing times of the two methods, it is evident that the dive approach consistently outperforms the branch and bound method for each constraint strategy. This conclusion is supported by the fact that the processing times for the dive method are consistently lower than those for the branch and bound method under all four constraint strategies and for the three sets $\mathcal{S}$. The processing times for the dive method are typically on the order of minutes or even less, whereas the branch and bound method tends to require significantly more time.

The difference in processing times for the branch and bound method is particularly noticeable when dealing with sets $\mathcal{S} = [1, 1, 1, 1, 1, 1, 1]$ and $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$, which introduce a larger number of constraints (respectively 99,463 constraints and 1,665, see Tables A.4 and A.1) that must be defined again at each node. Even when dealing with more complex sets, the dive approach only experiences a slight increase in processing time, remaining highly efficient.

$\mathcal{S}$	Constraint strategy	Time in min	
		BnB	dive
[1,1,1,1,1,1,1]	CM-FO	28:15	1:00
	CM-LV	19:00	1:10
	LJ	33:00	1:19
	CM-5LV	20:20	1:04
[0,0,1,0,0,0,0]	CM-FO	9:42	0:32
	CM-LV	9:21	0:33
	LJ	10:09	0:47
	CM-5LV	4:21	0:29
[0,0,0,0,0,0,0]	CM-FO	3:02	0:21
	CM-LV	3:06	0:22
	LJ	3:38	0:38
	CM-5LV	1:14	0:19

Table 2.3.: Comparison of processing times for dive and branch and bound methods for the four constraint strategies and three sets $\mathcal{S}$.

2.6.3. Integration of dives in branch and bound method

The dive approach offers the advantage of faster exploration of a number of nodes than the branch and bound method, allowing us to execute multiple dives in a relatively short time frame. This enables us to attempt to reach a low objective value. Subsequently, we can exploit the lowest objective value discovered during the dive phase to initiate the branch and bound method.

To accelerate the branch and bound process, it is crucial to implement efficient pruning techniques. Pruning is performed when the lower bound exceeds the upper bound, indicating that further exploration of that particular branch is unnecessary. Consequently, finding an initial upper bound with the dive approach could facilitate quicker pruning in the branch and bound method.

3. Evaluation

In this chapter, we discuss the strategies outlined in Chapter 2. All the Figures and Tables that are in this chapter are derived from the first problem instance. For the sake of clarity, the corresponding Figures and Tables derived from the second and third instances can be found in the respective sections of the appendix.

3.1. Simulating Annealing

Simulated annealing is a popular optimization algorithm used to tackle various combinatorial optimization problems. One important factor that can influence the performance of simulated annealing is the choice of neighborhood structure. In this study, we investigate the effect of different neighborhood structures and window sizes on the performance of simulated annealing for solving a job shop scheduling problem.

Our objective is to explore the three different neighborhood structures and evaluate their impact on the quality of the solutions obtained. The first neighborhood involves swapping two adjacent operations within a window of varying sizes. The second neighborhood extends the first one by swapping three operations within the window. Finally, the third neighborhood is a more complex neighborhood where an optimization problem is solved at each iteration.

By examining these three neighborhood structures, we aim to determine the relationship between window size and solution quality, as well as identify the most effective approach for our job shop scheduling problem. Through extensive experiments and analysis, we gain insights into the performance of simulated annealing and its sensitivity to the choice of neighborhood structure.

3.1.1. First neighborhood

To explore the effect of the window size w on the performance of simulated annealing, we tested twenty different sizes, ranging from one to twenty. The pairs of operations that were interchanged at each iteration of the simulated annealing algorithm were adjacent within a window of a size $2 * w$, i.e. from 2 to 40. For each window size w, we ran the algorithm ten times for a fixed number of iterations (40,000), with an initial temperature of 1000 and a cooling rate of 0.9999. After 40,000 iterations, the final temperature is 18.3119. We recorded the best solution found for each test. Each test was conducted in approximately seven minutes, the size does not affect the time complexity of the algorithm. The mean of the objective value over the ten tests for each different window size is shown in Figure 3.1, with error bars indicating the standard deviation of means over the tests. The reference objective value is 8,286,958, which is the objective value of the initial schedule used for all tests.
We then compared the quality of the solutions obtained using different window sizes and analyzed the effect of window size on the convergence rate.
Our results show that the choice of window size has a significant impact on the quality of the solution obtained by simulated annealing. Specifically, we find that larger neighborhood sizes lead to better objective values. The decrease in objective value is significant for window sizes from one to ten, while it is considerably less significant for sizes from ten to twenty until being constant around an objective value of 3,200,000. The best objective value was achieved with a window size of $w = 19$, which was 3,185,492.
These results suggest that the choice of window size should be carefully considered when using simulated annealing to solve scheduling problems.

For the second instance of the problem, we observe that the objective value tends to decrease as the window size increases, although not consistently. The variability in the results is slightly higher compared to the first instance. It is important to note that the scale of the first and second instances differs, which can lead to confusion when comparing the variability. However, it appears that similar to the first instance, a larger window size tends to result in a lower objective value.

Regarding the third instance, we observe that the objective value significantly decreases with window sizes ranging from 1 to 7 and then remains relatively

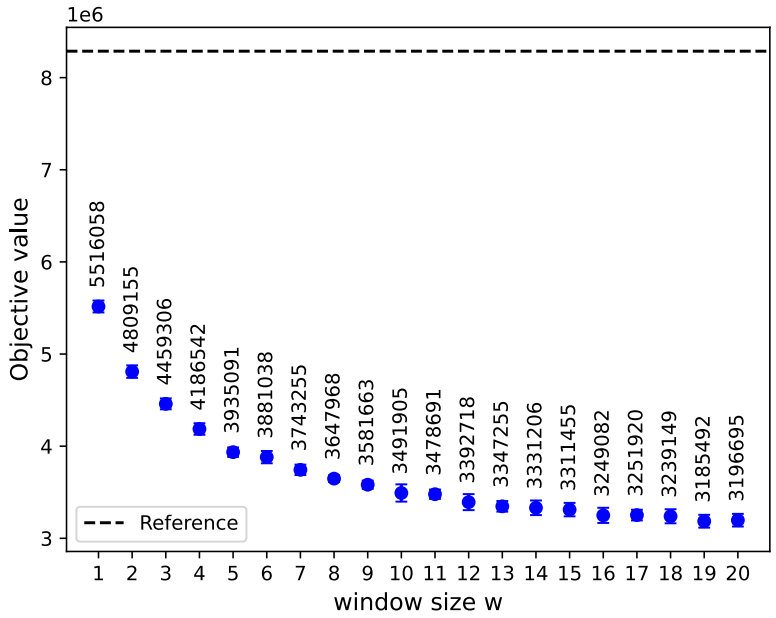


Figure 3.1.: Mean objective value over 10 tests for different window sizes w using the first neighborhood

constant. This suggests that a sufficiently large window size is needed to achieve good results. Therefore, if we choose a large window size such as $w = 20$, we can obtain favorable outcomes, even with the third instance.

These findings highlight the significance of carefully selecting the window size when utilizing simulated annealing for solving scheduling problems. It demonstrates the impact of window size on the quality of solutions obtained and emphasizes the need for experimentation and optimization to determine the most effective window size for specific problem instances.

3.1.2. Second neighborhood

In addition to the experiments conducted with the first neighborhood, we also performed similar tests using the second neighborhood. The second neighborhood is an extension of the first neighborhood, where instead of swapping two operations, we swap three operations within a window of a certain size. However, the results obtained with the second neighborhood, shown in Figure 3.2 were less

conclusive compared to the first neighborhood.

In the experiments with the second neighborhood, we followed a similar setup as before, testing twenty different window sizes ranging from one to twenty. We ran the simulated annealing algorithm ten times for each window size with a fixed number of iterations (40,000), an initial temperature of 1000, and a cooling rate of 0.9999. The final temperature achieved after 40,000 iterations is 18.3119.

Unlike the first neighborhood, the results obtained with the second neighborhood do not show a clear trend when observing the objective values in relation to the size of the window. The standard deviation of the objective values is considerably large, indicating high variability in the obtained solutions.

For the second neighborhood, we observe that a window size of 20 resulted in the lowest objective value, which is around 2,500,000. However, it is important to note that there are also instances where high objective values are obtained, reaching around 6,000,000.

Based on these findings, we can conclude that the second neighborhood is not as effective as initially expected. The objective values obtained using the second neighborhood were highly variable, making it difficult to determine a clear relationship between window size and solution quality. Therefore, it is not possible to draw definitive conclusions regarding the impact of window size on the performance of simulated annealing when using the second neighborhood.

The previous observations hold true for the second instance of the problem as well. However, the results obtained with the third instance present a notable contrast compared to the first two instances. In fact, the outcomes obtained with the third instance align with those obtained using the first neighborhood. Specifically, we observed a significant decrease in the objective value with window sizes ranging from 1 to 6, followed by a relatively constant value.

It is worth noting that the variability in the results is remarkably low, as all ten tests for each window size converge to the same objective value. Without a larger number of instances, we cannot draw conclusions about the second neighborhood on smaller instances like the third.

To conclude, the second neighborhood demonstrated its effectiveness only for the

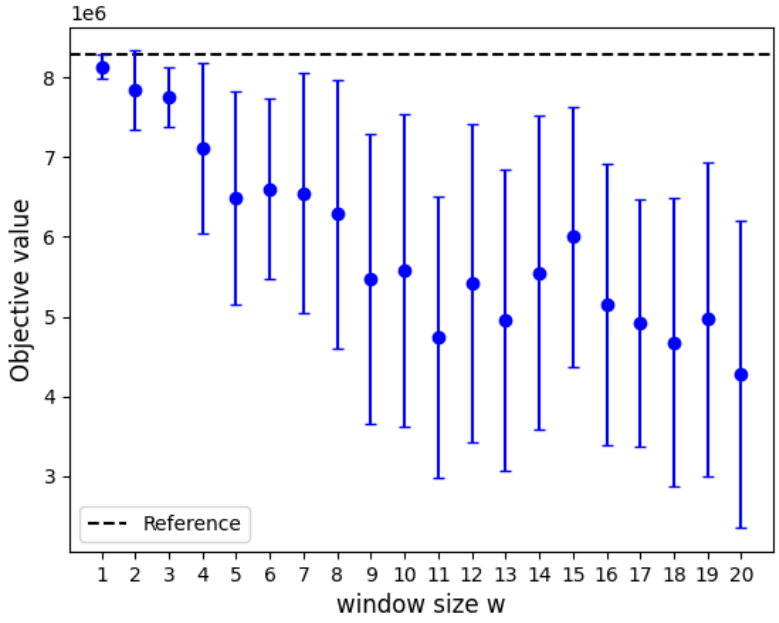


Figure 3.2.: Mean objective value over 10 tests for different window sizes w using the second neighborhood

third instance of the problem and cannot be generalized to solve industrial-scale job shop problems. It is evident that different problem instances may require different approaches, and the choice of neighborhood structure should be tailored accordingly. The results highlight the importance of carefully considering the problem characteristics and exploring various neighborhoods to identify the most suitable approach for optimizing different types of scheduling problems.

3.1.3. Third neighborhood

The experiments for the third neighborhood were conducted with an initial temperature of 100, a cooling rate of 0.9965, and a total of 2627 iterations, resulting in a runtime of approximately 8 minutes With 24 combinations tested at each iteration, a total of 63,048 schedules were evaluated.

However, the results obtained with the third neighborhood are inconclusive. Similar to the second neighborhood, Figure 3.3 does not show a clear trend in the objective values as a function of the window size. Additionally, the objective values display

significant variation and do not reach low values. Most tests yield high objective values, with only a few tests achieving objective values around 5,500,000. These values are obtained with window sizes of 14 and 16. Moreover, the objective reached for the second and the third instances are also lower than that achieved with the first and the second neighborhoods.

Considering the lack of consistent results and the high variability in objective values, it is challenging to draw definitive conclusions regarding the impact of window size on the performance of simulated annealing when using the third neighborhood. However, based on these findings, we can conclude that employing a more complex neighborhood does not appear to be beneficial for solving our three instances of job shop scheduling problem.

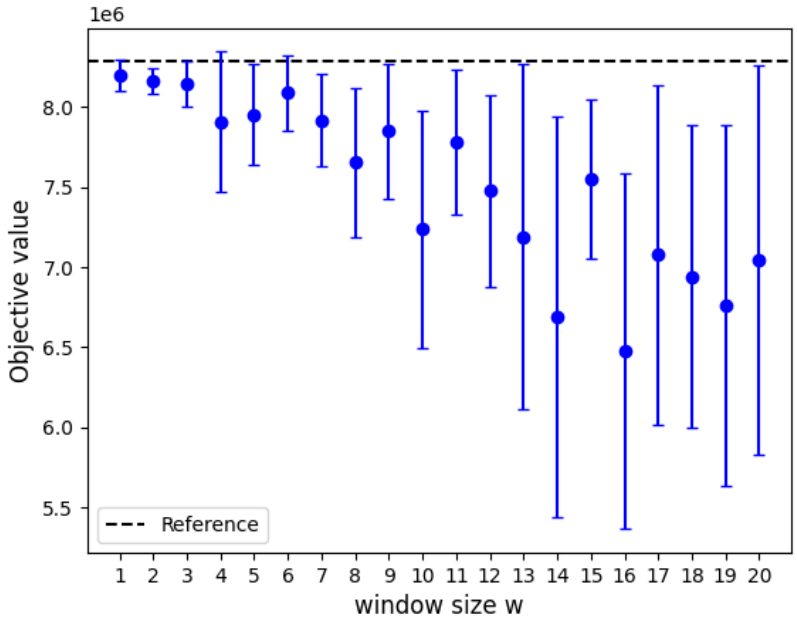


Figure 3.3.: Mean objective value over 10 tests for different window sizes w using the third neighborhood

3.1.4. Conclusion

In this study, we conducted experiments to evaluate the impact of different neighborhood structures and window sizes on the performance of simulated

annealing for a job shop scheduling problem.

For the first neighborhood, larger window sizes lead to better objective values, with the best result achieved at a window size of 19. However, the results for the second and third neighborhoods are inconclusive, showing high variability and no clear relationship between window size and solution quality. In addition, since the results obtained with the third neighborhood are inconclusive, it suggests that exploring more complex neighborhood structures may not be necessary.

These findings emphasize the importance of carefully selecting the neighborhood structure and the window size when using simulated annealing for scheduling problems. While the first neighborhood shows promising results, the second and third neighborhoods do not provide consistent improvements.

In summary, the choice of neighborhood structure and window size significantly influences the performance of simulated annealing for job shop scheduling, highlighting the need for further research to explore alternative approaches.

3.2. Ranking Method

We explored the effect on the performance of the ranking method of the set $S \subset V$ of jobs on which the constraint 2.6 is applied. We tested different sets composed of these subsets of operations for each machine, including:

1. All the pairs of operations
2. All the operations
3. All the operations except one
4. All the pairs of successive operations
5. All the triples of successive operations
6. All the sets of 4 successive operations
7. All the sets of 5 successive operations

Each subset can be included or excluded from S. When the subsets 1 and 4 are both included in S, the same constraint is applied multiple times since pairs of successive

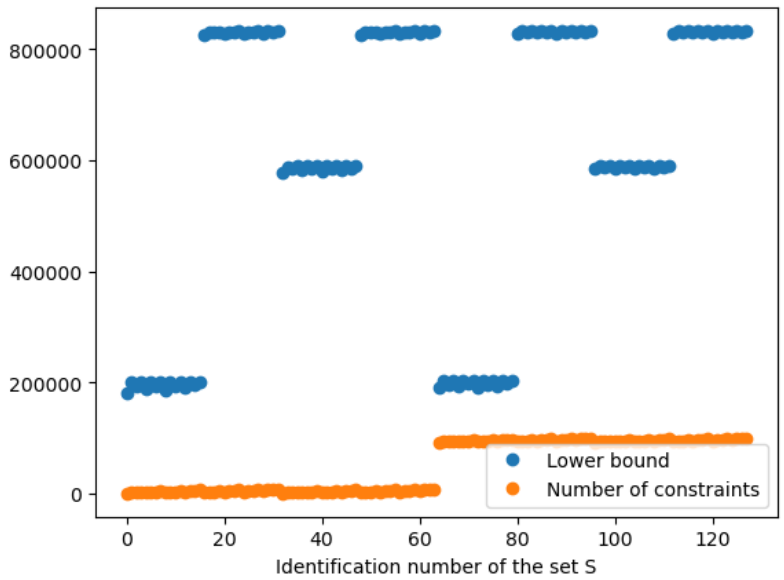


Figure 3.4.: Values of the lower bound and number of constraints of the ranking method over different sets $\mathcal{S}$

operations are also pairs of operations. Each of the subsets enumerated can be part or not of the S. We denote the set S where constraint 2.6 is applied to all successive pairs as $S = [0, 0, 0, 1, 0, 0, 0]$ (x), where x is the set identification number.

The tables A.1, A.2, A.3, and A.4 report the lower bound, objective, time and number of constraints for each set of S to solve the first instance.

3.2.1. About the lower bound

The values of the lower bound and number of constraints of the ranking method over different sets $\mathcal{S}$ is shown in Figure 3.4. We observed three clusters of lower bound values:

- Around 830,000 (high-value cluster): all sets include the subset 3 "All operations except one". The highest lower bound is 833,278 and is reached four times. The sets [1,0,1,0,1,1,1] (88) and [1,0,1,1,1,1,1] (96) both reach this value and are distinguished by the subset 4. The sets [1,1,1,0,1,1,1] (120) and [1,1,1,1,1,1] (128) also reach this value and are distinguished by the subset 4. These two pairs of sets differ from the subset 2.

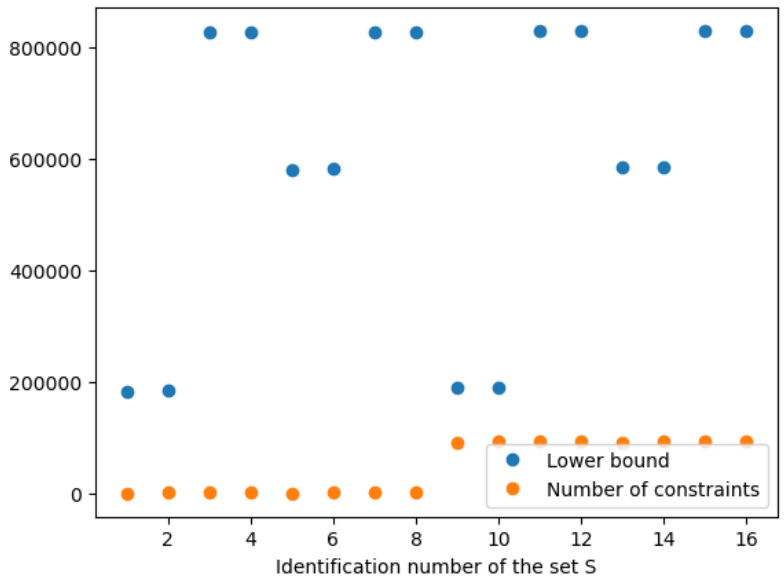


Figure 3.5.: Values of the lower bound and number of constraints of the ranking method over different sets $\mathcal{S}$ including only the subsets 1, 2, 3 and 4

- Around 590,000 (middle-value cluster): all of the sets have in common that they do not include the subset 3 and include the subset 2.
- Around 200,000 (low-value cluster): all of the sets have in common that they do not include the subsets 3 and 2. The lowest lower bound is 181,927 and it is reached by set [0,0,0,0,0,0,0] (1).

Moreover, Figure 3.4 shows that the total number of constraints does not have an impact on the lower bound. The discussion over the three clusters does not include any reference to the last three subsets 5, 6 and 7, which suggests that they do not influence the lower bound. This is confirmed by Figure 3.5 where we can still distinguish the three clusters. Figure 3.5 reports the values of the lower bound and number of constraints of the ranking method over different sets $\mathcal{S}$ including only the subsets 1, 2, 3 and 4. The description of these sets is reported in the table A.5.

In general, we can observe that the last three constraints and the total number of constraints do not have a huge influence on the lower bound.

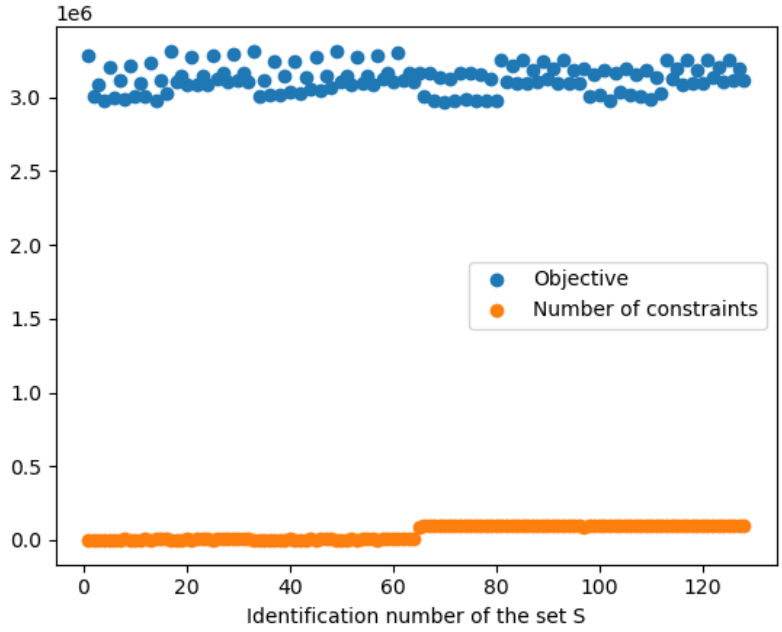


Figure 3.6.: Values of the objective and number of constraints of the ranking method over different sets $\mathcal{S}$

3.2.2. About the objective

The values of the objective and the number of constraints of the ranking method over different sets $\mathcal{S}$ is shown in Figure 3.6. We observed that whatever the set S and the number of constraints, the objective value is around 3,000,000. The minimum objective value is 2,969,619 and is reached by the set [1,0,0,0,1,0,1] (70). The lower bound of this set is 202,442 which is part of the low-value cluster.
The minimum objective value with a lower bound in the high-value cluster is 3,083,040 and is reached by the set [0,0,1,0,1,1,1] (24). The lower bound value is 831,859.

3.2.3. Conclusion

Regardless of the set applied, the ranking method consistently yields an objective value of approximately 3 million. The ranking method achieves this objective in under a second, while simulated annealing takes seven minutes and 40,000 iterations to reach 3,100,000. Therefore, if we have access to an efficient solver of

linear programs, the ranking method provides a superior solution in significantly less time than simulated annealing. Furthermore, the influence of the sets is primarily concentrated on the lower bound, whereby three distinct clusters can be identified. Additionally, all the observations and conclusions derived from the first problem instance remain valid for the second and third instances.

3.3. Branch and Bound Algorithm

3.3.1. Selection of the parameters to improve the pruning

Efficient pruning is essential for the success of the branch and bound algorithm. Pruning occurs when the lower bound of a node exceeds the current best objective. In this subsection, we explore three parameters that could influence the pruning process and improve its efficiency: the choice of the set $\mathcal{S}$, the constraint strategy and the expansion strategy.

Choice of the set $\mathcal{S}$

The selection of the set $\mathcal{S}$ could significantly impact the pruning effectiveness. To enhance pruning, we need to choose a set $\mathcal{S}$ that provides a high initial lower bound without slowing down the tree search. One promising alternative is the set $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$ (17). This set is composed of the unique subset "All the operations except one" subset and exhibits a lower bound in the high-value cluster. Moreover, using this set in the ranking method has shown promising results in terms of solving the problem efficiently. Notably, $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$ (17) requires the minimum number of constraints while still achieving a high-value cluster lower bound for the three instances of the problem.

Choice of the constraint strategy

The constraint strategy (see Section 2.5.2) is another parameter that can significantly influence pruning efficiency in the branch and bound approach

through the lower bound. The faster the lower bound increases when we add constraints to the model, the faster we can expect to prune the search.

We present a detailed evaluation of the four different constraint strategies (CM-FO, CM-LV, LJ and CM-5LV) in terms of their impact on the lower bound evolution during dive iterations. Two different sets of constraints $\mathcal{S}$ ([0,0,1,0,0,0,0] and [0,0,0,0,0,0,0]) are considered to examine the strategy's performance under different scenarios.

Figures 3.7 and 3.8 report the evolution of the lower bound of ten dives, employing the four different constraint strategies (CM-FO, CM-LV,LJ and CM-2LV) and two sets $\mathcal{S}$ ([0,0,1,0,0,0,0] and [0,0,0,0,0,0,0]).
The dives are executed independently, ensuring that each dive ends when its lower bound exceeds the best objective discovered during that specific dive. The intention is to obtain a comprehensive overview of the performances of the dives without allowing a single dive's exceptionally low objective to influence the others. In order to achieve this, we have adjusted the dive technique to enable pruning when the lower bound equals the best objective of the current dive, rather than maintaining a shared objective across all dives. This approach ensures that each dive's progress remains independent and unaffected by the outcomes of other dives.

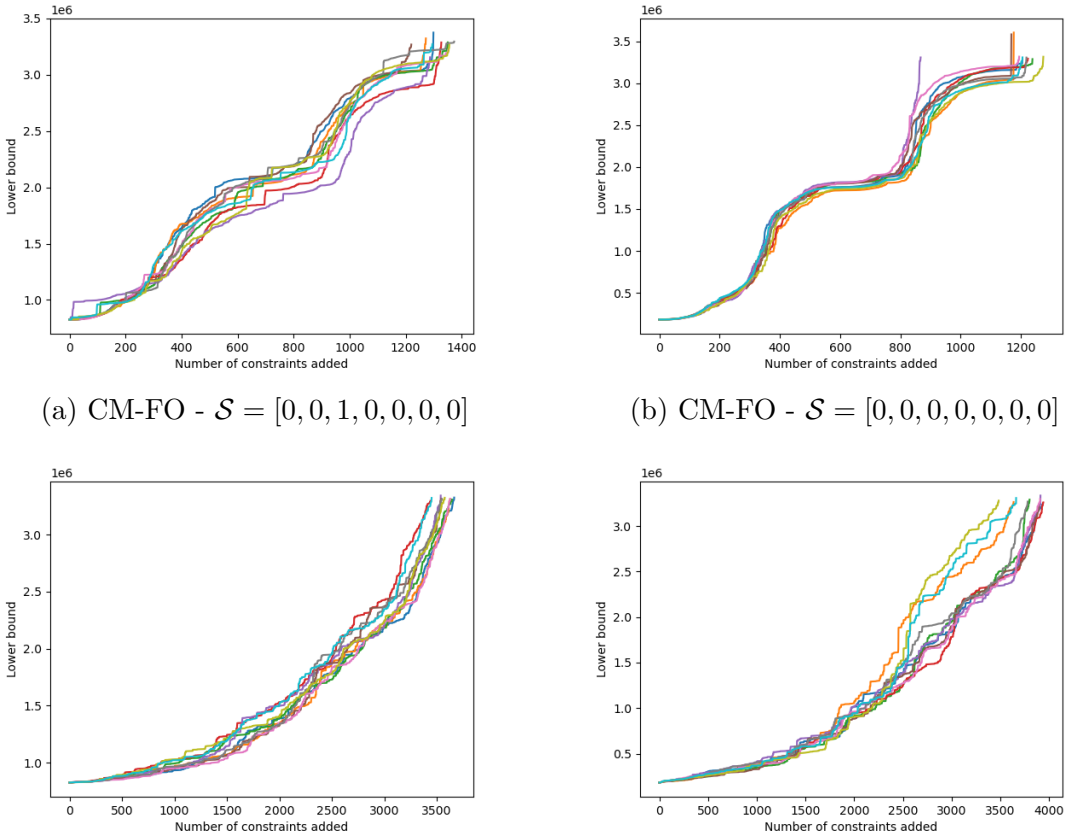
Firstly, it is important to note that pruning does not occur at the same objective value for different constraint strategies. Some strategies are more susceptible to reaching a good objective value faster during the exploration process.

Among the constraint strategies considered, the CM-FO strategy exhibits the highest increase in lower bound. After approximately 1300 constraints are added to the model, the lower bound reaches the upper bound at around 3,500,000.

The CM-LV and CM-5LV strategies have the second highest lower bound increase, and their lower bounds behave similarly. For the set [0,0,1,0,0,0,0], the lower bound reaches the upper bound between 2,500,000 and 2,750,000 after adding 2500 constraints. Similarly, for the set [0,0,0,0,0,0,0], the lower bound reaches the upper bound after adding around 2800 constraints. Although these strategies have a slower increase rate compared to CM-FO, they still demonstrate significantly higher growth than the LJ strategy.

In contrast, the lower bound of the LJ strategy exhibits only a slight increase initially, with the lower bound remaining under 1 million even after adding 1,000 constraints. Pruning occurs after adding more than 3,500 constraints, with the lower bound reaching slightly below 3,500,000. This suggests that the scheduling is significantly influenced by the critical machine rather than the size of jobs.

In summary, the constraint strategies CM-FO, CM-LV, and CM-5LV are the most promising strategies in terms of their lower bound growth and then demonstrate potential for improving the pruning process. Additionally, all the observations and conclusions derived from the first problem instance remain valid for the second and third instances.



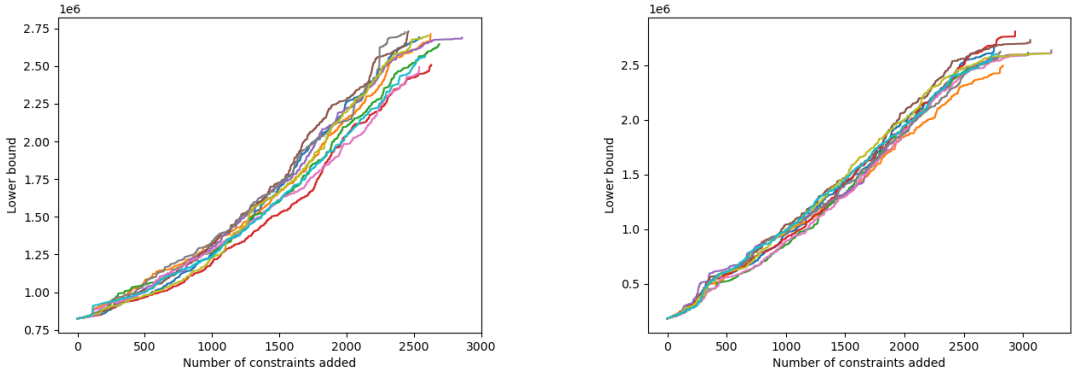
(a) CM-FO - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

(b) CM-FO - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

(c) LJ - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

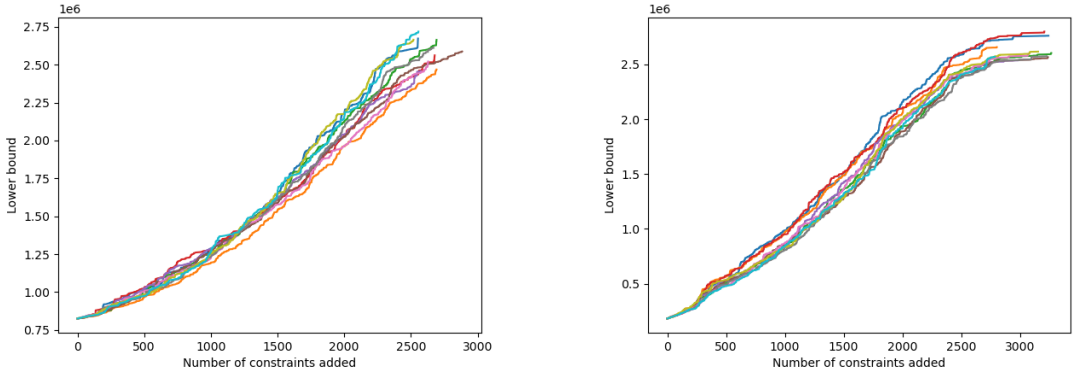
(d) LJ - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

Figure 3.7.: Evolution of the lower bound for 10 dives with the constraint strategies CM-FO and LJ and two sets $\mathcal{S}$



(a) CM-LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

(b) CM-LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$



(c) CM-5LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

(d) CM-5LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

Figure 3.8.: Evolution of the lower bound for 10 dives with the constraint strategies CM-LV and CM-5LV and two sets $\mathcal{S}$

Choice of the expansion strategies

In addition to the set $\mathcal{S}$ and the constraint strategy, the branch and bound method can be influenced by a third parameter: the expansion strategy.

We tested a first expansion strategy, the Breadth-First Search (BFS) strategy. With BFS, we prioritize visiting all nodes at the same depth before moving on to the next depth. This expansion strategy has the potential to increase the lower bound of the problem. Increasing this value allows us to decrease the optimality gap, and thus to determine how close the solution is to the optimal solution.

To gain a comprehensive understanding of the branch and bound method, we ran it for a duration of 16 hours using two different constraint strategies: Critical Machine

First Operation (CM-FO) and Critical Machine Largest Violation CM-LV. In both experiments, we used a schedule that was obtained from the dive approach, as discussed in Section 3.4. This schedule achieved an objective value of 2,422,458, which is the most optimal result attained during a two-hour dive approach run. We started with a new model. The constraints associated with the best schedule found during the dive approach are not added initially to the model. The initial lower bound is then only 825,303.

Using CM-FO strategy with the BFS expansion strategy, we visited a total of 123,487 nodes within the 16-hour timeframe. Surprisingly, the global lower bound was 825,448. On the other hand, with the CM-LV strategy and the BFS expansion strategy, we visited 111,360 nodes. The global lower bound was 826,459. In both strategies, no pruning was performed, and we did not encounter a node where further constraints could not be added.

Despite the extensive exploration of nodes and the considerable number of visited nodes, none of the approaches was able to achieve an objective value better than the initial one even after 16 hours. Additionally, the lower bound did not increase significantly as expected. Its value remained close to the initial lower bound. This observation can be attributed to the fact that even if 123,487 nodes were visited, we only reached a depth of 17. This indicates that a maximum of 17 constraints were added to the model. Figures 3.7 and 3.8 provide a visual representation of this observation. It highlights the need for a larger number of constraints before a significant increase in the lower bound becomes apparent. The BFS expansion strategy was also inconclusive on the second and third instances of the problem, even though the third is smaller.

When evaluating the breadth-first search (BFS) expansion strategy, it became clear that this approach does not effectively increase the global lower bound. This leads us to a simple conclusion: the best node search strategy, which involves expanding the node with the lowest lower bound, is unlikely to provide more conclusive results about the lower bound.

Expanding nodes based on the lowest lower bound does not address the underlying issue of limited lower bound improvement. Instead, it would lead to a similar exploration pattern as the BFS strategy, potentially resulting in a comparable

optimization gap. Therefore, it is evident that the choice of the expansion strategy alone cannot be relied upon to achieve a significant global increase in the lower bound within the branch and bound method.

Conclusion

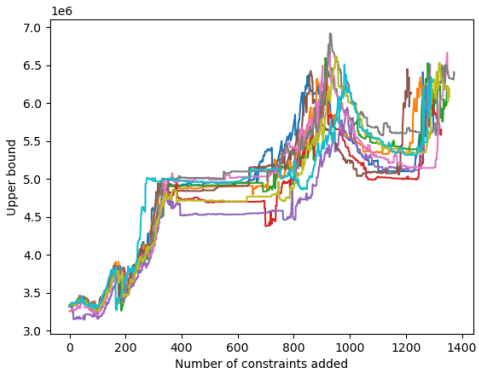
Even when applying parameters that favor pruning, it is evident that pruning alone is not effective in achieving significant improvements in the branch and bound method. Without pruning, the branch and bound approach essentially reverts to a simple method of complete exploration. Given the large size of the three instances of the problem at hand and the use of the breadth-first search expansion strategy, it becomes apparent that a schedule better than the initial one is not attainable.

3.3.2. Results of the branch and bound method

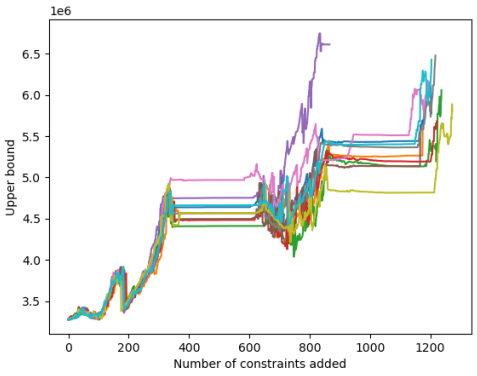
Taking into account the limitations of pruning and the challenges posed by the problem's size, it becomes evident that alternative strategies are needed to improve the branch and bound method's performance. In light of this observation, a more promising approach is to adopt the "greedy search" expansion strategy, with the hope of achieving better schedules. Additionally, it is worth considering the constraint strategy that facilitates the most rapid reduction in the upper bound, independently of the lower bound.

With these considerations in mind, let us now consider the evolution of the upper bound across multiple dives, using various constraint strategies. Figures 3.9 and 3.10 report the evolution of the upper bound across ten dives, employing the four different constraint strategies (CM-FO, CM-LV,LJ and CM-2LV) on two sets $\mathcal{S}$ ([0,0,1,0,0,0,0] and [0,0,0,0,0,0,0]).

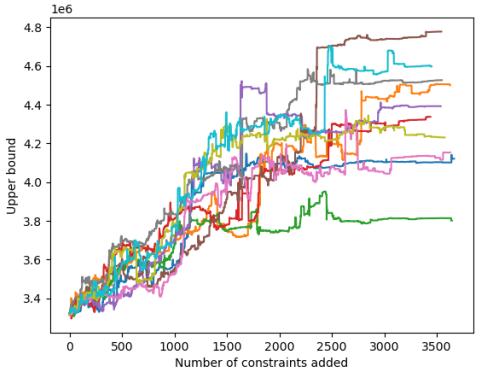
It is notable that all ten dives using the CM-FO and LJ constraint strategies display an increasing upper bound. In contrast, the two constraint strategies that show promise are CM-LV and CM-5LV. Their upper bounds show a similar pattern, initially experiencing a slight decrease, followed by subsequent increases. However, after the addition of approximately 600 constraints, a significant



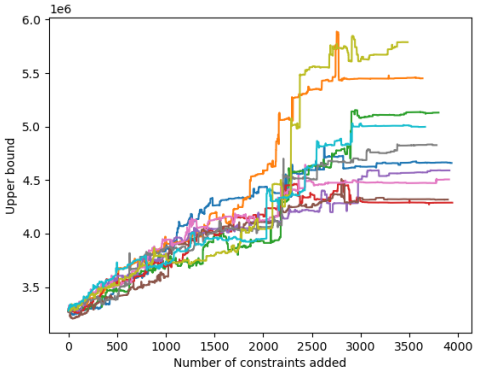
(a) CM-FO - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(b) CM-FO - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

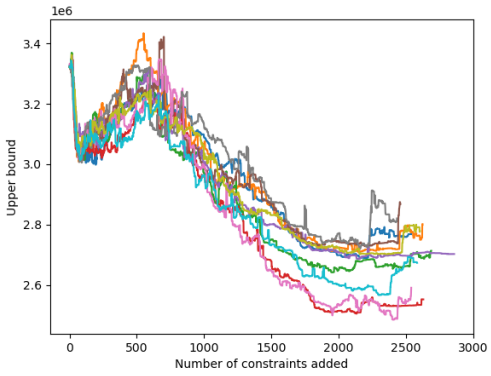


(c) LJ - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

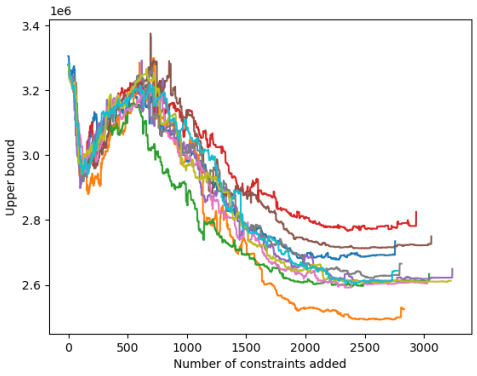


(d) LJ - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

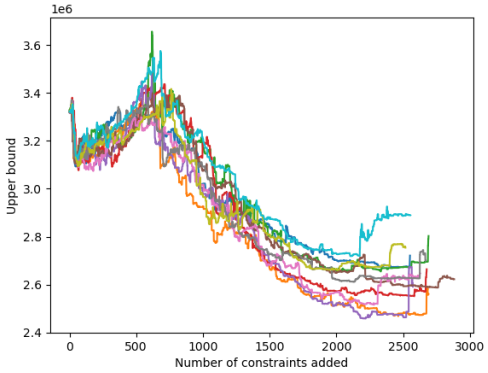
Figure 3.9.: Evolution of the upper bound for 10 dives with the constraint strategies CM-FO and LJ and two sets $\mathcal{S}$



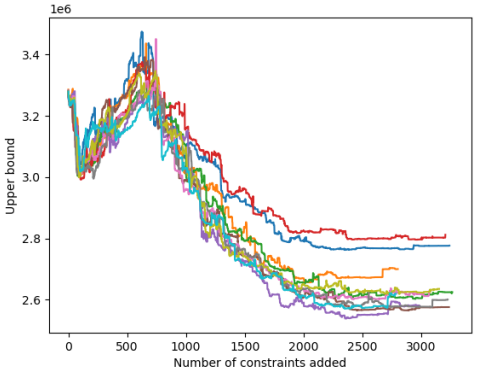
(a) CM-LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(b) CM-LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$



(c) CM-5LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(d) CM-5LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

Figure 3.10.: Evolution of the upper bound for 10 dives with the constraint strategies CM-LV and CM-5LV and two sets $\mathcal{S}$

reduction is observed, occasionally reaching values as low as 2,500,000.

In our experiments, we further investigated the branch and bound algorithm with the greedy search expansion strategy, using the CM-LV and CM-5LV constraint strategies for a duration of two hours. The CM-LV strategy resulted in visiting 17,167 nodes, while the CM-5LV strategy visited 14,954 nodes. However, despite the extended computational time and exploring various constraint strategies, we did not find a better schedule than the initial one obtained from the dive approach. In addition, the same observations about the evolution of the upper bound and the efficiency of the greedy search strategy can be made for the second and third instances of the problem.

3.3.3. Conclusion

In this study, we explored the application of the branch and bound algorithm to our job shop scheduling problem, aiming to find an optimal schedule. We investigated various aspects of the branch and bound method, including the choice of the set $\mathcal{S}$, the constraint strategies, and the expansion strategies, with the goal of improving the pruning process and overall performance.

Regarding the choice of the set $\mathcal{S}$, we identified that selecting the set $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$ (17) provided a high initial lower bound while minimizing the number of constraints required initially in the initial model.

We evaluated four constraint strategies (CM-FO, CM-LV, LJ, and CM-5LV) and their impact on the pruning process. The CM-FO strategy demonstrated the fastest pruning, requiring approximately 1300 constraints to initiate pruning. The CM-LV and CM-5LV strategies exhibited comparable performance, achieving lower bounds between 2,500,000 and 2,750,000 after adding around 2500-2800 constraints. These constraint strategies proved promising in increasing the lower bound during the branch and bound algorithm, unlike the LJ strategy.

We also investigated the impact of the expansion strategies on the branch and bound method. The Breadth-First Search (BFS) expansion strategy did not result in a significant global increase in the lower bound, even after exploring a large number of nodes. This observation suggests that the best node strategy, which selects nodes

based on the lowest lower bound, is unlikely to yield different results. Therefore, the expansion strategy alone does not provide a substantial improvement in the lower bound within the branch and bound method.

Considering the limitations of pruning and the challenges posed by the problem's size, alternative strategies are needed to enhance the branch and bound method's performance. Although the use of the "greedy search" expansion strategy shows promise for obtaining better schedules, this strategy was not conclusive.

In conclusion, the branch and bound method with specific parameter choices for the pruning cannot help achieve good lower bounds. Moreover, choosing specific parameters that increase the upper bound is also not conclusive. We did not find a better schedule than the one obtained through alternative approaches such as the dive approach. Therefore, further investigations and alternative techniques are required to tackle the scheduling problem effectively and find optimal or near-optimal schedules. Additionally, all the observations and conclusions derived from the initial problem instance remain valid for the second and third instances.

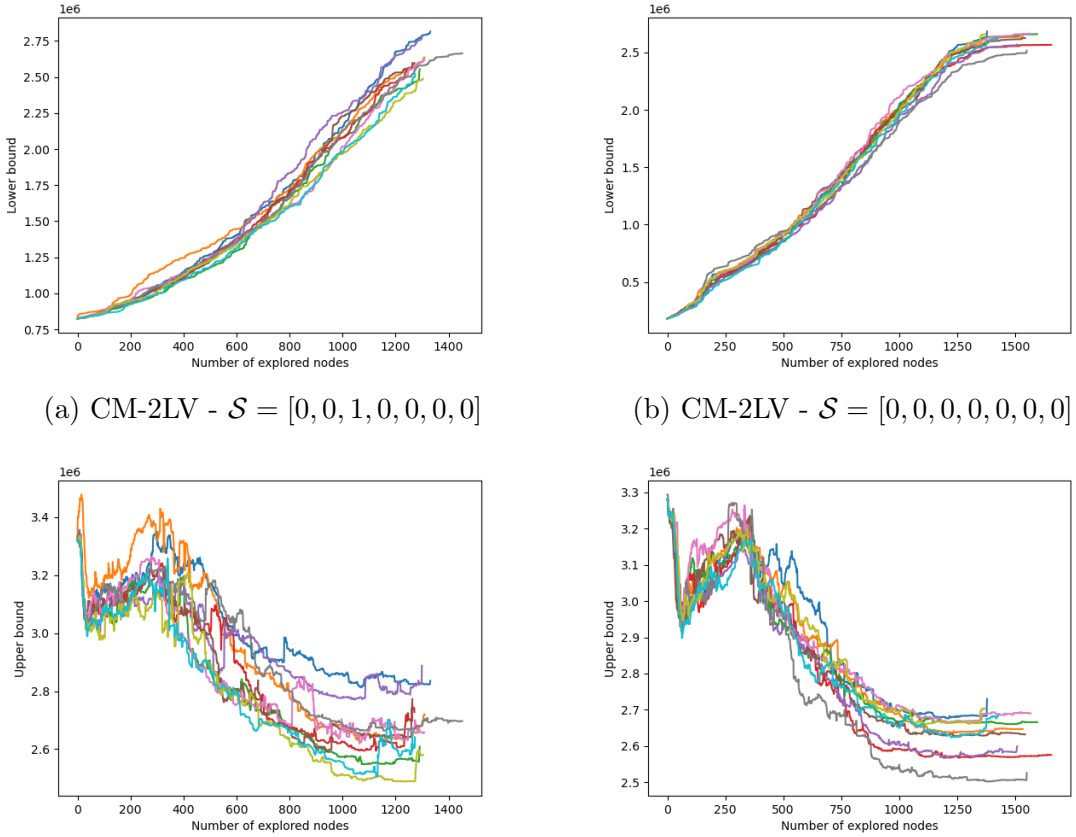
3.4. Dive

3.4.1. Selection of the constraint strategies

The application of the dive approach presents a valuable advantage to explore a significant number of nodes, surpassing the efficiency of the branch and bound method in terms of node exploration as seen in Section 2.6.2. By executing multiple dives within a relatively short time span, we increase the probability of discovering an initial low objective before initiating the branch and bound method.

In our experimental analysis, we conducted a series of dives over a two-hour duration, focusing on specific constraint strategies that exhibit a decrease in the upper bound. The evolution of the upper bound of the different constraint strategies ver ten dives is illustrated in Figures 3.9 and 3.10.

The constraint strategies of interest in this analysis are CM-LV and CM-5LV, which demonstrate potential for reducing the upper bound.



(a) CM-2LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

(b) CM-2LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

(c) CM-2LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

(d) CM-2LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

Figure 3.11.: Evolution of the lower bound and the upper bound for 10 dives with the constraint strategy CM-2LV and two sets $\mathcal{S}$

Additionally, we explored the constraint strategy specific to the dive approach, namely CM-2LV. Figure 3.11 reports the evolution of the lower bound and the upper over ten dives and two sets $\mathcal{S}$, [0,0,1,0,0,0,0] and [0,0,1,0,0,0,0], for that constraint strategy. We observe the same behavior as the CM-LV constraint strategy. However, the advantage of the CM-2LV strategy lies in its faster execution time, as indicated in Table 3.1. This implies that we have the potential to conduct a greater number of dives within a two-hour timeframe.

Despite observing an increase in the upper bound across the ten dives using the CM-FO strategy (as shown in Figure 3.9), it is worth noting that this strategy exhibits the lowest execution time among all the tested approaches (see Table 3.1). Therefore, when aiming to conduct a higher number of dives within the two-hour timeframe,

Constraint strategy	Time in minutes	
	$\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$	$\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$
CM-FO	6:30	2:34
LJ	17:18	12:48
CM-LV	10:12	7:12
CM-5LV	12:57	10:22
CM-2LV	6:36	4:59

Table 3.1.: Time to perform 10 dives with different constraint strategies with 2 different sets $\mathcal{S}$

the CM-FO strategy can prove to be a valuable choice. While the upper bound may not be ideal, the advantage of reduced execution time allows for a greater exploration of the solution space, potentially leading to the discovery of more optimal schedules. Consequently, considering the trade-off between execution time and upper bound performance, the CM-FO strategy presents an interesting option for maximizing the number of dives performed within the allotted time frame.

In conclusion, the constraint strategies that are interesting to achieve a low objective value during the dive approach are CM-FO, CM-LV, CM-5LV, and CM-2LV. And this, for the three instances of the problem.

3.4.2. Results

In our experimental analysis, we conducted a series of dives over a two-hour duration with the set $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$. The outcomes of these dives are presented in Table 3.2.

Constraint strategy	Upper bound	Lower bound	Number of constraints added	Number of dives
CM-LV	2,430,527	1,966,009	2,014	130
CM-5LV	2,400,752	2,196,444	2,343	106
CM-2LV	2,425,140	2,047,498	2137	210
CM-FO	3,060,762	988,225	59	208

Table 3.2.: Comparison of Upper Bounds, Lower Bounds, Constraints Added, and Number of Dives for Different Constraint Strategies within a 2-Hour Dive Approach Run

We observe that using the CM-FO strategy allowed us to conduct a remarkable total of 208 dives within the allocated time frame. However, it is important to highlight that despite the extensive exploration, this strategy yielded the largest upper bound out of the four constraint strategies tested. While the CM-FO strategy enabled a significant number of dives, it is evident that its performance in terms of achieving the best objective was not as favorable as desired. The CM-FO strategy may have explored a wide range of solutions but struggled to find an optimal or near-optimal schedule. We have to note that the same conclusion can be done for the second instance.
However, the application of the CM-FO strategy to the third instance yields a comparable upper bound of around 123,000, similar to the other constraint strategies (see Table A.19). Examining the distribution of the best objective values obtained from 100 dives see A.24d), we observe that none of the dives achieved a value below 126,000. Hence, obtaining a CM-FO result reaching 123,000 is more exceptional.

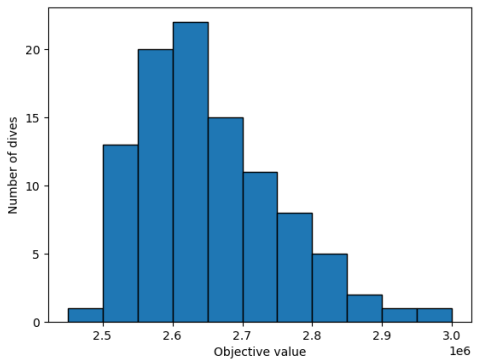
In contrast, the CM-5LV strategy achieved the lowest upper bound among the tested strategies. Although it involved a relatively low number of dives (106 dives), it demonstrated better performance in terms of objective attainment.

The CM-2LV strategy falls in between the CM-FO and CM-5LV strategies, offering a trade-off between upper bounds and the number of dives performed. It achieved a moderate upper bound and involved a substantial number of dives (210 dives). This indicates a balanced exploration strategy that managed to strike a compromise between thoroughly exploring the search space and achieving a relatively favorable objective.

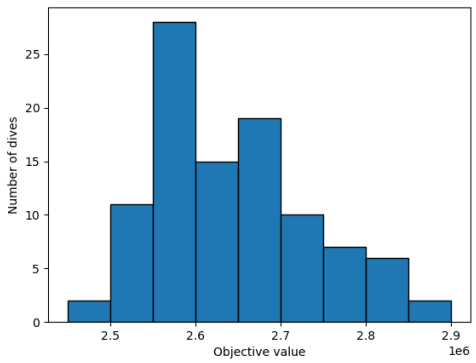
In our analysis, we have observed that the constraint strategies CM-LV, CM-5LV, and CM-2LV, which prioritize the largest violation in selecting constraints, have consistently achieved relatively low objective values of around 2,400,000. This raises the question of whether it is necessary to wait for the full two-hour duration to obtain these favorable objective values.

To investigate this further, we have constructed histograms depicting the distribution of the best objective values obtained from 100 independent dives for each of the constraint strategies (CM-LV, CM-5LV, and CM-2LV). The histograms, shown in Figure 3.12, provide insights into the variability and range of

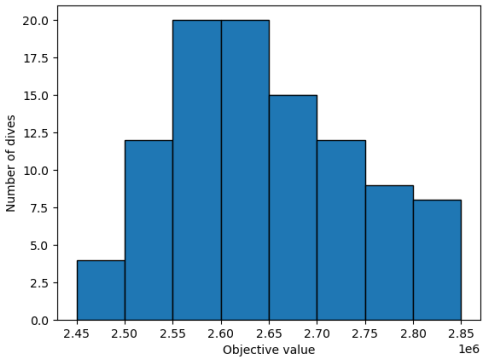
objective values attained within this set of dives. The time to perform the 100 dives is reported in Table 3.3.



(a) CM-LV



(b) CM-5LV



(c) CM-2LV

Figure 3.12.: Distribution of best objective values from 100 dives for constraint strategies CM-LV, CM-5LV, and CM-2LV

We observe that for all three constraint strategies, only a small fraction of dives (less than 5) out of the 100 dives resulted in an objective value lower than 2,500,000. This suggests that a significant number of dives is necessary to ensure the attainment of a low objective value. Among the constraint strategies tested, the CM-2LV strategy emerges as the most promising choice. In 100 dives, it achieves objective values below 2,500,000 more frequently than the other strategies. Additionally, the CM-2LV strategy demonstrates faster execution times, allowing for a greater number of dives within the allotted timeframe.

Importantly, the CM-2LV strategy also demonstrates its efficiency when applied to the other two instances. It consistently delivers lower objective values more

Constraint strategy	Time (in hours)
CM-LV	1:44:47
CM-5LV	2:02:47
CM-2LV	1:02:39

Table 3.3.: Time to perform 100 dives with constraint strategies CM-LV, CM-5LV, and CM-2LV

frequently than the alternative strategies over the 100 dives.

In summary, our analysis suggests that a significant number of dives is necessary to attain a low objective value in the scheduling problem. The CM-2LV constraint strategy stands out as the preferred option, as it not only achieves desirable objectives more frequently within 100 dives but also completes the dives in less time compared to the other strategies.

3.4.3. Conclusion

In conclusion, the dive approach offers a valuable advantage to explore a significant number of nodes compared to the branch and bound method. By conducting multiple dives within a relatively short time span, the chances of discovering an initial low objective value before initiating the branch and bound method are increased.

In our experimental analysis, we focused on specific constraint strategies that exhibited a decrease in the upper bound over ten dives. The constraint strategies of interest were CM-LV, CM-5LV, and CM-2LV.

The CM-2LV strategy, specific to the dive approach, showed promising results. It achieved objective values lower than 2,500,000 more frequently than the other strategies within 100 dives. Additionally, the CM-2LV strategy demonstrated faster execution times, allowing for a greater number of dives within the allocated two-hour timeframe.

While the CM-5LV strategy achieved the lowest upper bound among the tested strategies, it involved a relatively lower number of dives. On the other hand, the CM-FO strategy allowed for a remarkable total of 208 dives within the two-hour

duration, but yielded the largest upper bound.

Overall, our analysis indicates that a significant number of dives is necessary to ensure the attainment of a low objective value in the scheduling problem. The CM-2LV constraint strategy stands out as the preferred option, as it not only achieves desirable objectives more frequently within 100 dives but also completes the dives in less time compared to the other strategies. It strikes a balance between objective attainment and dive efficiency, making it an effective choice in the dive approach.

3.5. Mix of Methods

The simulated annealing algorithm is an optimization technique that relies on an initial schedule to guide its search process. The performance of the algorithm heavily depends on the initial schedule. In the previous section 3.1.1, we investigated the application of simulated annealing using a specific initial schedule where jobs are programmed sequentially, one after the other. This initial schedule, starting with a high initial objective, provides ample room for improvement.

To explore alternative initial schedules, we consider two effective methods: the ranking method and the dive method. Both of these methods offer initial schedules that exhibit favorable objective values compared to the sequential approach. The simulated annealing algorithm could be used to upgrade the initial schedules obtained from the dive and ranking methods. Using the power of simulated annealing, we hope to refine these initial schedules and achieve further improvements in the optimization process.

3.5.1. Simulated annealing after the ranking method

In our first experiment, we used the ranking method to generate an initial schedule. This schedule was obtained by applying the ranking approach with the set $\mathcal{S} = [1, 0, 0, 0, 1, 0, 1]$ (70), which yielded the lowest objective value among all the tested sets. The initial value of the objective is 2,984,830.

To optimize the initial schedule, we ran the simulated annealing algorithm with a

fixed number of iterations (40,000), an initial temperature of 1000, and a cooling rate of 0.9999. After 40,000 iterations, the algorithm reached a final temperature of 18.3119.

The test duration lasted approximately 7 minutes, during which we observed a significant improvement in the objective value. The resulting objective value obtained was 2,092,161, indicating a reduction from the initial objective value. This outcome highlights the effectiveness of the simulated annealing algorithm in optimizing the initial schedule provided by the ranking method.

The summary of the first experimentation on the three instances is reported in Table 3.4. For the three instances, the simulated annealing algorithm is effective to decrease the initial objective value obtained from the ranking method.

Instance	Set $\mathcal{S}$	Initial objective	Final objective	Percentage of reduction
1	[1,0,0,0,1,0,1] (70)	2,984,830	2,092,161	29.9%
2	[1,1,0,0,1,1,1] (104)	2,925,468	2,240,060	23.4%
3	[0,0,0,0,0,1,1] (4)	131,632	121,194	7.9 %

Table 3.4.: Objective values of initial and final schedules obtained with the ranking method and the simulated annealing algorithm

3.5.2. Simulated annealing after the dive method

In our second experiment, we used the dive method to generate an initial schedule. We initialized the dive method with the schedule obtained from the ranking method with the set $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$. The dive method was then applied using the CM5LV constraint strategy and the set $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$. The value of the objective of the schedule obtained is 2,400,752.

Similar to the previous experiment, we employed the simulated annealing algorithm with a fixed number of iterations (40,000), an initial temperature of 1000, and a cooling rate of 0.9999. The chosen neighborhood for exploration was the first one, with a window size of 20.

During the approximately 7-minute test duration, we observed a significant improvement in the objective value. The resulting objective value obtained was

2,009,964, which demonstrated a decrease compared to the initial objective value. This outcome emphasizes the effectiveness of the simulated annealing algorithm in optimizing the initial schedule provided by the dive method.

The summary of the second experimentation on the three instances is reported in Table 3.5. For the three instances, the simulated annealing algorithm is effective in reducing the initial objective value obtained from the dive approach, although the percentage of reduction for the third instance is very small.

Instance	Constraint strategy	Initial objective	Final objective	Percentage of reduction
1	CM-5LV	2,400,752	2,009,964	16.3%
2	CM-2LV	2,540,168	2,163,092	14.8%
3	CM-LV	122,988	119,685	2.7%

Table 3.5.: Objective values of initial and final schedules obtained with the dive method and the simulated annealing algorithm

The second experiment was repeated using the dive approach to generate an initial schedule. But this time, the initial schedule selected has an objective value that is not the lowest obtained during two hours of dives. The summary of the third experiment for the three instances is presented in Table 3.6.

The simulated annealing algorithm effectively improves the initial schedules generated by the dive method for all instances in this third experiment. However, starting from a schedule with a higher objective value does not allow us to achieve objective values as low as those obtained in the second experiment. For the first two instances, although the objective values obtained in the second experiment were slightly higher compared to the initial objectives of the first experiment, they were still relatively close. For the third instance, the objective value obtained in the third experiment was even higher than the initial objective value of the second experiment.

3.5.3. Conclusion

In this section, we explored the combination of the simulated annealing algorithm with alternative initial scheduling methods, namely the ranking method and the

Instance	Constraint strategy	Initial objective	Final objective	Percentage of reduction
1	CM-5LV	2,740,849	2,024,809	26.1 %
2	CM-2LV	2,680,832	2,177,864	18.7 %
3	CM-LV	138,058	123,012	10.9 %

Table 3.6.: Objective values of initial and final schedules obtained with the dive method and the simulated annealing algorithm

dive method.

By using the ranking method, we obtained initial schedules with favorable objective values compared to the sequential approach. The simulated annealing algorithm was then used to further optimize these initial schedules further. The results show that the simulated annealing algorithm effectively improved the initial schedules obtained from the ranking method, resulting in significant reductions in the objective values for all tested instances.

Similarly, we applied the dive method to generate initial schedules and took the best ones. The simulated annealing algorithm was used to refine these initial schedules, resulting in reductions in objective values. However, when the simulated annealing algorithm is applied starting from a schedule with a higher objective value, the objective values obtained were not as low as those obtained in the previous experiment.

In the end, the combination of the simulated annealing algorithm with the best schedule of the dive method is the most promising strategy to achieve the lowest objectives for our three instances.

Overall, the combination of simulated annealing with alternative initial scheduling methods proved to be effective in improving the initial schedules and achieving better optimization results than using either method alone. These results demonstrate the potential of simulated annealing as a valuable tool for refining and improving schedules with good initial objectives obtained from different methods, leading to improved optimization results.

4. Conclusion

4.1. Brief Summary and Conclusion

This thesis addresses the challenges posed by large-scale job shop scheduling problems. The main objective is to explore the performance and limitations of established optimization techniques, namely simulated annealing, branch and bound, and dives, when applied to industrial-size problems.

The research makes use of the ranking method, an approach that uses linear programming relaxation to generate initial schedules with low objective values. The ranking method consistently outperforms the simulated annealing algorithm in terms of speed and objective value, achieving remarkable results within seconds.

Additionally, the dive approach is proposed as an alternative to the branch and bound method. By maintaining the same model within each dive and introducing random order constraints, the dive approach enables deeper exploration of the search space. It outperforms the branch and bound method and produces schedules with low objective in less than 100 dives.

Comparative analyses between the simulated annealing algorithm, ranking method, and dive approach highlight the benefits of combining these methods. The simulated annealing algorithm enhances the initial schedules obtained from the ranking method and dive approach, leading to further optimization and improved objective values.

In conclusion, this thesis demonstrates the limited effectiveness of the branch and bound method for large-scale job shop problems. The ranking method, dive approach, and simulated annealing algorithm provide valuable tools for generating high-quality initial schedules and refining them to achieve optimal results.

4.2. Future Work

Following this thesis, a promising avenue for future research lies in improving the computation of the lower bound within the context of the branch and bound algorithm. The lower bound serves as a critical component in pruning search branches and optimizing the algorithm's efficiency. Enhancing the accuracy and efficiency of lower bound computations can lead to more effective pruning strategies and, consequently, faster convergence toward optimal or near-optimal schedules.

Moreover, alternative perspectives can be adopted when approaching the scheduling problem. Future research can explore the application of constraint programming as an alternative approach to scheduling optimization in TOOWHE. Constraint programming offers a declarative modeling framework that allows for the representation of complex constraints and their relationships. By formulating the scheduling problem using constraints and using specialized constraint solvers, it becomes possible to systematically explore the solution space and find high-quality schedules. Testing constraint programming in the context of TOOWHE scheduling can provide insights into its applicability, scalability, and performance compared to other optimization algorithms. This involves formulating the scheduling problem using constraints that capture the specific requirements and constraints of TOOWHE, such as job dependencies, resource limitations, and client orders.

Given the increasing prominence of artificial intelligence, another perspective is to explore reinforcement learning. Reinforcement learning algorithms excel in learning optimal decision-making policies through interactions with an environment. In the context of TOOWHE, where schedules could be updated dynamically as new jobs arrive based on client orders, reinforcement learning can offer a flexible and adaptive approach to scheduling. By formulating the scheduling problem as a Markov decision process (MDP) and employing reinforcement learning techniques, such as Q-learning or deep reinforcement learning, it becomes possible to train an agent to make scheduling decisions based on the current state of the system. The agent can learn to balance various objectives, such as minimizing makespan, maximizing resource utilization, or meeting client deadlines,

while adapting to changing conditions in real-time.

Exploring the integration of reinforcement learning in TOOWHE can shed light on the feasibility, effectiveness, and advantages of using this approach for dynamic scheduling. It opens up possibilities for developing intelligent scheduling algorithms that can continuously adapt and optimize schedules as new jobs arrive, leading to improved efficiency and customer satisfaction.

In conclusion, exploring these potential future research directions might contribute to the field of scheduling optimization within the context of TOOWHE. By focusing on improving the computation of the lower bound in the branch and bound algorithm, investigating the application of constraint programming, and considering the integration of reinforcement learning, researchers can enhance scheduling practices and uncover novel approaches for optimizing schedules.

List of Figures

List of Tables

Bibliography

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1988.

[2] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401, 1988.

[3] David Applegate and William Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on computing*, 3(2):149–156, 1991.

[4] Arash Asadpour. An analysis of the weighted round robin rule in an online environment. Technical report, Working Paper, IOMS Department, Stern School of Business, New York University, 2015.

[5] Kenneth R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, 1974.

[6] Dimitris Bertsimas and Robert Weismantel. *Optimization over intergers.* Dynamic Ideas, 2005.

[7] APG Brown and ZA Lomnicki. Some applications of the "branch-and-bound" algorithm to the machine scheduling problem. *Journal of the Operational Research Society*, 17(2):173–186, 1966.

[8] Peter Brucker and Bernd Jurisch. A new lower bound for the job-shop scheduling problem. *European Journal of Operational Research*, 64(2):156–167, 1993.

[9] Xiaoqiang Q. Cai, Xianyi Wu, and Xian Zhou. *Optimal Stochastic Scheduling*, volume 207 of *Operations Research and Management Science.* F.S. Hillier, 2014.

[10] Jacques Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11:42–47, 1982.

[11] Jacques Carlier and Eric Pinson. An algorithm for solving job shop problem. *Management Science*, 35:164–176, 02 1989.

[12] Edward G. Coffman. *Computer and Job Shop Scheduling Theory*. John Wiley, 1976.

[13] Giacomo Da Col and Erich Teppan. Industrial-size job shop scheduling with constraint programming. *Operations Research Perspectives*, 9:100249, 08 2022.

[14] Mauro Dell'Amico and Marco Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41:231–252, 09 1993.

[15] Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.

[16] H. Fisher and G.L. Thompson. *Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules*. Prentice-Hall, 1963.

[17] Rudolf Fleischer and Michaela Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–355, 2000.

[18] Jose Framinan, Rainer Leisten, and Rubén García. *Manufacturing Scheduling Systems - An Integrated View on Models, Methods and Tools*. Springer, 11 2014.

[19] Simon French. *Sequencing and Scheduling: an Introduction to the Mathematics of the Job Shop*. Horwood, 1982.

[20] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023.

[21] Randolph Hall. *Handbook of Healthcare System Scheduling*, volume 168 of *Operations Research and Management Science*. F.S. Hillier, 2012.

[22] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science (New York, N.Y.)*, 220:671–80, 06 1983.

[23] Stephen Lawrence. Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement). *Graduate School of Industrial Administration, Carnegie-Mellon University*, 1984.

[24] Kangbok Lee, Joseph Y.-T. Leung, and Michael L. Pinedo. Makespan minimization in online scheduling with machine eligibility. *4OR*, 8(4):331–364,

2010.

[25] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[26] Nicole Megow, Marc Uetz, and Tjark Vredeveld. Models and algorithms for stochastic online scheduling. *Mathematics of Operations Research*, 31(3):513–525, 2006.

[27] John F Muth and Gerald L Thompson. *Industrial scheduling*. Prentice-Hall, 1963.

[28] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[29] Eugeniusz Nowicki and Czesław Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 04 2005.

[30] Michael Pinedo and Xiuli Chao. *Operations Scheduling with Applications in Manufacturing and Services*. Operations Scheduling with Applications in Manufacturing and Services. Irwin/McGraw-Hill, 1999.

[31] Michael L Pinedo. *Scheduling: Theory, Algorithms, And Systems*. Springer, 2015.

[32] Maurice Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58(1-3):263–285, 1993.

[33] R.A. Rutenbar. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine*, 5(1):19–26, 1989.

[34] Subhash C. Sarin, Balaji Nagarajan, and Lingrui Liao. *Stochastic Scheduling: Expectation-Variance Analysis of a Schedule*. Cambridge University Press, 2010.

[35] Andreas S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of lp-based heuristics and lower bounds. In William H. Cunningham, S. Thomas McCormick, and Maurice Queyranne, editors, *Integer Programming and Combinatorial Optimization*, pages 301–315, 1996.

[36] Yuri Sotskov, Nadezhda Sotskova, Tsung-Chyan Lai, and Frank Werner. *Scheduling under Uncertainty: Theory and Algorithms*. Belarusian Science,

2010.

[37] Yuri Sotskov and Frank Werner. *Sequencing and Scheduling with Inaccurate Data*. 01 2014.

[38] Robert H Storer, S David Wu, and Renzo Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management science*, 38(10):1495–1509, 1992.

[39] ED. Taillard. Parallel taboo search techniques for the job shop scheduling. *ORSA Journal on Computing*, 6(2):108–17, 1994.

[40] Eric Taillard. Benchmarks for basic scheduling problems. *european journal of operational research*, 64(2):278–285, 1993.

[41] Takeshi Yamada and Ryohei Nakano. A genetic algorithm applicable to large-scale job-shop problems. In *PPSN*, volume 2, pages 281–290, 1992.

[42] ChaoYong Zhang, PeiGen Li, ZaiLin Guan, and YunQing Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11):3229–3242, 2007.

A. Appendix

A.1. Simulated Annealing

A.1.1. First neighborhood

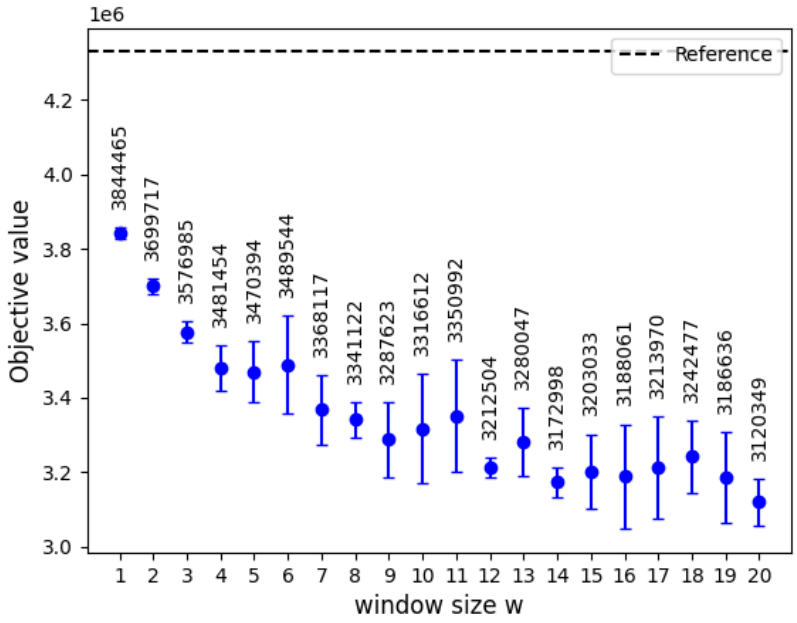


Figure A.1.: Instance 2: Mean objective value over 10 tests for different window sizes w using the first neighborhood

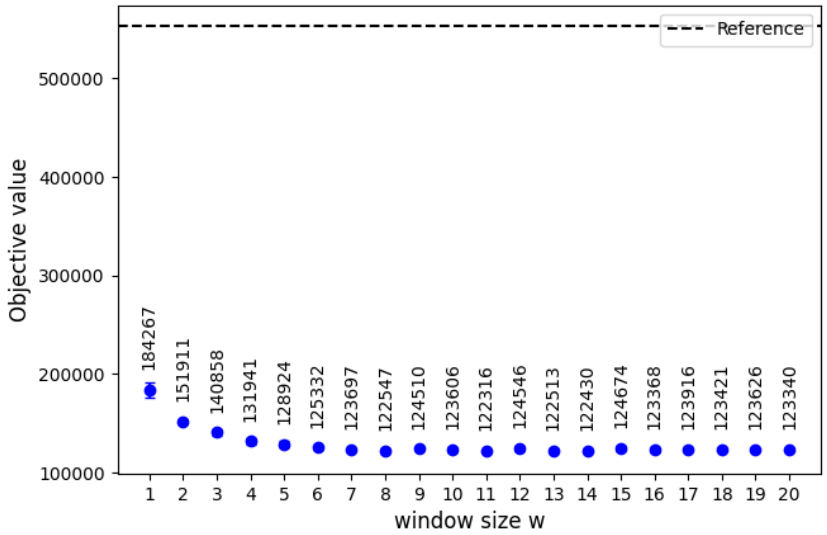


Figure A.2.: Instance 3: Mean objective value over 10 tests for different window sizes w using the first neighborhood

A.1.2. Second neighborhood

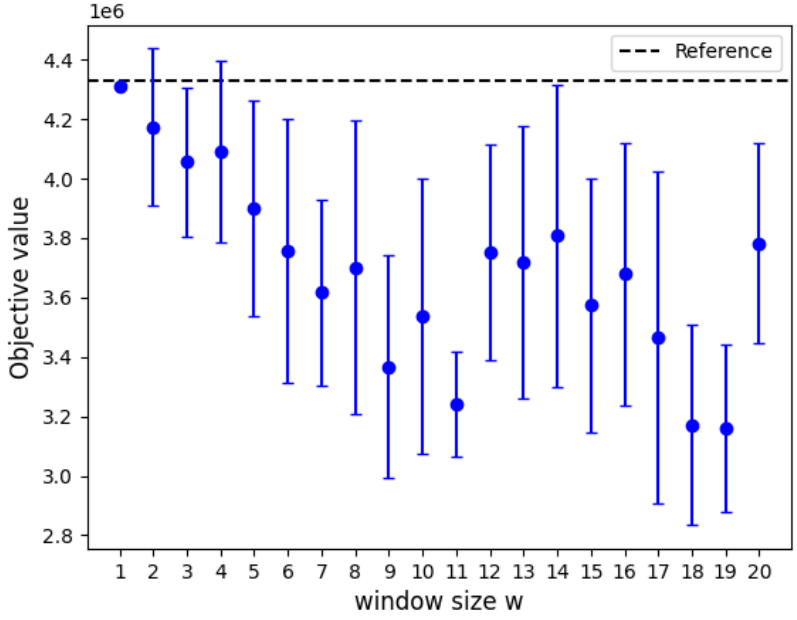


Figure A.3.: Instance 2: Mean objective value over 10 tests for different window sizes w using the second neighborhood

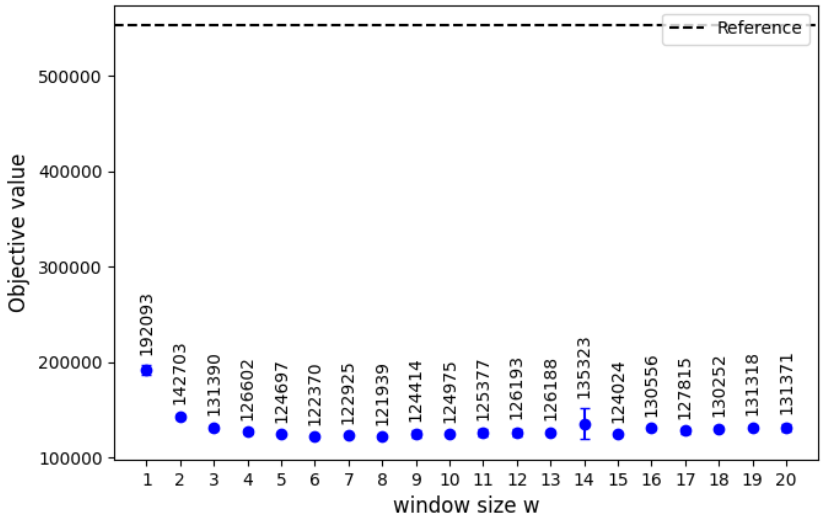


Figure A.4.: Instance 3: Mean objective value over 10 tests for different window sizes w using the second neighborhood

A.1.3. Third neighborhood

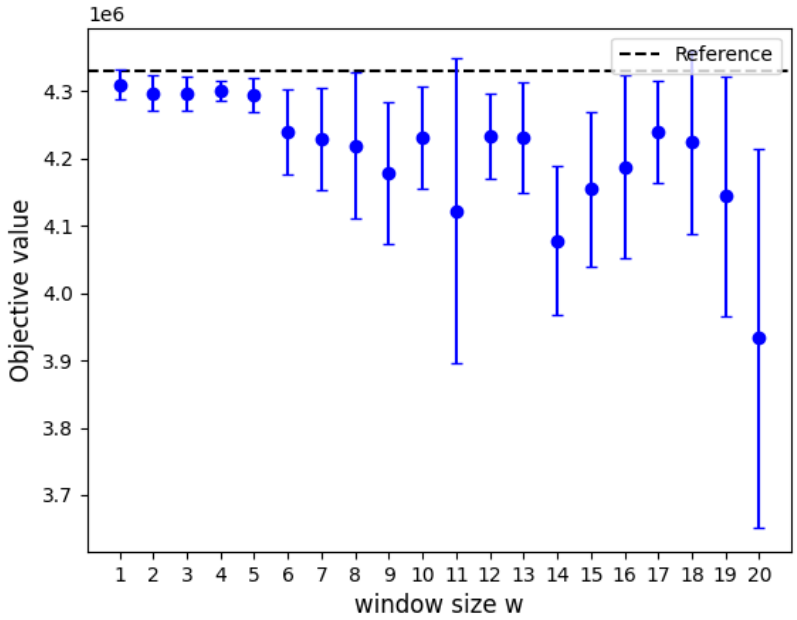


Figure A.5.: Instance 2: Mean objective value over 10 tests for different window sizes w using the third neighborhood

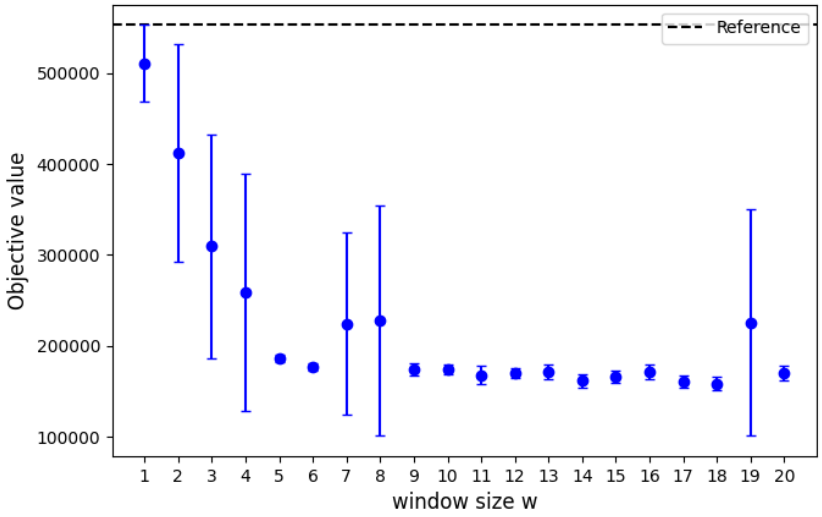


Figure A.6.: Instance 3: Mean objective value over 10 tests for different window sizes w using the third neighborhood

A.2. Ranking method

identification number	S	lower bound	objective	time (in s)	number of constraints
1	[0, 0, 0, 0, 0, 0, 0]	181927	3279809	0.0576	0
2	[0, 0, 0, 0, 0, 0, 1]	200244	3011397	0.0674	1544
3	[0, 0, 0, 0, 0, 1, 0]	193710	3083911	0.0668	1572
4	[0, 0, 0, 0, 0, 1, 1]	200467	2975769	0.0872	3116
5	[0, 0, 0, 0, 1, 0, 0]	189347	3203426	0.0716	1600
6	[0, 0, 0, 0, 1, 0, 1]	200699	2996660	0.084	3144
7	[0, 0, 0, 0, 1, 1, 0]	194292	3118612	0.0716	3172
8	[0, 0, 0, 0, 1, 1, 1]	200790	2986312	0.1036	4716
9	[0, 0, 0, 1, 0, 0, 0]	185748	3212681	0.0577	1631
10	[0, 0, 0, 1, 0, 0, 1]	200627	3011488	0.072	3175
11	[0, 0, 0, 1, 0, 1, 0]	194272	3094417	0.0801	3203
12	[0, 0, 0, 1, 0, 1, 1]	200791	3005745	0.0951	4747
13	[0, 0, 0, 1, 1, 0, 0]	189907	3231182	0.0829	3231
14	[0, 0, 0, 1, 1, 0, 1]	200950	2980719	0.1132	4775
15	[0, 0, 0, 1, 1, 1, 0]	194699	3120312	0.1061	4803
16	[0, 0, 0, 1, 1, 1, 1]	201025	3023492	0.1224	6347
17	[0, 0, 1, 0, 0, 0, 0]	825303	3310618	0.1852	1665
18	[0, 0, 1, 0, 0, 0, 1]	831428	3106474	0.2565	3209
19	[0, 0, 1, 0, 0, 1, 0]	829394	3140986	0.2364	3237
20	[0, 0, 1, 0, 0, 1, 1]	831631	3085004	0.2642	4781
21	[0, 0, 1, 0, 1, 0, 0]	828167	3272964	0.2368	3265
22	[0, 0, 1, 0, 1, 0, 1]	831753	3086324	0.2748	4809
23	[0, 0, 1, 0, 1, 1, 0]	829734	3148876	0.2677	4837
24	[0, 0, 1, 0, 1, 1, 1]	831859	3083040	0.2758	6381
25	[0, 0, 1, 1, 0, 0, 0]	826959	3281326	0.1997	3296
26	[0, 0, 1, 1, 0, 0, 1]	831807	3124380	0.2717	4840
27	[0, 0, 1, 1, 0, 1, 0]	829813	3166520	0.2646	4868
28	[0, 0, 1, 1, 0, 1, 1]	831949	3106081	0.2962	6412
29	[0, 0, 1, 1, 1, 0, 0]	828664	3297546	0.2733	4896
30	[0, 0, 1, 1, 1, 0, 1]	832033	3116899	0.2934	6440
31	[0, 0, 1, 1, 1, 1, 0]	830069	3166210	0.2828	6468
32	[0, 0, 1, 1, 1, 1, 1]	832110	3107793	0.3092	8012

Table A.1.: Instance 1: Study of the effect of the set S on the ranking method - Part 1

identification number	S	lower bound	objective	time (in s)	number of constraints
33	[0, 1, 0, 0, 0, 0, 0]	578794	3314103	0.0568	34
34	[0, 1, 0, 0, 0, 0, 1]	588741	3004586	0.0694	1578
35	[0, 1, 0, 0, 0, 1, 0]	585313	3119377	0.0716	1606
36	[0, 1, 0, 0, 0, 1, 1]	589064	3016931	0.0904	3150
37	[0, 1, 0, 0, 1, 0, 0]	583114	3248747	0.0653	1634
38	[0, 1, 0, 0, 1, 0, 1]	589148	3016893	0.0955	3178
39	[0, 1, 0, 0, 1, 1, 0]	585727	3144809	0.0858	3206
40	[0, 1, 0, 0, 1, 1, 1]	589330	3035357	0.098	4750
41	[0, 1, 0, 1, 0, 0, 0]	581034	3240448	0.07	1665
42	[0, 1, 0, 1, 0, 0, 1]	589143	3027669	0.0864	3209
43	[0, 1, 0, 1, 0, 1, 0]	585777	3138813	0.0935	3237
44	[0, 1, 0, 1, 0, 1, 1]	589381	3054901	0.1014	4781
45	[0, 1, 0, 1, 1, 0, 0]	583626	3276595	0.0801	3265
46	[0, 1, 0, 1, 1, 0, 1]	589427	3042301	0.101	4809
47	[0, 1, 0, 1, 1, 1, 0]	586084	3144104	0.1081	4837
48	[0, 1, 0, 1, 1, 1, 1]	589572	3063484	0.1123	6381
49	[0, 1, 1, 0, 0, 0, 0]	825305	3310618	0.1816	1699
50	[0, 1, 1, 0, 0, 0, 1]	831428	3106474	0.2383	3243
51	[0, 1, 1, 0, 0, 1, 0]	829394	3140986	0.2538	3271
52	[0, 1, 1, 0, 0, 1, 1]	831631	3085175	0.2714	4815
53	[0, 1, 1, 0, 1, 0, 0]	828167	3272710	0.2403	3299
54	[0, 1, 1, 0, 1, 0, 1]	831753	3093080	0.2695	4843
55	[0, 1, 1, 0, 1, 1, 0]	829735	3145579	0.2702	4871
56	[0, 1, 1, 0, 1, 1, 1]	831859	3086587	0.2856	6415
57	[0, 1, 1, 1, 0, 0, 0]	826961	3281326	0.2039	3330
58	[0, 1, 1, 1, 0, 0, 1]	831807	3124380	0.2585	4874
59	[0, 1, 1, 1, 0, 1, 0]	829813	3162888	0.261	4902
60	[0, 1, 1, 1, 0, 1, 1]	831949	3107589	0.2909	6446
61	[0, 1, 1, 1, 1, 0, 0]	828664	3300143	0.2621	4930
62	[0, 1, 1, 1, 1, 0, 1]	832033	3116899	0.2848	6474
63	[0, 1, 1, 1, 1, 1, 0]	830069	3162430	0.2988	6502
64	[0, 1, 1, 1, 1, 1, 1]	832110	3107793	0.3168	8046

Table A.2.: Instance 1: Study of the effect of the set S on the ranking method - Part 2

identification number	S	lower bound	objective	time (in s)	number of constraints
65	[1, 0, 0, 0, 0, 0, 0]	190646	3167833	0.9699	91417
66	[1, 0, 0, 0, 0, 0, 1]	202189	3006167	0.9962	92961
67	[1, 0, 0, 0, 0, 1, 0]	196635	3164387	1.0019	92989
68	[1, 0, 0, 0, 0, 1, 1]	202328	2977206	1.0135	94533
69	[1, 0, 0, 0, 1, 0, 0]	193222	3132126	0.9833	93017
70	[1, 0, 0, 0, 1, 0, 1]	202442	2969619	0.9988	94561
71	[1, 0, 0, 0, 1, 1, 0]	196992	3121287	0.988	94589
72	[1, 0, 0, 0, 1, 1, 1]	202505	2973191	1.0479	96133
73	[1, 0, 0, 1, 0, 0, 0]	190646	3167926	0.973	93048
74	[1, 0, 0, 1, 0, 0, 1]	202189	2991483	1.0146	94592
75	[1, 0, 0, 1, 0, 1, 0]	196635	3164455	1.0263	94620
76	[1, 0, 0, 1, 0, 1, 1]	202328	2978460	1.0628	96164
77	[1, 0, 0, 1, 1, 0, 0]	193222	3158710	1.0384	94648
78	[1, 0, 0, 1, 1, 0, 1]	202442	2974039	1.0513	96192
79	[1, 0, 0, 1, 1, 1, 0]	196992	3125697	1.0094	96220
80	[1, 0, 0, 1, 1, 1, 1]	202505	2979043	1.0112	97764
81	[1, 0, 1, 0, 0, 0, 0]	829153	3253629	1.1677	93082
82	[1, 0, 1, 0, 0, 0, 1]	833063	3109814	1.1802	94626
83	[1, 0, 1, 0, 0, 1, 0]	831247	3212501	1.1816	94654
84	[1, 0, 1, 0, 0, 1, 1]	833174	3095569	1.2177	96198
85	[1, 0, 1, 0, 1, 0, 0]	830349	3251148	1.1864	94682
86	[1, 0, 1, 0, 1, 0, 1]	833222	3096808	1.2211	96226
87	[1, 0, 1, 0, 1, 1, 0]	831465	3182889	1.2084	96254
88	[1, 0, 1, 0, 1, 1, 1]	833278	3109570	1.2412	97798
89	[1, 0, 1, 1, 0, 0, 0]	829153	3244039	1.1835	94713
90	[1, 0, 1, 1, 0, 0, 1]	833063	3125863	1.1889	96257
91	[1, 0, 1, 1, 0, 1, 0]	831247	3196506	1.2058	96285
92	[1, 0, 1, 1, 0, 1, 1]	833174	3094303	1.2066	97829
93	[1, 0, 1, 1, 1, 0, 0]	830349	3255099	1.1946	96313
94	[1, 0, 1, 1, 1, 0, 1]	833222	3098883	1.2588	97857
95	[1, 0, 1, 1, 1, 1, 0]	831465	3184476	1.2082	97885
96	[1, 0, 1, 1, 1, 1, 1]	833278	3093070	1.2275	99429

Table A.3.: Instance 1: Study of the effect of the set S on the ranking method - Part 3

identification number	S	lower bound	objective	time (in s)	number of constraints
97	[1, 1, 0, 0, 0, 0, 0]	584159	3193486	0.976	91451
98	[1, 1, 0, 0, 0, 0, 1]	590550	3003782	1.0529	92995
99	[1, 1, 0, 0, 0, 1, 0]	587488	3158041	1.0109	93023
100	[1, 1, 0, 0, 0, 1, 1]	590732	3013860	1.0199	94567
101	[1, 1, 0, 0, 1, 0, 0]	585836	3183831	0.9571	93051
102	[1, 1, 0, 0, 1, 0, 1]	590749	2996877	1.0053	94595
103	[1, 1, 0, 0, 1, 1, 0]	587753	3160657	0.9993	94623
104	[1, 1, 0, 0, 1, 1, 1]	590854	3033677	1.0251	96167
105	[1, 1, 0, 1, 0, 0, 0]	584159	3193410	0.9658	93082
106	[1, 1, 0, 1, 0, 0, 1]	590550	3012728	1.0012	94626
107	[1, 1, 0, 1, 0, 1, 0]	587488	3159881	0.9937	94654
108	[1, 1, 0, 1, 0, 1, 1]	590732	3008064	1.0149	96198
109	[1, 1, 0, 1, 1, 0, 0]	585836	3181848	1.0101	94682
110	[1, 1, 0, 1, 1, 0, 1]	590749	2991563	1.0327	96226
111	[1, 1, 0, 1, 1, 1, 0]	587753	3137638	1.0182	96254
112	[1, 1, 0, 1, 1, 1, 1]	590854	3022448	1.0314	97798
113	[1, 1, 1, 0, 0, 0, 0]	829155	3253597	1.1521	93116
114	[1, 1, 1, 0, 0, 0, 1]	833063	3121792	1.1972	94660
115	[1, 1, 1, 0, 0, 1, 0]	831247	3193084	1.166	94688
116	[1, 1, 1, 0, 0, 1, 1]	833174	3088489	1.1943	96232
117	[1, 1, 1, 0, 1, 0, 0]	830351	3250560	1.1664	94716
118	[1, 1, 1, 0, 1, 0, 1]	833222	3096989	1.2104	96260
119	[1, 1, 1, 0, 1, 1, 0]	831465	3186501	1.1835	96288
120	[1, 1, 1, 0, 1, 1, 1]	833278	3100603	1.2032	97832
121	[1, 1, 1, 1, 0, 0, 0]	829155	3253581	1.1697	94747
122	[1, 1, 1, 1, 0, 0, 1]	833063	3131683	1.1978	96291
123	[1, 1, 1, 1, 0, 1, 0]	831247	3208796	1.193	96319
124	[1, 1, 1, 1, 0, 1, 1]	833174	3110636	1.1977	97863
125	[1, 1, 1, 1, 1, 0, 0]	830351	3255234	1.1731	96347
126	[1, 1, 1, 1, 1, 0, 1]	833222	3114926	1.2192	97891
127	[1, 1, 1, 1, 1, 1, 0]	831465	3197513	1.1778	97919
128	[1, 1, 1, 1, 1, 1, 1]	833278	3111996	1.2166	99463

Table A.4.: Instance 1: Study of the effect of the set S on the ranking method - Part 4

identification number	S	lower bound	objective	time (in s)	number of constraints
1	[0, 0, 0, 0]	181927	3279809	0.063	0
2	[0, 0, 0, 1]	185748	3212681	0.0638	1631
3	[0, 0, 1, 0]	825303	3310618	0.1903	1665
4	[0, 0, 1, 1]	826959	3281326	0.2299	3296
5	[0, 1, 0, 0]	578794	3314103	0.0535	34
6	[0, 1, 0, 1]	581034	3240448	0.0759	1665
7	[0, 1, 1, 0]	825305	3310618	0.1989	1699
8	[0, 1, 1, 1]	826961	3281326	0.2418	3330
9	[1, 0, 0, 0]	190646	3167833	1.0445	91417
10	[1, 0, 0, 1]	190646	3167926	1.0514	93048
11	[1, 0, 1, 0]	829153	3253629	1.2909	93082
12	[1, 0, 1, 1]	829153	3244039	1.2453	94713
13	[1, 1, 0, 0]	584159	3193486	1.0392	91451
14	[1, 1, 0, 1]	584159	3193410	1.0799	93082
15	[1, 1, 1, 0]	829155	3253597	1.2829	93116
16	[1, 1, 1, 1]	829155	3253581	1.3056	94747

Table A.5.: Instance 1: Study of the effect of the set S on the ranking method with only the subset 1, 2, 3 and 4.

identification number	S	lower bound	objective	time (in s)	number of constraints
1	[0, 0, 0, 0, 0, 0, 0]	189079	3237570	0.0535	0
2	[0, 0, 0, 0, 0, 0, 1]	237606	3098261	0.0779	1540
3	[0, 0, 0, 0, 0, 1, 0]	225501	3049275	0.0686	1569
4	[0, 0, 0, 0, 0, 1, 1]	238698	3013458	0.1274	3109
5	[0, 0, 0, 0, 1, 0, 0]	213192	3047757	0.0783	1598
6	[0, 0, 0, 0, 1, 0, 1]	238745	2973546	0.1079	3138
7	[0, 0, 0, 0, 1, 1, 0]	226320	2970630	0.0977	3167
8	[0, 0, 0, 0, 1, 1, 1]	239266	3000039	0.134	4707
9	[0, 0, 0, 1, 0, 0, 0]	200976	3133319	0.084	1628
10	[0, 0, 0, 1, 0, 0, 1]	238602	3052526	0.098	3168
11	[0, 0, 0, 1, 0, 1, 0]	226271	3051347	0.095	3197
12	[0, 0, 0, 1, 0, 1, 1]	239385	3015376	0.1298	4737
13	[0, 0, 0, 1, 1, 0, 0]	213694	3100204	0.096	3226
14	[0, 0, 0, 1, 1, 0, 1]	239249	3038978	0.1336	4766
15	[0, 0, 0, 1, 1, 1, 0]	226804	2979578	0.1208	4795
16	[0, 0, 0, 1, 1, 1, 1]	239733	2980083	0.1463	6335
17	[0, 0, 1, 0, 0, 0, 0]	857209	3298164	0.2286	1660
18	[0, 0, 1, 0, 0, 0, 1]	879054	3162389	0.2677	3200
19	[0, 0, 1, 0, 0, 1, 0]	874292	3166683	0.2681	3229
20	[0, 0, 1, 0, 0, 1, 1]	880330	3109709	0.3045	4769
21	[0, 0, 1, 0, 1, 0, 0]	868985	3113918	0.263	3258
22	[0, 0, 1, 0, 1, 0, 1]	880405	3036858	0.3002	4798
23	[0, 0, 1, 0, 1, 1, 0]	875158	3124266	0.2919	4827
24	[0, 0, 1, 0, 1, 1, 1]	881002	3016258	0.3334	6367
25	[0, 0, 1, 1, 0, 0, 0]	863151	3205488	0.2848	3288
26	[0, 0, 1, 1, 0, 0, 1]	880243	3106458	0.3401	4828
27	[0, 0, 1, 1, 0, 1, 0]	875078	3182677	0.313	4857
28	[0, 0, 1, 1, 0, 1, 1]	881069	3100280	0.3134	6397
29	[0, 0, 1, 1, 1, 0, 0]	869523	3124783	0.288	4886
30	[0, 0, 1, 1, 1, 0, 1]	880945	3068304	0.3171	6426
31	[0, 0, 1, 1, 1, 1, 0]	875623	3143168	0.3156	6455
32	[0, 0, 1, 1, 1, 1, 1]	881483	3021505	0.3767	7995

Table A.6.: Instance 2: Study of the effect of the set S on the ranking method - Part 1

identification number	S	lower bound	objective	time (in s)	number of constraints
33	[0, 1, 0, 0, 0, 0, 0]	609123	3245370	0.0673	32
34	[0, 1, 0, 0, 0, 0, 1]	639082	3077052	0.0839	1572
35	[0, 1, 0, 0, 0, 1, 0]	632559	3133552	0.0829	1601
36	[0, 1, 0, 0, 0, 1, 1]	640527	3044327	0.1114	3141
37	[0, 1, 0, 0, 1, 0, 0]	625060	3097306	0.0815	1630
38	[0, 1, 0, 0, 1, 0, 1]	640542	2992689	0.1029	3170
39	[0, 1, 0, 0, 1, 1, 0]	633481	3042637	0.1256	3199
40	[0, 1, 0, 0, 1, 1, 1]	641241	2933748	0.146	4739
41	[0, 1, 0, 1, 0, 0, 0]	617167	3159414	0.0955	1660
42	[0, 1, 0, 1, 0, 0, 1]	640304	3077904	0.1103	3200
43	[0, 1, 0, 1, 0, 1, 0]	633385	3125316	0.0961	3229
44	[0, 1, 0, 1, 0, 1, 1]	641287	3035575	0.1387	4769
45	[0, 1, 0, 1, 1, 0, 0]	625611	3130348	0.099	3258
46	[0, 1, 0, 1, 1, 0, 1]	641085	3029928	0.1165	4798
47	[0, 1, 0, 1, 1, 1, 0]	633974	3052090	0.1438	4827
48	[0, 1, 0, 1, 1, 1, 1]	641731	2944475	0.1531	6367
49	[0, 1, 1, 0, 0, 0, 0]	857211	3298164	0.2294	1692
50	[0, 1, 1, 0, 0, 0, 1]	879055	3159102	0.2676	3232
51	[0, 1, 1, 0, 0, 1, 0]	874294	3162337	0.2753	3261
52	[0, 1, 1, 0, 0, 1, 1]	880333	3112462	0.2914	4801
53	[0, 1, 1, 0, 1, 0, 0]	868987	3113716	0.2543	3290
54	[0, 1, 1, 0, 1, 0, 1]	880408	3040749	0.3018	4830
55	[0, 1, 1, 0, 1, 1, 0]	875161	3124266	0.3176	4859
56	[0, 1, 1, 0, 1, 1, 1]	881005	3016241	0.317	6399
57	[0, 1, 1, 1, 0, 0, 0]	863153	3195684	0.2836	3320
58	[0, 1, 1, 1, 0, 0, 1]	880245	3128095	0.2944	4860
59	[0, 1, 1, 1, 0, 1, 0]	875080	3183084	0.2863	4889
60	[0, 1, 1, 1, 0, 1, 1]	881071	3096483	0.331	6429
61	[0, 1, 1, 1, 1, 0, 0]	869525	3116156	0.2675	4918
62	[0, 1, 1, 1, 1, 0, 1]	880947	3068973	0.3358	6458
63	[0, 1, 1, 1, 1, 1, 0]	875626	3129051	0.3172	6487
64	[0, 1, 1, 1, 1, 1, 1]	881486	3018228	0.356	8027

Table A.7.: Instance 2: Study of the effect of the set S on the ranking method - Part 2

identification number	S	lower bound	objective	time (in s)	number of constraints
65	[1, 0, 0, 0, 0, 0, 0]	205260	3123781	1.0626	91875
66	[1, 0, 0, 0, 0, 0, 1]	239573	3012939	1.1507	93415
67	[1, 0, 0, 0, 0, 1, 0]	227372	3055429	1.16	93444
68	[1, 0, 0, 0, 0, 1, 1]	240269	2954977	1.1843	94984
69	[1, 0, 0, 0, 1, 0, 0]	215404	3113334	1.1282	93473
70	[1, 0, 0, 0, 1, 0, 1]	240071	3032434	1.1611	95013
71	[1, 0, 0, 0, 1, 1, 0]	227752	3011248	1.1692	95042
72	[1, 0, 0, 0, 1, 1, 1]	240510	2971108	1.1895	96582
73	[1, 0, 0, 1, 0, 0, 0]	205260	3123859	1.0626	93503
74	[1, 0, 0, 1, 0, 0, 1]	239573	3005797	1.1818	95043
75	[1, 0, 0, 1, 0, 1, 0]	227372	3056065	1.1564	95072
76	[1, 0, 0, 1, 0, 1, 1]	240269	2967244	1.1902	96612
77	[1, 0, 0, 1, 1, 0, 0]	215404	3114341	1.1248	95101
78	[1, 0, 0, 1, 1, 0, 1]	240071	2977161	1.1768	96641
79	[1, 0, 0, 1, 1, 1, 0]	227752	2976273	1.1839	96670
80	[1, 0, 0, 1, 1, 1, 1]	240510	2992303	1.1902	98210
81	[1, 0, 1, 0, 0, 0, 0]	865194	3181245	1.2294	93535
82	[1, 0, 1, 0, 0, 0, 1]	881106	3064991	1.3413	95075
83	[1, 0, 1, 0, 0, 1, 0]	875941	3134435	1.3285	95104
84	[1, 0, 1, 0, 0, 1, 1]	881865	3069317	1.3444	96644
85	[1, 0, 1, 0, 1, 0, 0]	870508	3152538	1.3283	95133
86	[1, 0, 1, 0, 1, 0, 1]	881675	3037791	1.357	96673
87	[1, 0, 1, 0, 1, 1, 0]	876350	3124734	1.353	96702
88	[1, 0, 1, 0, 1, 1, 1]	882171	3022253	1.3846	98242
89	[1, 0, 1, 1, 0, 0, 0]	865194	3182241	1.2339	95163
90	[1, 0, 1, 1, 0, 0, 1]	881106	3061150	1.3326	96703
91	[1, 0, 1, 1, 0, 1, 0]	875941	3145133	1.3542	96732
92	[1, 0, 1, 1, 0, 1, 1]	881865	3052055	1.359	98272
93	[1, 0, 1, 1, 1, 0, 0]	870508	3154222	1.3373	96761
94	[1, 0, 1, 1, 1, 0, 1]	881675	3050714	1.372	98301
95	[1, 0, 1, 1, 1, 1, 0]	876350	3121366	1.3609	98330
96	[1, 0, 1, 1, 1, 1, 1]	882171	3027301	1.3876	99870

Table A.8.: Instance 2: Study of the effect of the set S on the ranking method - Part 3

identification number	S	lower bound	objective	time (in s)	number of constraints
97	[1, 1, 0, 0, 0, 0, 0]	619829	3133408	1.0454	91907
98	[1, 1, 0, 0, 0, 0, 1]	641337	3009736	1.1653	93447
99	[1, 1, 0, 0, 0, 1, 0]	634402	3058820	1.1544	93476
100	[1, 1, 0, 0, 0, 1, 1]	642207	2990305	1.18	95016
101	[1, 1, 0, 0, 1, 0, 0]	626889	3113610	1.1369	93505
102	[1, 1, 0, 0, 1, 0, 1]	641943	2986113	1.1747	95045
103	[1, 1, 0, 0, 1, 1, 0]	634843	3007289	1.1586	95074
104	[1, 1, 0, 0, 1, 1, 1]	642523	2925468	1.1962	96614
105	[1, 1, 0, 1, 0, 0, 0]	619829	3133408	1.0637	93535
106	[1, 1, 0, 1, 0, 0, 1]	641337	3017836	1.1685	95075
107	[1, 1, 0, 1, 0, 1, 0]	634402	3053092	1.1529	95104
108	[1, 1, 0, 1, 0, 1, 1]	642207	2982238	1.218	96644
109	[1, 1, 0, 1, 1, 0, 0]	626889	3112490	1.1525	95133
110	[1, 1, 0, 1, 1, 0, 1]	641943	2984605	1.2047	96673
111	[1, 1, 0, 1, 1, 1, 0]	634843	2996875	1.1644	96702
112	[1, 1, 0, 1, 1, 1, 1]	642523	2934815	1.2154	98242
113	[1, 1, 1, 0, 0, 0, 0]	865196	3182549	1.2362	93567
114	[1, 1, 1, 0, 0, 0, 1]	881109	3057696	1.3681	95107
115	[1, 1, 1, 0, 0, 1, 0]	875944	3143896	1.3442	95136
116	[1, 1, 1, 0, 0, 1, 1]	881868	3072141	1.348	96676
117	[1, 1, 1, 0, 1, 0, 0]	870510	3156516	1.352	95165
118	[1, 1, 1, 0, 1, 0, 1]	881678	3053756	1.3831	96705
119	[1, 1, 1, 0, 1, 1, 0]	876353	3086074	1.4462	96734
120	[1, 1, 1, 0, 1, 1, 1]	882174	3039813	1.3948	98274
121	[1, 1, 1, 1, 0, 0, 0]	865196	3183379	1.2427	95195
122	[1, 1, 1, 1, 0, 0, 1]	881109	3065287	1.3439	96735
123	[1, 1, 1, 1, 0, 1, 0]	875944	3131518	1.3213	96764
124	[1, 1, 1, 1, 0, 1, 1]	881868	3078819	1.3627	98304
125	[1, 1, 1, 1, 1, 0, 0]	870510	3153701	1.3169	96793
126	[1, 1, 1, 1, 1, 0, 1]	881678	3058526	1.3459	98333
127	[1, 1, 1, 1, 1, 1, 0]	876353	3087074	1.3622	98362
128	[1, 1, 1, 1, 1, 1, 1]	882174	3047629	1.3888	99902

Table A.9.: Instance 2: Study of the effect of the set S on the ranking method - Part 4

identification number	S	lower bound	objective	time (in s)	number of constraints
1	[0, 0, 0, 0]	189079	3237570	0.0485	0
2	[0, 0, 0, 1]	237606	3098261	0.0956	1628
3	[0, 0, 1, 0]	225501	3049275	0.0815	1660
4	[0, 0, 1, 1]	238698	3013458	0.1014	3288
5	[0, 1, 0, 0]	213192	3047757	0.0754	32
6	[0, 1, 0, 1]	238745	2973546	0.0952	1660
7	[0, 1, 1, 0]	226320	2970630	0.0961	1692
8	[0, 1, 1, 1]	239266	3000039	0.123	3320
9	[1, 0, 0, 0]	200976	3133319	1.0964	91875
10	[1, 0, 0, 1]	238602	3052526	1.1202	93503
11	[1, 0, 1, 0]	226271	3051347	1.0976	93535
12	[1, 0, 1, 1]	239385	3015376	1.1348	95163
13	[1, 1, 0, 0]	213694	3100204	1.0903	91907
14	[1, 1, 0, 1]	239249	3038978	1.138	93535
15	[1, 1, 1, 0]	226804	2979578	1.2218	93567
16	[1, 1, 1, 1]	239733	2980083	1.3482	95195

Table A.10.: Instance 2: Study of the effect of the set S on the ranking method with only the subset 1, 2, 3 and 4.

identification number	S	lower bound	objective	time (in s)	number of constraints
1	[0, 0, 0, 0, 0, 0, 0]	12669	159814	0.0158	0
2	[0, 0, 0, 0, 0, 0, 1]	22701	140734	0.0204	427
3	[0, 0, 0, 0, 0, 1, 0]	20541	140187	0.0266	449
4	[0, 0, 0, 0, 0, 1, 1]	23076	131632	0.0314	876
5	[0, 0, 0, 0, 1, 0, 0]	17784	146796	0.0181	471
6	[0, 0, 0, 0, 1, 0, 1]	23091	136662	0.0388	898
7	[0, 0, 0, 0, 1, 1, 0]	20661	143143	0.0294	920
8	[0, 0, 0, 0, 1, 1, 1]	23208	134862	0.0338	1347
9	[0, 0, 0, 1, 0, 0, 0]	15178	149975	0.0297	500
10	[0, 0, 0, 1, 0, 0, 1]	23002	133179	0.0343	927
11	[0, 0, 0, 1, 0, 1, 0]	20692	144045	0.0284	949
12	[0, 0, 0, 1, 0, 1, 1]	23234	133650	0.0286	1376
13	[0, 0, 0, 1, 1, 0, 0]	17884	142453	0.048	971
14	[0, 0, 0, 1, 1, 0, 1]	23187	137243	0.0476	1398
15	[0, 0, 0, 1, 1, 1, 0]	20754	145183	0.0344	1420
16	[0, 0, 0, 1, 1, 1, 1]	23298	136392	0.0485	1847
17	[0, 0, 1, 0, 0, 0, 0]	75202	160731	0.0473	534
18	[0, 0, 1, 0, 0, 0, 1]	76933	146371	0.0463	961
19	[0, 0, 1, 0, 0, 1, 0]	76802	145838	0.0439	983
20	[0, 0, 1, 0, 0, 1, 1]	77292	146064	0.0498	1410
21	[0, 0, 1, 0, 1, 0, 0]	76311	155893	0.0463	1005
22	[0, 0, 1, 0, 1, 0, 1]	77244	148970	0.0475	1432
23	[0, 0, 1, 0, 1, 1, 0]	76899	150618	0.0419	1454
24	[0, 0, 1, 0, 1, 1, 1]	77370	150679	0.0693	1881
25	[0, 0, 1, 1, 0, 0, 0]	75855	160992	0.047	1034
26	[0, 0, 1, 1, 0, 0, 1]	77183	147309	0.0519	1461
27	[0, 0, 1, 1, 0, 1, 0]	76942	145778	0.0494	1483
28	[0, 0, 1, 1, 0, 1, 1]	77423	147415	0.0578	1910
29	[0, 0, 1, 1, 1, 0, 0]	76434	151706	0.0478	1505
30	[0, 0, 1, 1, 1, 0, 1]	77350	147391	0.0815	1932
31	[0, 0, 1, 1, 1, 1, 0]	77001	146096	0.0595	1954
32	[0, 0, 1, 1, 1, 1, 1]	77468	147798	0.0479	2381

Table A.11.: Instance 3: Study of the effect of the set S on the ranking method - Part 1

identification number	S	lower bound	objective	time (in s)	number of constraints
33	[0, 1, 0, 0, 0, 0, 0]	57558	175288	0.0215	34
34	[0, 1, 0, 0, 0, 0, 1]	61161	149950	0.0167	461
35	[0, 1, 0, 0, 0, 1, 0]	60614	153708	0.031	483
36	[0, 1, 0, 0, 0, 1, 1]	61585	146130	0.0384	910
37	[0, 1, 0, 0, 1, 0, 0]	59720	158412	0.0141	505
38	[0, 1, 0, 0, 1, 0, 1]	61593	143607	0.0317	932
39	[0, 1, 0, 0, 1, 1, 0]	60782	148935	0.0424	954
40	[0, 1, 0, 0, 1, 1, 1]	61756	147168	0.0415	1381
41	[0, 1, 0, 1, 0, 0, 0]	58754	160804	0.0061	534
42	[0, 1, 0, 1, 0, 0, 1]	61482	146704	0.0316	961
43	[0, 1, 0, 1, 0, 1, 0]	60819	152965	0.0348	983
44	[0, 1, 0, 1, 0, 1, 1]	61765	150887	0.0405	1410
45	[0, 1, 0, 1, 1, 0, 0]	59864	155085	0.0278	1005
46	[0, 1, 0, 1, 1, 0, 1]	61701	143886	0.058	1432
47	[0, 1, 0, 1, 1, 1, 0]	60908	153896	0.035	1454
48	[0, 1, 0, 1, 1, 1, 1]	61857	146370	0.0469	1881
49	[0, 1, 1, 0, 0, 0, 0]	75203	162041	0.0226	568
50	[0, 1, 1, 0, 0, 0, 1]	76933	146371	0.0455	995
51	[0, 1, 1, 0, 0, 1, 0]	76804	145838	0.0455	1017
52	[0, 1, 1, 0, 0, 1, 1]	77292	145895	0.0585	1444
53	[0, 1, 1, 0, 1, 0, 0]	76314	156204	0.0431	1039
54	[0, 1, 1, 0, 1, 0, 1]	77244	149132	0.0527	1466
55	[0, 1, 1, 0, 1, 1, 0]	76901	150578	0.05	1488
56	[0, 1, 1, 0, 1, 1, 1]	77370	150522	0.0485	1915
57	[0, 1, 1, 1, 0, 0, 0]	75856	162279	0.0475	1068
58	[0, 1, 1, 1, 0, 0, 1]	77183	147154	0.0589	1495
59	[0, 1, 1, 1, 0, 1, 0]	76944	145719	0.0533	1517
60	[0, 1, 1, 1, 0, 1, 1]	77423	147141	0.0686	1944
61	[0, 1, 1, 1, 1, 0, 0]	76436	151666	0.0422	1539
62	[0, 1, 1, 1, 1, 0, 1]	77350	147387	0.0635	1966
63	[0, 1, 1, 1, 1, 1, 0]	77004	146096	0.0588	1988
64	[0, 1, 1, 1, 1, 1, 1]	77468	147794	0.0632	2415

Table A.12.: Instance 3: Study of the effect of the set S on the ranking method - Part 2

identification number	S	lower bound	objective	time (in s)	number of constraints
65	[1, 0, 0, 0, 0, 0, 0]	15951	203892	0.1267	8352
66	[1, 0, 0, 0, 0, 0, 1]	23410	148842	0.1366	8779
67	[1, 0, 0, 0, 0, 1, 0]	21130	160501	0.1373	8801
68	[1, 0, 0, 0, 0, 1, 1]	23592	147646	0.1393	9228
69	[1, 0, 0, 0, 1, 0, 0]	18373	166155	0.1423	8823
70	[1, 0, 0, 0, 1, 0, 1]	23523	146150	0.1386	9250
71	[1, 0, 0, 0, 1, 1, 0]	21153	159754	0.1411	9272
72	[1, 0, 0, 0, 1, 1, 1]	23617	146761	0.1527	9699
73	[1, 0, 0, 1, 0, 0, 0]	15951	203897	0.1403	8852
74	[1, 0, 0, 1, 0, 0, 1]	23410	147067	0.1366	9279
75	[1, 0, 0, 1, 0, 1, 0]	21130	160302	0.1371	9301
76	[1, 0, 0, 1, 0, 1, 1]	23592	146581	0.1514	9728
77	[1, 0, 0, 1, 1, 0, 0]	18373	163239	0.1387	9323
78	[1, 0, 0, 1, 1, 0, 1]	23523	146551	0.1406	9750
79	[1, 0, 0, 1, 1, 1, 0]	21153	148269	0.1424	9772
80	[1, 0, 0, 1, 1, 1, 1]	23617	149365	0.167	10199
81	[1, 0, 1, 0, 0, 0, 0]	76205	179931	0.165	8886
82	[1, 0, 1, 0, 0, 0, 1]	77428	163532	0.1678	9313
83	[1, 0, 1, 0, 0, 1, 0]	77151	161543	0.1614	9335
84	[1, 0, 1, 0, 0, 1, 1]	77613	159253	0.1838	9762
85	[1, 0, 1, 0, 1, 0, 0]	76655	164978	0.1644	9357
86	[1, 0, 1, 0, 1, 0, 1]	77535	163226	0.1672	9784
87	[1, 0, 1, 0, 1, 1, 0]	77184	163150	0.1646	9806
88	[1, 0, 1, 0, 1, 1, 1]	77638	159224	0.196	10233
89	[1, 0, 1, 1, 0, 0, 0]	76205	180014	0.1566	9386
90	[1, 0, 1, 1, 0, 0, 1]	77428	163870	0.1719	9813
91	[1, 0, 1, 1, 0, 1, 0]	77151	162020	0.1644	9835
92	[1, 0, 1, 1, 0, 1, 1]	77613	159253	0.1878	10262
93	[1, 0, 1, 1, 1, 0, 0]	76655	164977	0.1875	9857
94	[1, 0, 1, 1, 1, 0, 1]	77535	163452	0.1826	10284
95	[1, 0, 1, 1, 1, 1, 0]	77184	163150	0.1685	10306
96	[1, 0, 1, 1, 1, 1, 1]	77638	159206	0.2009	10733

Table A.13.: Instance 3: Study of the effect of the set S on the ranking method - Part 3

identification number	S	lower bound	objective	time (in s)	number of constraints
97	[1, 1, 0, 0, 0, 0, 0]	59199	213596	0.131	8386
98	[1, 1, 0, 0, 0, 0, 1]	61742	171816	0.118	8813
99	[1, 1, 0, 0, 0, 1, 0]	61057	174391	0.1282	8835
100	[1, 1, 0, 0, 0, 1, 1]	61967	164228	0.1423	9262
101	[1, 1, 0, 0, 1, 0, 0]	60140	176489	0.1377	8857
102	[1, 1, 0, 0, 1, 0, 1]	61897	159626	0.1322	9284
103	[1, 1, 0, 0, 1, 1, 0]	61117	167295	0.1425	9306
104	[1, 1, 0, 0, 1, 1, 1]	62034	158731	0.1656	9733
105	[1, 1, 0, 1, 0, 0, 0]	59199	213596	0.1516	8886
106	[1, 1, 0, 1, 0, 0, 1]	61742	171482	0.1358	9313
107	[1, 1, 0, 1, 0, 1, 0]	61057	174386	0.1386	9335
108	[1, 1, 0, 1, 0, 1, 1]	61967	164083	0.1661	9762
109	[1, 1, 0, 1, 1, 0, 0]	60140	176503	0.1329	9357
110	[1, 1, 0, 1, 1, 0, 1]	61897	159626	0.1469	9784
111	[1, 1, 0, 1, 1, 1, 0]	61117	167192	0.1465	9806
112	[1, 1, 0, 1, 1, 1, 1]	62034	158607	0.1568	10233
113	[1, 1, 1, 0, 0, 0, 0]	76206	179931	0.1447	8920
114	[1, 1, 1, 0, 0, 0, 1]	77428	163860	0.1797	9347
115	[1, 1, 1, 0, 0, 1, 0]	77153	162023	0.1516	9369
116	[1, 1, 1, 0, 0, 1, 1]	77613	159256	0.1825	9796
117	[1, 1, 1, 0, 1, 0, 0]	76657	162473	0.1672	9391
118	[1, 1, 1, 0, 1, 0, 1]	77535	163378	0.1659	9818
119	[1, 1, 1, 0, 1, 1, 0]	77185	163149	0.1703	9840
120	[1, 1, 1, 0, 1, 1, 1]	77638	159220	0.1583	10267
121	[1, 1, 1, 1, 0, 0, 0]	76206	179931	0.1618	9420
122	[1, 1, 1, 1, 0, 0, 1]	77428	163341	0.1602	9847
123	[1, 1, 1, 1, 0, 1, 0]	77153	162020	0.1751	9869
124	[1, 1, 1, 1, 0, 1, 1]	77613	159427	0.174	10296
125	[1, 1, 1, 1, 1, 0, 0]	76657	162472	0.1634	9891
126	[1, 1, 1, 1, 1, 0, 1]	77535	157696	0.1753	10318
127	[1, 1, 1, 1, 1, 1, 0]	77185	163145	0.1771	10340
128	[1, 1, 1, 1, 1, 1, 1]	77638	159214	0.1965	10767

Table A.14.: Instance 3: Study of the effect of the set S on the ranking method - Part 4

identification number	S	lower bound	objective	time (in s)	number of constraints
1	[0, 0, 0, 0]	12669	159814	0.0189	0
2	[0, 0, 0, 1]	15178	149975	0.0121	500
3	[0, 0, 1, 0]	75202	160731	0.0451	534
4	[0, 0, 1, 1]	75855	160992	0.0486	1034
5	[0, 1, 0, 0]	57558	175288	0.0298	34
6	[0, 1, 0, 1]	58754	160804	0.0161	534
7	[0, 1, 1, 0]	75203	162041	0.0465	568
8	[0, 1, 1, 1]	75856	162279	0.0477	1068
9	[1, 0, 0, 0]	15951	203892	0.134	8352
10	[1, 0, 0, 1]	15951	203897	0.1431	8852
11	[1, 0, 1, 0]	76205	179931	0.1492	8886
12	[1, 0, 1, 1]	76205	180014	0.1752	9386
13	[1, 1, 0, 0]	59199	213596	0.1333	8386
14	[1, 1, 0, 1]	59199	213596	0.1509	8886
15	[1, 1, 1, 0]	76206	179931	0.1521	8920
16	[1, 1, 1, 1]	76206	179931	0.1808	9420

Table A.15.: Instance 3: Study of the effect of the set S on the ranking method with only the subset 1, 2, 3 and 4.

A.2.1. About the lower bound

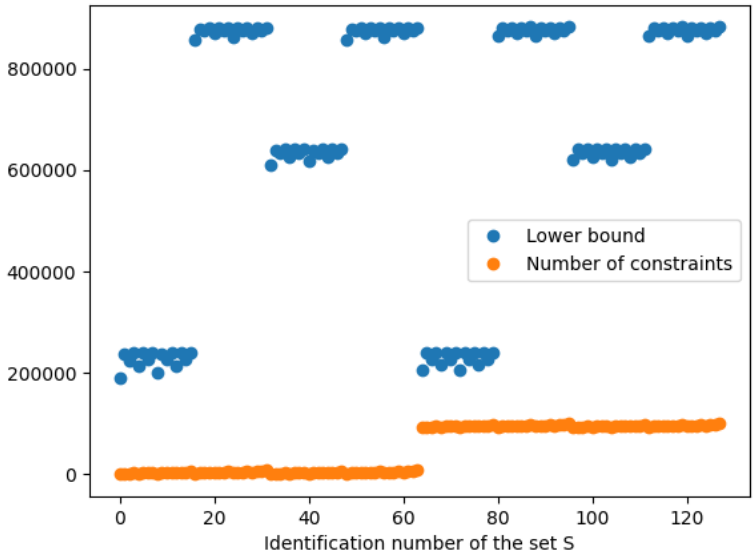


Figure A.7.: Instance 2: Values of the lower bound and number of constraints of the ranking method over different sets $\mathcal{S}$

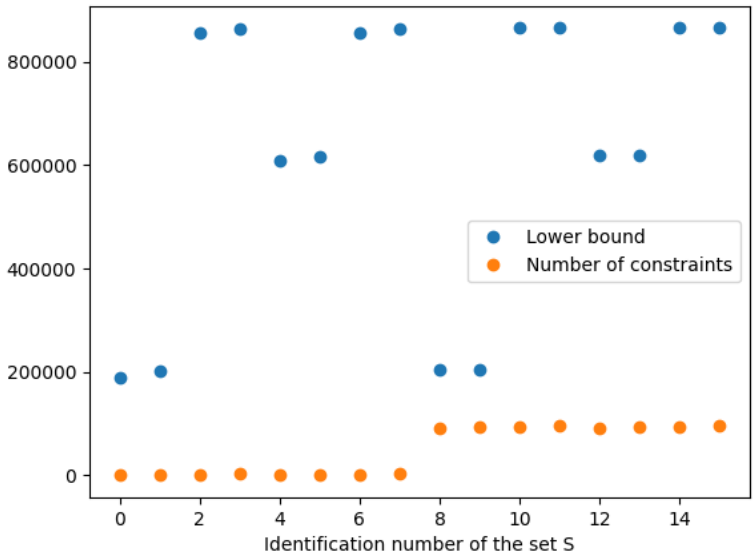


Figure A.8.: Instance 2: Values of the lower bound and number of constraints of the ranking method over different sets $\mathcal{S}$ including only the subsets 1, 2, 3 and 4

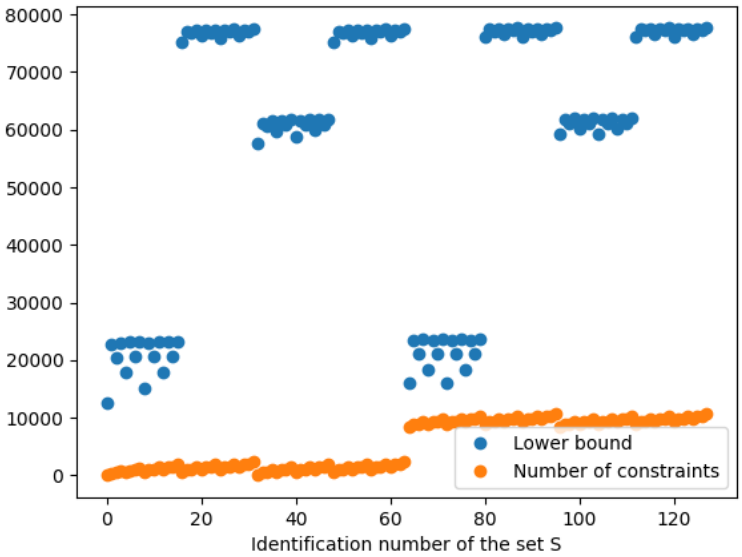


Figure A.9.: Instance 3: Values of the lower bound and number of constraints of the ranking method over different sets $\mathcal{S}$

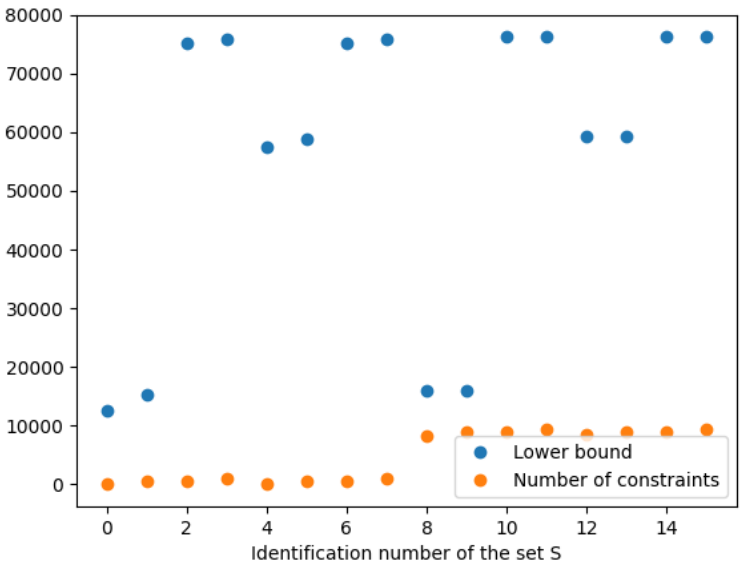


Figure A.10.: Instance 3: Values of the lower bound and number of constraints of the ranking method over different sets $\mathcal{S}$ including only the subsets 1, 2, 3 and 4

A.2.2. About the objective

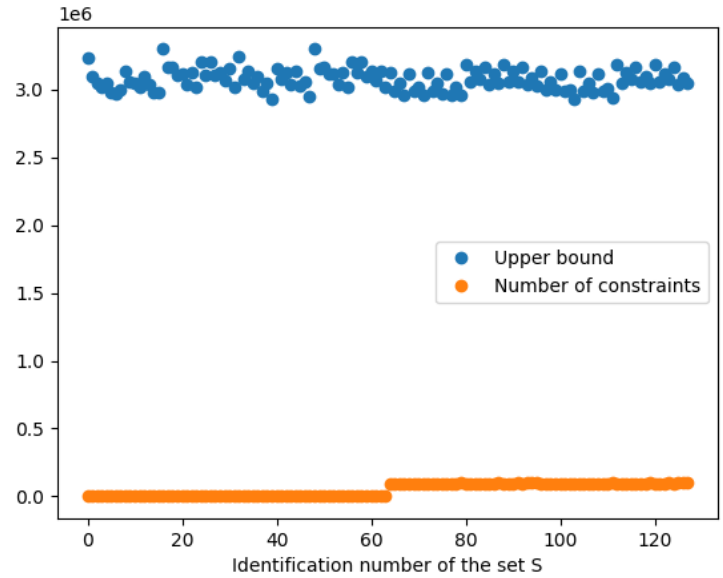


Figure A.11.: Instance 2: Values of the objective and number of constraints of the ranking method over different sets $\mathcal{S}$

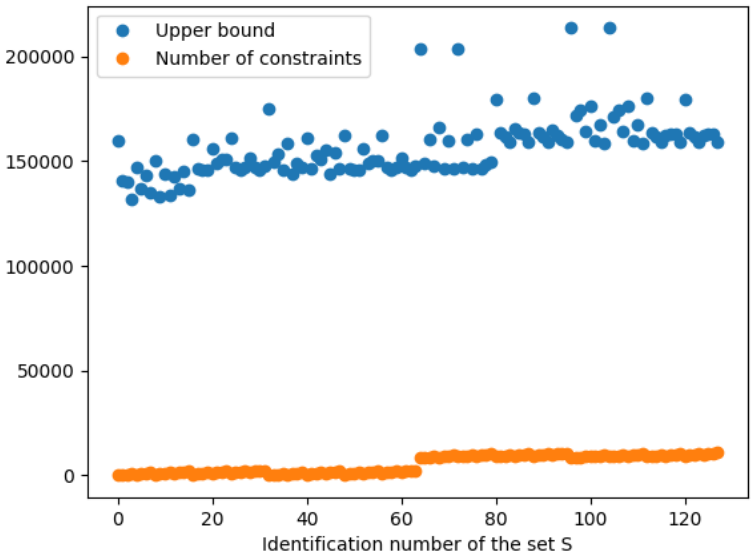
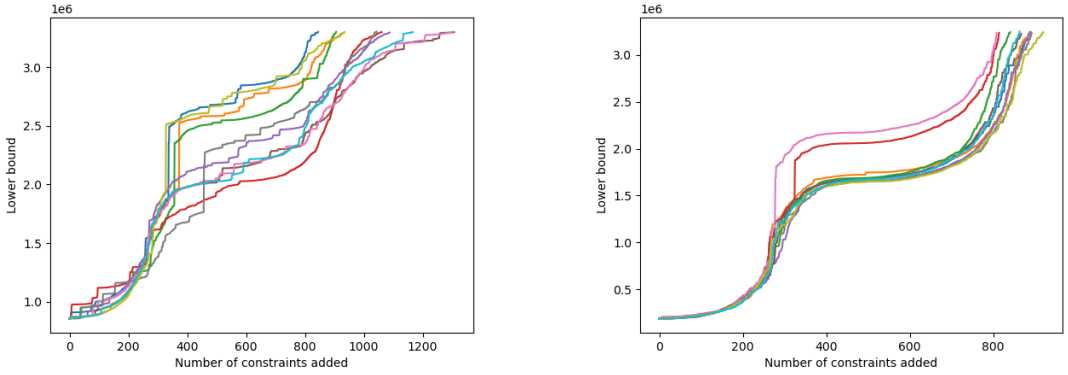


Figure A.12.: Instance 3: Values of the objective and number of constraints of the ranking method over different sets $\mathcal{S}$

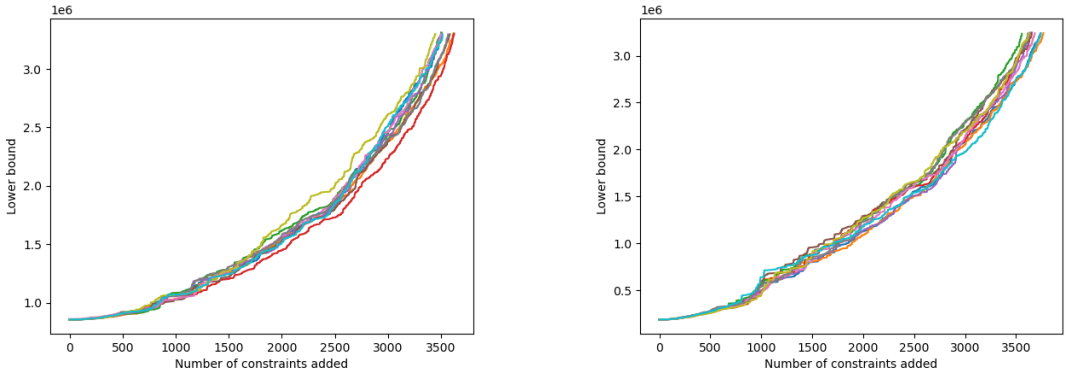
A.3. Branch and Bound Algorithm

A.3.1. Selection of the parameters to improve the pruning

Choice of the constraint strategy

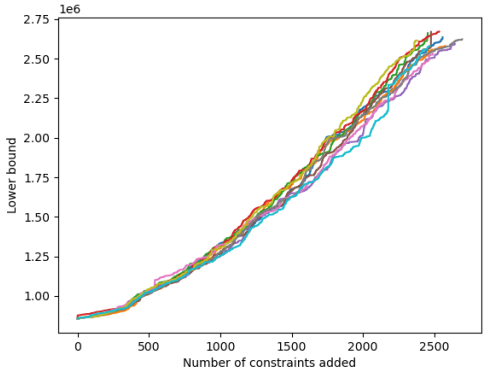


(a) CM-FO - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$ (b) CM-FO - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

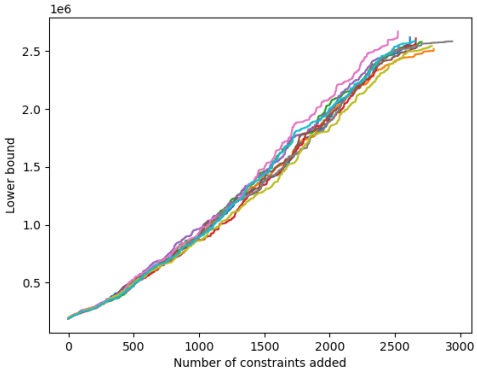


(c) LJ - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$ (d) LJ - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

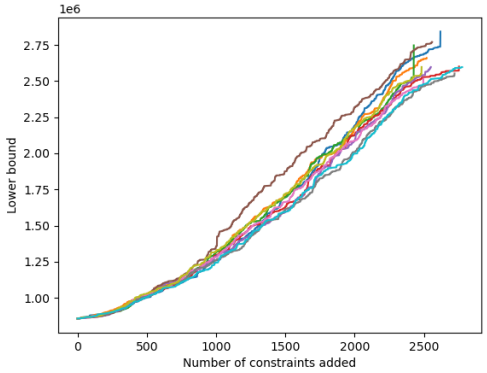
Figure A.13.: Instance 2: Evolution of the lower bound for 10 dives with the constraint strategies CM-FO and LJ and two sets $\mathcal{S}$



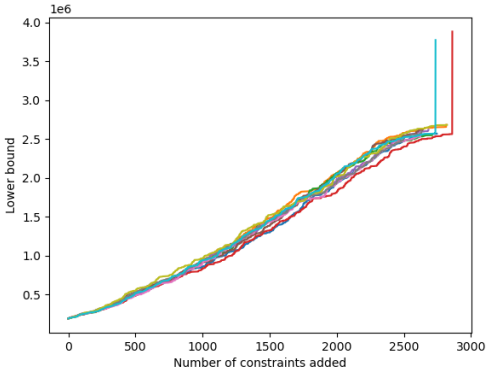
(a) CM-LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(b) CM-LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

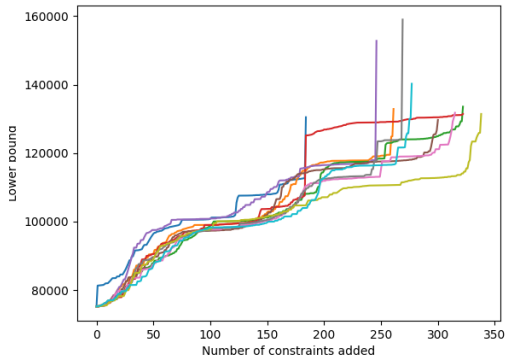


(c) CM-5LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

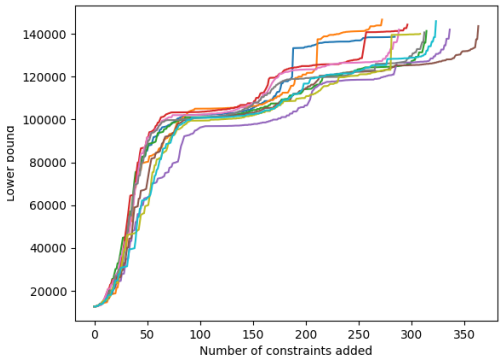


(d) CM-5LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

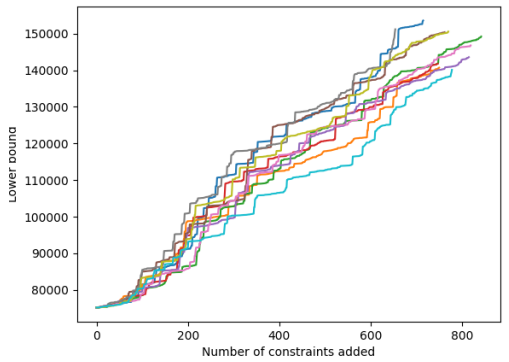
Figure A.14.: Instance 2: Evolution of the lower bound for 10 dives with the constraint strategies CM-LV and CM-5LV and two sets $\mathcal{S}$



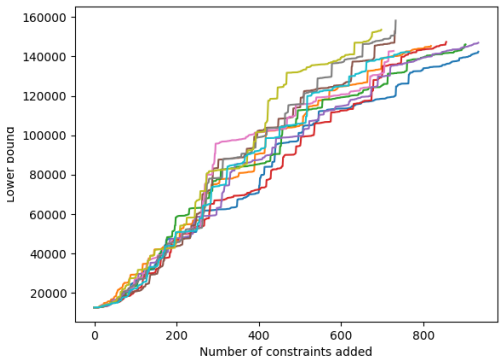
(a) CM-FO - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(b) CM-FO - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

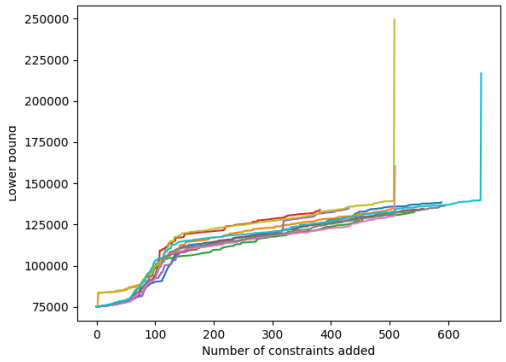


(c) LJ - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

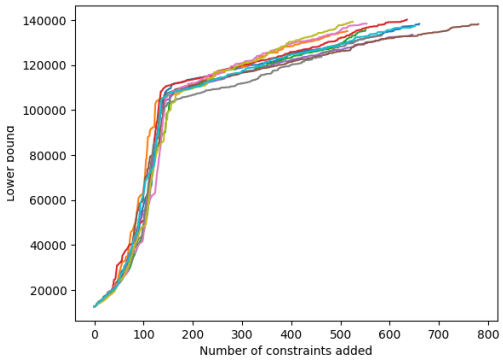


(d) LJ - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

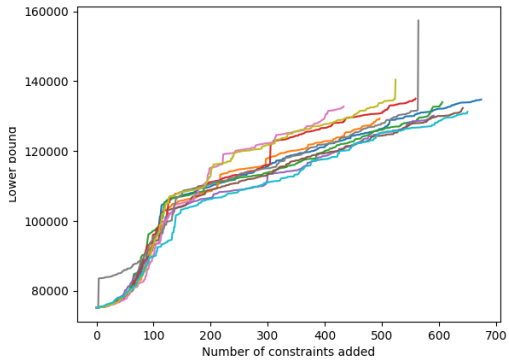
Figure A.15.: Instance 3: Evolution of the lower bound for 10 dives with the constraint strategies CM-FO and LJ and two sets $\mathcal{S}$



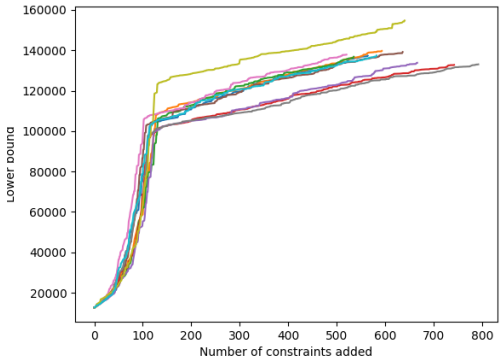
(a) CM-LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(b) CM-LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$



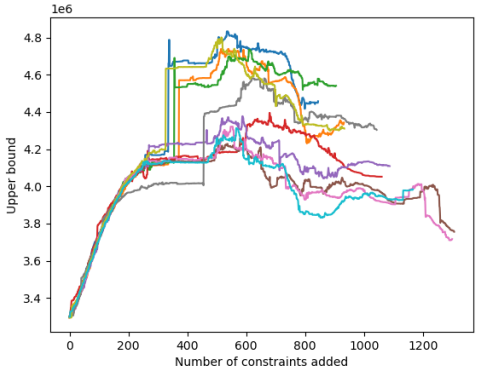
(c) CM-5LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



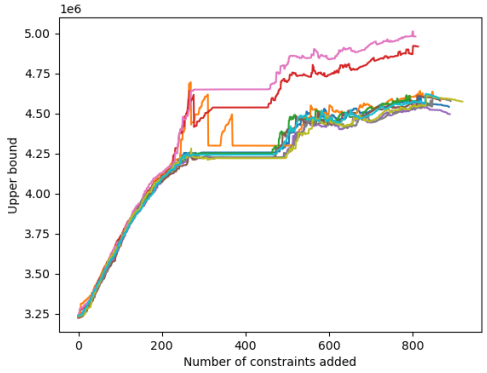
(d) CM-5LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

Figure A.16.: Instance 3: Evolution of the lower bound for 10 dives with the constraint strategies CM-LV and CM-5LV and two sets $\mathcal{S}$

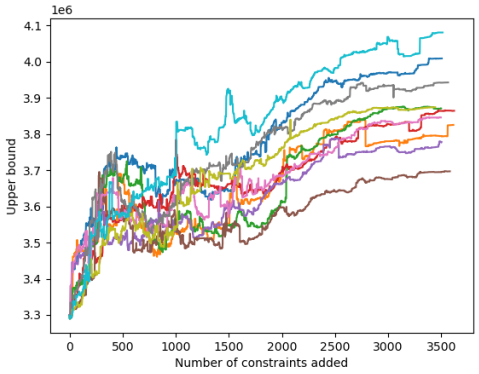
A.3.2. Results of the branch and bound method



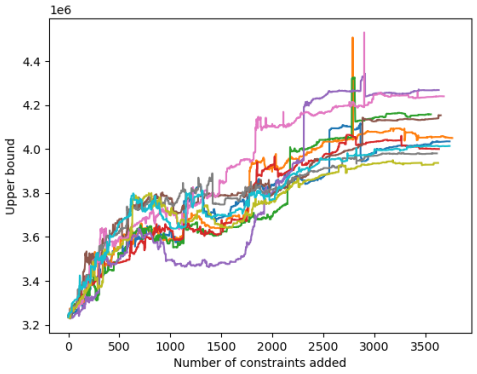
(a) CM-FO - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(b) CM-FO - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

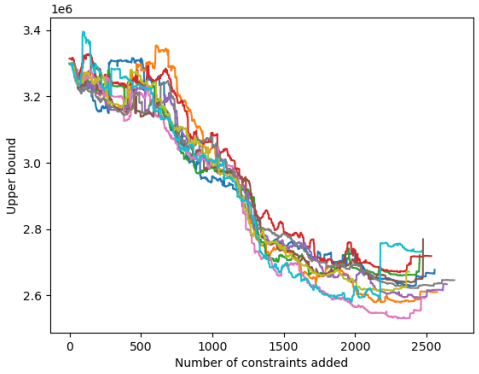


(c) LJ - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

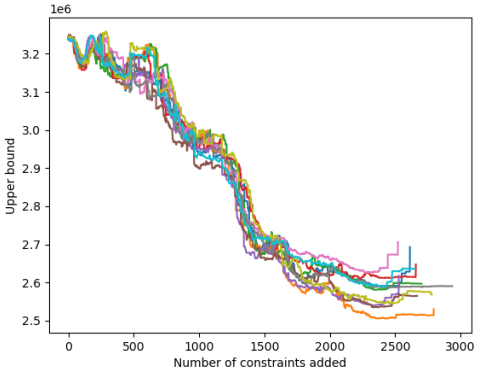


(d) LJ - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

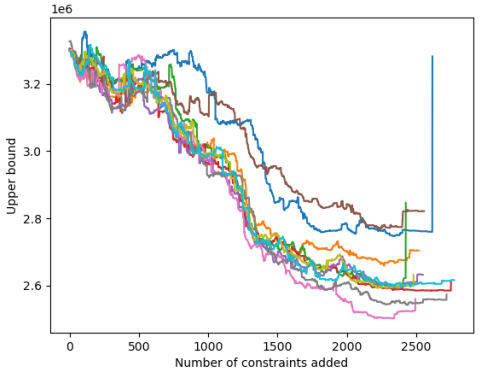
Figure A.17.: Instance 2: Evolution of the upper bound for 10 dives with the constraint strategies CM-FO and LJ and two sets $\mathcal{S}$



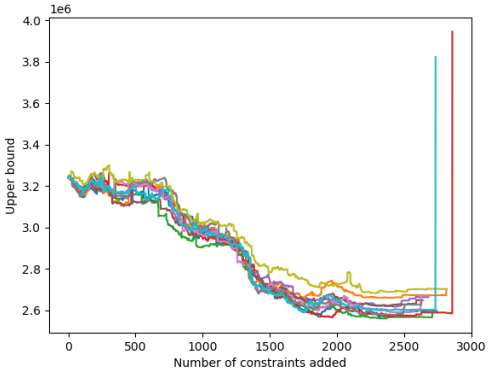
(a) CM-LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(b) CM-LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

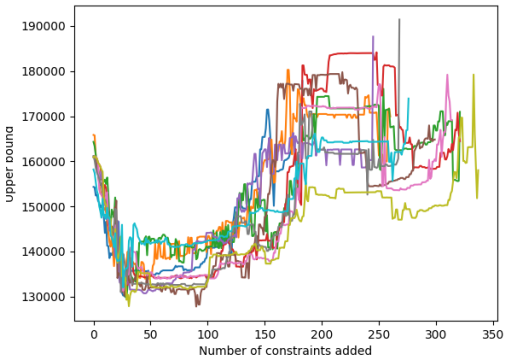


(c) CM-5LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

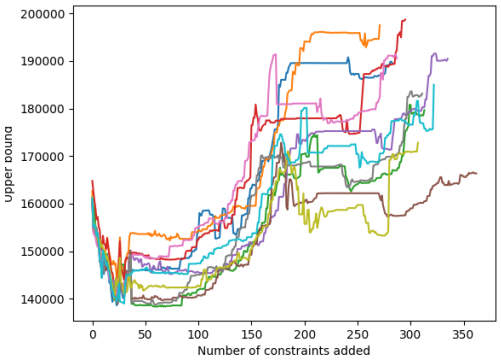


(d) CM-5LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

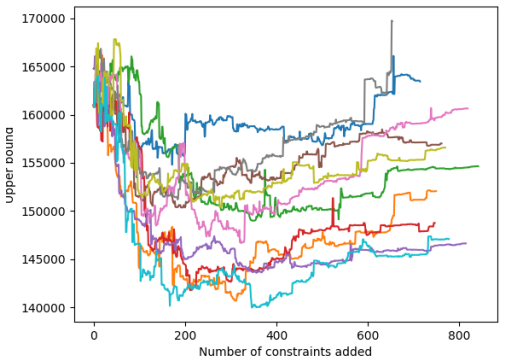
Figure A.18.: Instance 2: Evolution of the upper bound for 10 dives with the constraint strategies CM-LV and CM-5LV and two sets $\mathcal{S}$



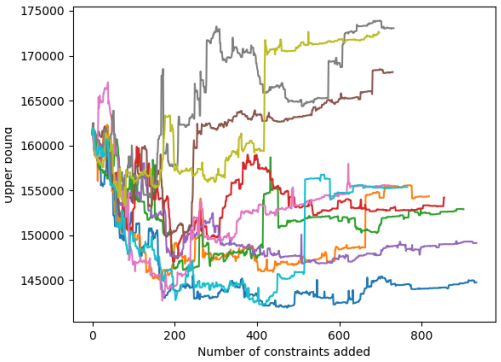
(a) CM-FO - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(b) CM-FO - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

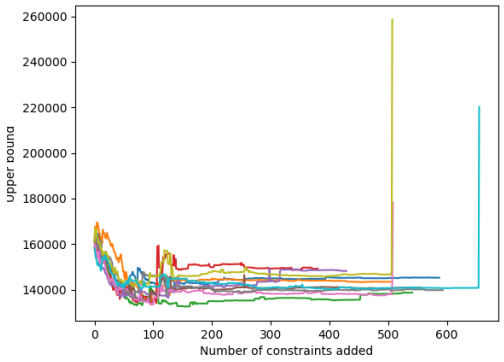


(c) LJ - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

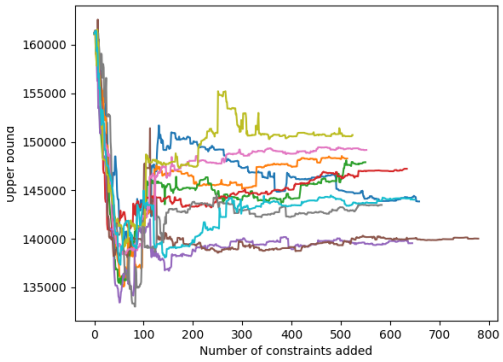


(d) LJ - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

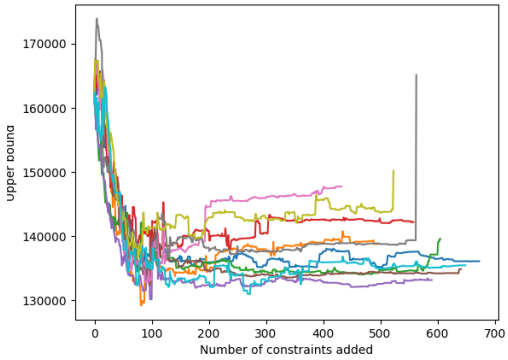
Figure A.19.: Instance 3: Evolution of the upper bound for 10 dives with the constraint strategies CM-FO and LJ and two sets $\mathcal{S}$



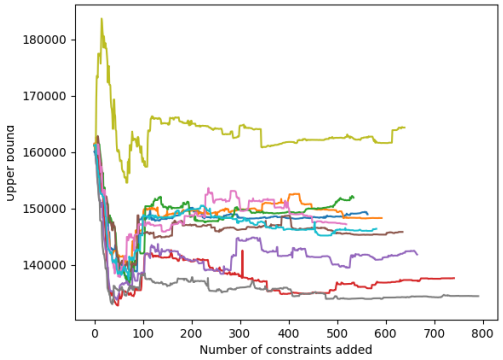
(a) CM-LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(b) CM-LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$



(c) CM-5LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

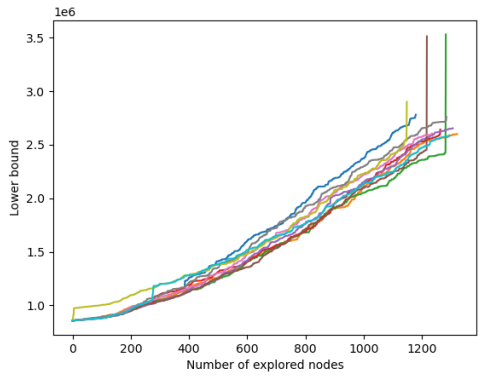


(d) CM-5LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

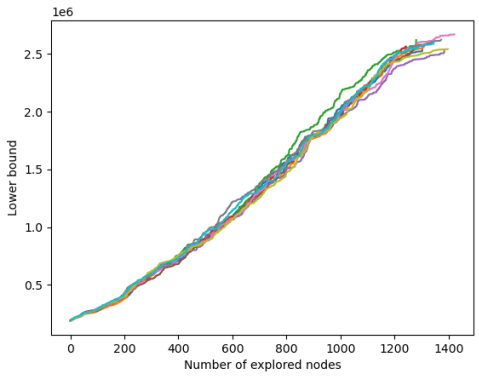
Figure A.20.: Instance 3: Evolution of the upper bound for 10 dives with the constraint strategies CM-LV and CM-5LV and two sets $\mathcal{S}$

A.4. Dive

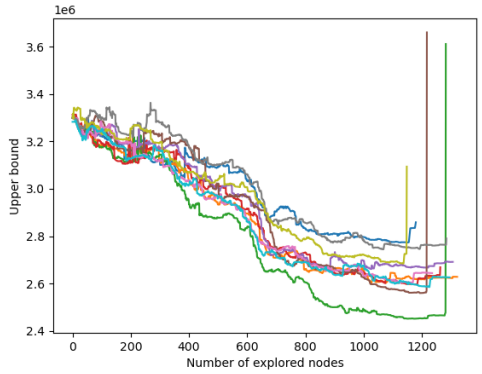
A.4.1. Selection of the constraint strategies



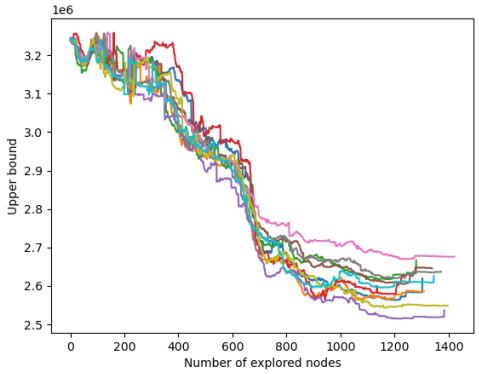
(a) CM-2LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(b) CM-2LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

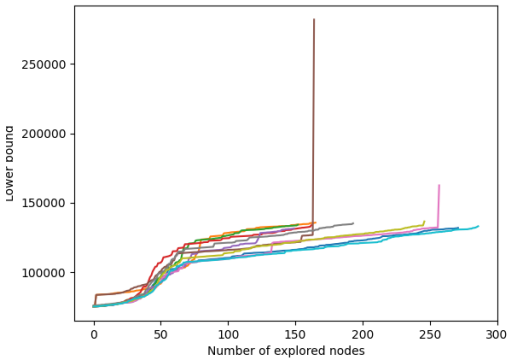


(c) CM-2LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$

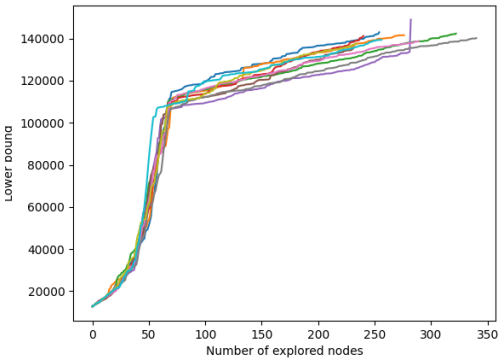


(d) CM-2LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

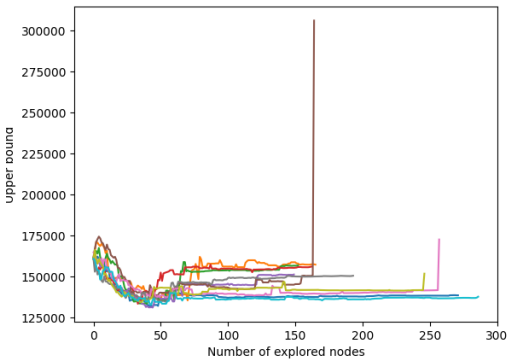
Figure A.21.: Instance 2: Evolution of the lower bound and the upper bound for 10 dives for the constraint strategy CM-2LV and two sets $\mathcal{S}$



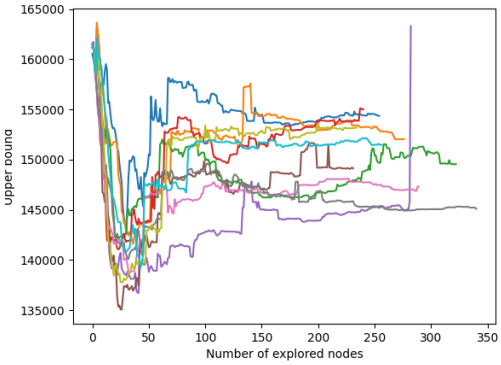
(a) CM-2LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(b) CM-2LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$



(c) CM-2LV - $\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$



(d) CM-2LV - $\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$

Figure A.22.: Instance 3: Evolution of the lower bound and the upper bound for 10 dives for the constraint strategy CM-2LV and two sets $\mathcal{S}$

Constraint strategy	Time in minutes	
	$\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$	$\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$
CM-FO	5:37	1:47
LJ	16:16	12:05
CM-LV	9:19	6:21
CM-5LV	11:59	9:08
CM-2LV	5:48	4:13

Table A.16.: Instance 2: Time to perform 10 dives with different constraint strategies with 2 different sets $\mathcal{S}$

Constraint strategy	Time in minutes	
	$\mathcal{S} = [0, 0, 1, 0, 0, 0, 0]$	$\mathcal{S} = [0, 0, 0, 0, 0, 0, 0]$
CM-FO	0:28	0:22
LJ	1:17	1:32
CM-LV	0:48	0:49
CM-5LV	1:20	1:12
CM-2LV	0:33	0:31

Table A.17.: Instance 3: Time to perform 10 dives with different constraint strategies with 2 different sets $\mathcal{S}$

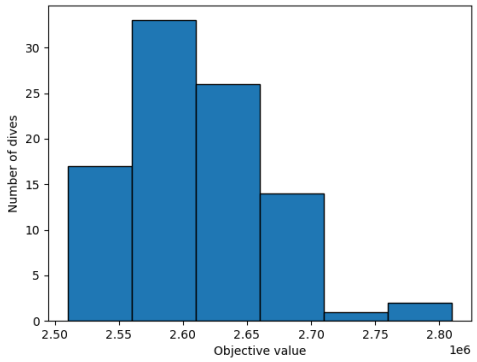
A.4.2. Results

Constraint strategy	Upper bound	Lower bound	Number of constraints added	Number of dives
CM-LV	2,468,899	2,309,829	2,265	140
CM-5LV	2,490,272	2,374,428	2,295	109
CM-2LV	2,461,249	2,282,403	2329	226
CM-FO	3,271,120	888,374	2	194

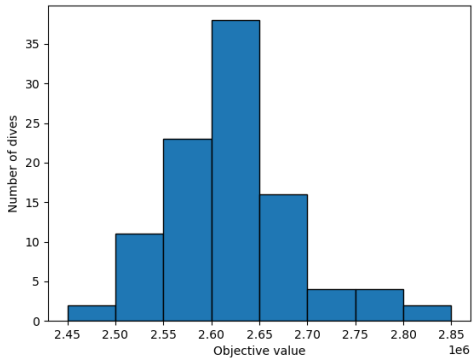
Table A.18.: instance 2: Comparison of Upper Bounds, Lower Bounds, Constraints Added, and Number of Dives for Different Constraint Strategies within a 2-Hour Dive Approach Run

Constraint strategy	Upper bound	Lower bound	Number of constraints added	Number of dives
CM-LV	122,988	92,794	107	2544
CM-5LV	123,032	107,199	352	1625
CM-2LV	123,413	115,799	485	3653
CM-FO	123,783	96,590	91	3014

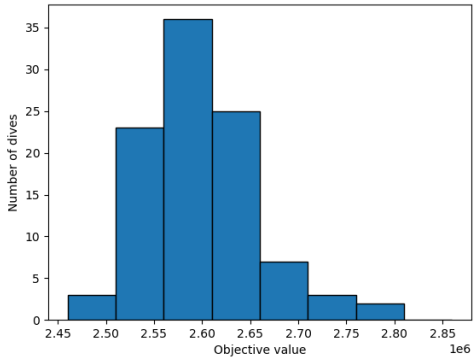
Table A.19.: instance 3: Comparison of Upper Bounds, Lower Bounds, Constraints Added, and Number of Dives for Different Constraint Strategies within a 2-Hour Dive Approach Run



(a) CM-LV



(b) CM-5LV

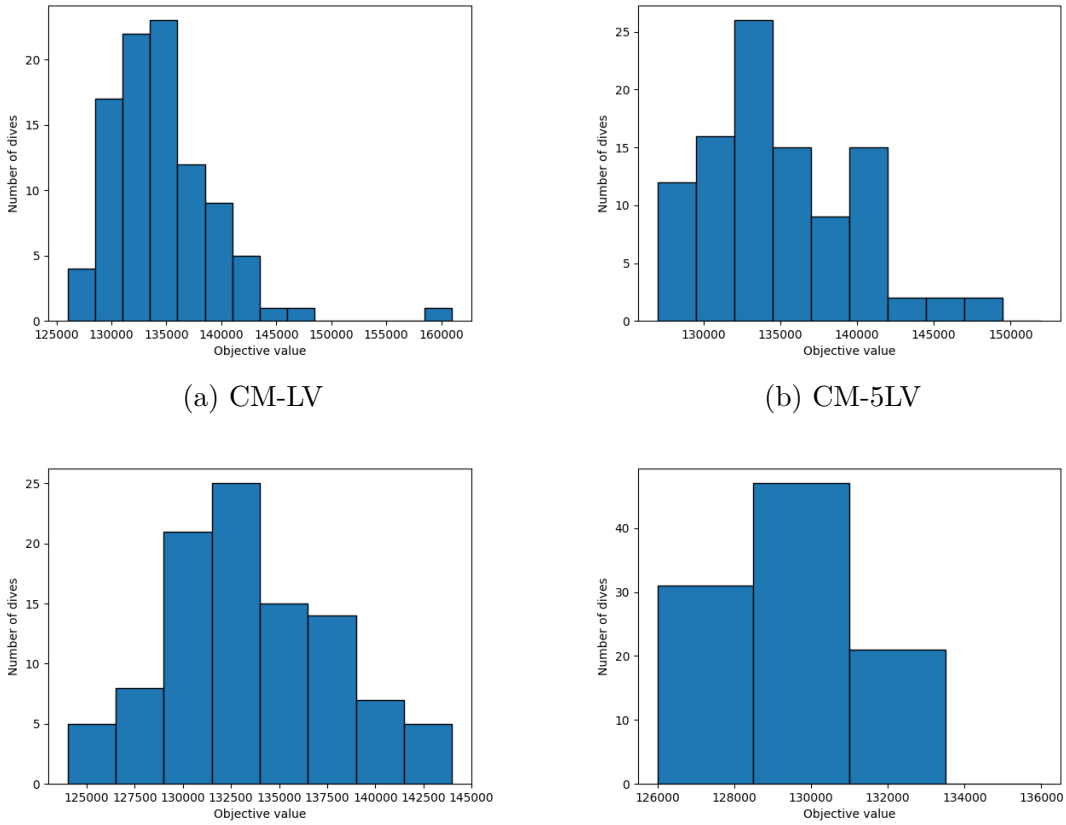


(c) CM-2LV

Figure A.23.: Instance 2: Distribution of best objective values from 100 dives for constraint strategies CM-LV, CM-5LV, and CM-2LV

Constraint strategy	Time (in hours)
CM-LV	1:35:06
CM-5LV	1:58:02
CM-2LV	0:57:39

Table A.20.: Instance 2: Time to perform 100 dives with constraint strategies CM-LV, CM-5LV, and CM-2LV



(a) CM-LV

(b) CM-5LV

(c) CM-2LV

(d) CM-FO

Figure A.24.: Instance 3: Distribution of best objective values from 100 dives for constraint strategies CM-LV, CM-5LV, CM-2LV and CM-FO

Constraint strategy	Time (in minutes)
CM-LV	9:26
CM-5LV	12:48
CM-2LV	5:05
CM-FO	4:45

Table A.21.: Instance 3: Time to perform 100 dives with constraint strategies CM-LV, CM-5LV, CM-2LV and CM-FO