ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

Business Analytics and Data Science
Πρόγραμμα Μεταπτυχιακών Σπουδών στην
ΑΝΑΛΥΤΙΚΗ ΤΩΝ ΕΠΙΧΕΙΡΗΣΕΩΝ ΚΑΙ ΕΠΙΣΤΗΜΗ ΤΩΝ ΔΕΔΟΜΕΝΩΝ
Τμήμα Οργάνωσης και Διοίκησης Επιχειρήσεων

Πρόγραμμα Μεταπτυχιακών Σπουδών
στην Αναλυτική των Επιχειρήσεων και Επιστήμη των Δεδομένων
Τμήμα Οργάνωσης και Διοίκησης Επιχειρήσεων

Διπλωματική Εργασία

Machine Learning for Scheduling Problems

του

Γεώργιου Υφαντίδη του Ελευθερίου

Υποβλήθηκε ως απαιτούμενο για την απόκτηση του μεταπτυχιακού διπλώματος στην Αναλυτική των Επιχειρήσεων και Επιστήμη των Δεδομένων

Φεβρουάριος 2023

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα κατ’ αρχάς να ευχαριστήσω τον καθηγητή μου Κωνσταντίνο Καπάρη για την αμέριστη συμπαράσταση στην πραγμάτωση αυτής της διπλωματικής. Σε συνέχεια, θα ήθελα να ευχαριστήσω το οικογενειακό μου περιβάλλον για τη βοήθεια και την υποστήριξη που επίσης μου προσέφερε.

Abstract

In the modern competitive world, it is crucial for manufacturers to be able to quickly and accurately predict the best possible manufacturing schedule for their machines. In this thesis we try to create a machine learning algorithm that is capable of creating a near optimal schedule, based on prior knowledge i.e. previously obtained optimal solutions, to quickly predict the daily schedule of a flow shop. As described, solving an integer programming problem to obtain the optimal schedule is not viable. Though other techniques like heuristics exist, we propose a different way, not only to obtain near optimum schedules but also to provide lower and upper bounds to new problems so that when a new problem arises it will be easier for mixed integer programs to solve them. Heuristics have the disadvantage that it is not clear how close the solution that is proposed is to the optimal one. With machine learning it is clear the linkage between the schedule that is proposed and the optimum one due to the metrics that are in place.

Table of Contents

Contents

1. Brief Introduction

In this thesis, first we explore the ways in which scheduling problems are solved. Then we introduce the types of scheduling problems there are and the objective functions that are used to obtain the optimal solution that is needed. In the following section, we introduce the machine learning ecosystem. In the last part we apply a machine learning algorithm for classification, called XGboost, that solves instances of flow shop problems found in (Taillard, 1993). Our goal is to obtain a schedule that is produced ad hoc and is accessible to the scheduler – manufacturing supervisor to plan the daily or weekly schedule. In real life implementation, more attributes will be added, more parameters will be accessible, and experts' opinion will be available, so the prediction capabilities of the algorithm can be more precise.

2. Existing Typical Methodologies

2.1. Mixed Integer Problems

MIPs (Mixed Integer Problems) are a subset of problems regarding Linear Problems. We will provide a general guideline for integer problems and the optimization process in this chapter.

Integer programming is an area of optimization that is thriving and is applied nowadays to a variety of problems due to the evolution of software and hardware. The origin is dated back to 1957, when Ralph Gomory announced that he would lecture on solving linear programs in integers. At that time, he was occupying the position of Higgins Lecturer of Mathematics at Princeton. He worked as a scientist at IBM from 1959 until 1970 (when he took over as Director for Research). Gomory developed approaches like cutting planes and the polyhedral combinatorics of corner polyhedra to group knapsack relaxations, that remain striking to researchers even today (Chandru & Rao, 1998). The main solving techniques that are used for finding the exact solution of an IP (Integer Problem) are Cutting Planes, Branch and Bound, Branch and Cut (which is a combination of the above). We will discuss these three methods in depth in this chapter. Then, we will discuss heuristic methods for finding a near optimal solution, and lastly, we will discuss constraint programming.

2.1.1. Cutting Planes

Cutting planes is a way of solving IPs that was initially developed, as mentioned above, by Ralph Gomory. The idea behind this method is based on polyhedral combinatorics. To solve the constraint set of an integer programming problem, we replace them with polyhedral combinatorics through an alternative convexification of the feasible points and extreme rays of the problem. Both the size and the complexity of the problems solved have been increased considerable when polyhedral theory was applied to numerical problem solving. (Weyl, 1950) proved that a convex polyhedron can alternatively be defined as the intersection of a finite number of half spaces or as the convex hull plus the canonical hull of some finite number of vectors or points. Based on this theoretical result, Gomory derived a 'Cutting Plane' algorithm for integer programming problems, which can be viewed as a constructive proof of Weyl's theorem. The general cutting plane approach initially relaxes the integrality restrictions on the variables and solves the resulting linear program over the constraint system. In the same way that the linear program is unbounded or infeasible, the same is valid for the integer program. In the case where the solution to the linear program is an integer, this is the optimal solution to the integer program. When there is no optimal integer solution for the linear program, a facet-identification problem must be solved. Here the objective is to find a linear inequality that cuts off the fractional linear programming solution while assuring that all feasible integer points satisfy the inequality. The conditions to terminate the algorithm are the following:

a) An integer solution is found.

b) The linear program is infeasible, and consequently so is the integer,

c) No cut is identified by the facet-identification procedures either because a full description of the facial structure is not known or because the procedures are inexact, so there is no possibility for algorithmically generating cuts of a known form. (Genova & Guliashki, 2011)

Cutting planes for integer programs can be classified into two categories. The first is generic and does not require knowledge of the underlying constraint matrix, while the other does. First, we will present the Cutting Planes implementation that does not require knowledge of the structure and then the one that does. (Jünger & Naddef, 2001)

2.1.1.1.Cutting Planes independent of problem structure

Some examples of cutting planes in pure integer programs that do not require knowledge of the structure of the constraint matrix are mixed integer Gomory cuts, mixed integer rounding cuts, lift and project cuts. Gomory's mixed integer cuts approach fails when both continuous and integer values are present in a problem due to the inconsistency of the nature of the algorithm. Chvátal, in a later work, concluded that there was a distinct but close way of discovering the linear description of the convex hall, but his method also fails when both continuous and integer values are present. So, for the Mixed Integer Programs, we can rely on the mixed integer rounding and the lift and project cuts. (Jünger & Naddef, 2001)

2.1.1.2.Cutting Planes exploiting structure

The main idea behind this method is to find out the substructure of an inequity and rely on the polyhedral knowledge of the substructure to make the original formulation more robust. The main task is to find some substructure with a familiar polyhedral structure and discover violated cutting planes for the given substructure. The first implementation was made in 1983 by Crowder, Johnson, and Padberg regarding 0/1 integer programs, and each row in the constraint matrix was though of as a knapsack problem. It is still in use by many Mixed Integer Program solvers. (Jünger & Naddef, 2001)

2.1.2. Branch and Bound

The Branch and Bound (B&B) algorithm is a fundamental and widely used method for finding exact solutions to NP-Hard optimization problems. The algorithm was first proposed by Land and Doig in 1960. The algorithm implicitly enumerates all possible solutions to the given problem, by storing all the partial solutions, called subproblems, in a tree-like structure. The nodes that are yet to be explored produce children by partitioning the solution space into smaller parts that can be solved recursively (branching) and by giving rules that are used to prune off regions of the search space that are profoundly suboptimal (bounding). Once the entire tree is searched, the best solution is returned. The above framework has some strategies and rules that must be taken into consideration to make it more efficient. The first strategy is the search strategy that is applied, which determines the order in which the subproblems will be explored. The second strategy is the branching strategy, which dictates the way that the solution space is divided to produce new subproblems in the tree. Last there is the pruning rule, which implies the rules that are taken into consideration for the prevention of exploring suboptimal regions of the tree. There has been significant effort to develop both problem-specific Branch and Bound algorithms, which take advantage of the strategies and rules above, and general-purpose ones. (Morrison, Jacobson, Sauppe & Sewell, 2016)

2.1.2.1. The search strategies are:

- Depth First search: In this search strategy, during the first phase, the algorithm investigates a random node until it reaches the final leaf. Then the algorithm goes to the next intersection and exhausts it as well, until it has searched all the possible search spaces. Due to the nature of the B&B algorithm, search space does not include suboptimal solutions. Some of the advantages of this method are that it can use low memory quantities as it can track, in some cases, only the branch from the root to the specific subproblem that it explores and not the entire tree, and then backtrack to the nearest unexplored intersection, and the ability to store lower bounds that will provide a more specific search space (Bound) for the following iterations. Some of the disadvantages are that this search is naïve and does not take into consideration the problem structure or the bounds, which results in the exploration of extensive regions that are not near optimal, and the other disadvantage is that it is unbalanced and random, which can result in big branches that are explored without reason.
- Breadth First search: In this strategy, the opposite of the above takes place. In the first phase, the algorithm examines all nodes near the root, then those near the explored root, until it has exhausted all possible solutions in the given depth. The advantage is that the algorithm can easily find a solution that is near the root and performs well on unbalanced trees. The disadvantage is that it uses a lot of memory as it cannot exploit the pruning rules and therefore is not usually used in B&B.
- Best-Bound search: In this strategy, which is also called minimum value search, the goal is that given sufficient memory storage, the algorithm stores the entire tree and uses a heuristic measure of the best function that computes a value for every one of the unexplored subproblems and selects as the next exploration problem the one that

minimizes that value. There are many choices regarding the measure of the best function. One of the most common is a lower bound on the value of the best solution in the subproblem. In the event that the lower bound is correlated with the subproblem objective value, this measure of the best will encourage the exploration of subproblems with better solutions. Although there is a chance, the lower bounds may not be a good proxy for the objective function's value. This is due to the relaxation of the IP that produces the LP, a small lower bound may just indicate that the LP can ‘cheat’ in ways that the IP cannot. There are ways of overcoming this problem, such as heuristic approaches to estimate the quality of the lower bounds, and CPLEX, which is a commercial solver, uses that approach to estimate the objective value of a particular subproblem. This algorithm has some significant advantages over the two previously described algorithms, as it is not tied to exhausting the whole depth-width of the trees but can find good solutions earlier than them and can also explore fewer subproblems than any other search strategy that has access to the same heuristic function and other information. However, given a large number of subproblems and depending on the tie-breaking rule, it may spend a significant amount of time in the middle regions of the exploration and never reach an optimal solution. To address this issue, some implementations employ heuristics such as diving heuristics, which aim a subproblem at a new incumbent solution and thus aid in pruning.

- Cyclic Best first search: Originally it was named "distributed best first search." This strategy can be described as a hybrid algorithm between depth-first and breadth-first algorithms. This algorithm divides the unexplored subproblems over a collection of heaps referred to as contours. When a new subproblem is identified, it is inserted into one of the heaps based on a set of rules. To explore the search space, this algorithm’s strategy repeatedly iterates through all the non-empty contours, selecting the best subproblem from each contour before moving to the next one. Following this procedure, the measure of the best function a local ranking assigns to each subproblem within each contour is used instead of the global ranking that would otherwise be assigned. This way, each contour is used to group subproblems that can be directly comparable. By cycling through each contour frequently, the search is more diversified and can aid in generating incumbent solutions. (Ibaraki, 1976)

2.1.2.2.The branching strategies are:

- Binary branching: This strategy focuses on subdividing a subproblem into two mutually exclusive, smaller subproblems. As an example, in the knapsack problem, which tries to find the maximum number of items that will fit inside a bag with a given capacity, this technique for a subproblem S selects an unassigned item and creates two branches, one indicating that the item is included and one indicating that the item is excluded from the knapsack. This is a simplified method, and more complex mechanisms exist such as the graph coloring branch-and-price solver developed by Mehrota and Trick, where branching is performed by contracting vertices of the graph or by adding edges to force a pair of non-adjacent vertices to either use distinct colors or the same one.
- Wide branching: This strategy is the opposite of the one described above. Wide branching focuses on selecting one element from a set of different options. This strategy allows for potentially large reductions in the size of the search trees. The wide branching technique has two potential downsides. The first is that it usually does not create mutually exclusive branches, and it is possible to arrive at the same subproblem through many different paths. The second problem can occur if the number of branches that can materialize from a specific problem is very large. If this problem is at stake, there is a chance that the algorithm will spend a lot of time on a specific problem and never explore the other subproblems.
- Branching in integer programming: Many optimization problems can be modeled using integer programming; thus, much effort has been devoted to developing branching strategies for these kinds of problems. In these problems, the strategy is divided into two phases. The first one is to select a variable or a set of variables to branch on and then create subproblems by creating bounds on the selected variables to make them move away from the fractional values. The variable to be chosen on which we decide to force the branch can have a significant impact on the performance of the algorithm, and there are plenty of techniques to choose the suitable variable on

which to branch. The first technique, which is the most common and easiest to implement, is the rule called “most fractional” or “most infeasible,” where the variable yi is selected whose fractional part is closest to 0.5. This branching technique is no better in general than selecting a random branching variable in terms of computational time and the subproblems that are explored. The opposite rule, selecting the “least fractional,” is to select the yi that is closer to 0 or 1 than 0.5, is also used, though less frequently, and is also outperformed by other rules. The second technique, “pseudo cost branching” tries to predict the change that each unit will have in the objective function for every yi based on the knowledge that is acquired in the tree. By advancing the subproblem by a variable that is supposed to give a notable change in the objective function, the more likely it is that the problems generated will be pruned. The main difficulty is that there is no prior knowledge about the branching behavior of the variables at the start of the algorithm, so the algorithm must be initialized in some way, and one strategy to avoid this is to set the pseudo-cost of each variable to the value of the coefficient in the objective function, as if these values would correspond to the actual pseudo-cost given that all variables are independent. The third technique is “strong branching,” in which the variable to be branched is the one that inflicts the biggest change in the objective function. During this technique, the algorithm computes the LP relaxation objective value of the descendant of a subproblem S for a fraction of the candidate branching variables, and then the variable that impacts the biggest change in the objective function is selected. There is a variation of this technique called “full strong branching,” which considers all the variables that are candidates for branching before selecting one. Also, there is a technique that combines “pseudo cost branching” and “strong branching” so that on the upper levels of the tree search it uses “strong branching” and in the lower levels it uses “pseudo cost branching.” The fourth is described in a paper by (Pryor & Chinneck, 2011) and uses branching rules that try to find feasible integer solutions that solve the problem quickly. This is achieved by selecting variables to branch on that impact the largest number of variables in the integer problem, which is different from the other techniques. The fifth technique is called “backdoor branching,” which solves an auxiliary integer program that determines a subset of the variables that should be selected before the others. Finally, there is a method that removes uncertainty from the subproblems in the tree. The subproblems that are high on the tree have large uncertainty since a small number of variables is fixed, and on the other hand, at the bottom of the tree, the uncertainty is zero since all variables are integers. In this method, the values of the fractional variables are treated as probabilities, and entropy is computed for the variables, and the variable with the least entropy is selected to be branched on. (Morrison et al., 2016)

2.1.2.3.Dominance and Pruning rules

One of the most significant aspects of Branch and Bound is choosing the pruning rules that help us reduce the space in which we search by excluding them from the search space. There are a plethora of families of rules that prune the derived trees, but they are usually specific for each problem and must be found anew for each one at hand. Due to the nature of Integer Programming many of the pruning rules are focused on these types of problems. There are five families of pruning and dominance rules, and we will present them briefly below. (Morrison et al., 2016)

2.1.2.3.1. Lower Bounds

One of the most common ways to achieve the pruning of the tree is by producing a lower (or upper) bound that is based on the objective function and implementing it in every subproblem. These bounds are found by using relaxations in various aspects of the integer problem. In the case that a problem can be formulated as an integer program, the solution of the linear relaxation problem is a common lower bound to be chosen. Another method is to use duality, in which the optimal solution value of the Lagrangian relaxations is bound above by the solution to the non-dual problem. (Morrison et al., 2016)

2.1.2.3.2. Dominance Relations

In contrast to the rules described above, dominance relations allow subproblems to be pruned if they are dominated by another subproblem. Given that a subproblem (S1) is dominated by another (S2), every solution provided by the second (S2) is at least better than the solutions given by the first (S1), so it is sufficient to search only the first (S1). There are two types that describe dominance rules. The first one is memory-based, which means that unexplored subproblems are compared to subproblems that were previously explored and stored in the memory. This process requires the entire tree to be stored during the execution of the algorithm, and this way it is possible that additional pruning is performed that would be impossible otherwise. The second one is non-memory-based rules, which do not require an a priori dominant state during the search but instead can imply the existence of a subproblem that dominates the others without the implicit exploration or generation of that problem. This type of dominance rule has the advantage of not requiring extra memory to be allocated to the generated search tree, but it may be unable to prune as many subproblems as memory-based ones. (Morrison et al., 2016)

2.1.3. Branch and Cut

Branch and Cut is another method for solving integer linear programs. As implied by the name, this method is a combination of the two mentioned above. This method involves running a branch and bound algorithm and then using cutting planes to make the linear programming relaxations tighter. (Mitchell, 2002)

2.1.4. Computational Complexity

Computational complexity is the study of the resources that are required to solve a problem. These resources are usually time and space (memory) (Loui, 1996). The theory tries to make distinctions by proposing a formal criterion to determine what it takes for a problem to be characterized as feasibly decidable – a problem that can be solved in a number of steps that is proportional to the amount needed for a polynomial function to be solved by a Turing machine regarding its size. If a problem, can be solved in polynomial time then it has the property of P. If a problem needs exponential time, then it is EXP. Lastly, a problem that cannot be solved in a non- deterministic polynomial time is called NP (Walter, 2021). There is a prize of one million dollars towards proving that P is not NP by the Clay Mathematical Institute (Jaffe, 2006). There is a way to solve new problems which is to reduce the given problem into previously solved ones. This method is called NP-hardness and frequently instances of new problems are formulated entirely from previous problems. Lastly, NP-hardness describes a problem that the entire NP class problems reduce, polynomially, to P. (Pinedo, 2016)

Scheduling problems depending on the characteristics of the specific problem, such as if the problem is single machine, parallel machines or shops or even if there are batches, if permutation is allowed, the objective function and other criteria can be characterized as P, NP-Hard or Strongly NP-Hard. In (Pinedo, 2016) we can find a categorization of scheduling problems depending on the computational complexity of the problem.

2.1.5. Heuristics and Meta-Heuristics

Heuristics have played a key role in operational research for a lengthy period. They were described in the earliest operational research textbooks before the 1960s, but not explicitly as heuristics. It is usually understood as an iterative algorithm that converges to a solution that is neither the optimal nor the only feasible. There are various applications for heuristics. The four that are most important are:

- Problems with no efficient way of finding the optimal solution, implying that the solution cannot be found within acceptable time constraints and heuristics are the only option.
- Problems where there is an optimal solution, and heuristics are a way to collaborate to make the process faster.

- Heuristics can be used to look ahead in tree search techniques such as Branch and Bound.
- Converging algorithms have heuristic elements, such as the pivot selection rules.

Most of the problems that need to be treated with a heuristic approach are of the combinatorial type because those problems usually have no efficient converging algorithms. Until the seventies, there was hope that efficient algorithms for combinatorial problems would be developed, and thus the mathematicians that were occupied in the operational research field did not hold heuristics in high regard. This began to change in the early 1970s, with the rise of the computational complexity field.

As mentioned above, combinatorial problems are those for which a heuristic approach is of highest importance. Combinatorial problems deal with one or more discrete sets of objects. Three pure types of these problems can be presented: sequencing problems, assignment problems, and selection problems. Most of the standard problems are optimization problems and seek for the optimal solution among the feasible ones, but there is a category of problems that seek only a feasible solution, such as the Timetable Problems. (Müller-Merbach, 1981)

2.1.5.1.Heuristics in the system of algorithms

Algorithms should be understood as "Procedures for solving a problem stated in mathematical terms." There are several types of algorithms, each with significant differences. Some of them are:

- Direct algorithms: Algorithms that do not work in an iterative way.
- Iterative algorithms: Algorithms that work in an iterative way. Most of the algorithms have sub-procedures that are repetitive.
    - Algorithms without proven convergence: Heuristics are a part of this category as they do not necessarily converge to the sought-after solution.

- Converging algorithms: Iterative algorithms that do converge to the sough solution.
    - Approximation algorithms: Algorithms that do not necessarily converge to the optimal solution but are approximations to it. These algorithms are not finite, and each iteration brings them closer to the optimal solution.
    - Finite algorithms: Algorithms that guarantee the optimal solution within a finite number of iterations.
        - Path structured algorithms: In such algorithms as simplex, one iteration is followed by another without the existence of branching capabilities.
        - Tree structured algorithms: Such as Branch and Bound, where the iterations form a tree which consists of parallel branches. (Müller-Merbach, 1981)

2.1.5.2.Meta-Heuristics

A metaheuristic is an algorithmic framework that is high-level, problem-independent, and provides a set of rules, guidelines, and/or strategies to develop heuristic optimization algorithms. Some examples of metaheuristic algorithms that we are going to analyze in this paper are genetic/evolutionary, simulated annealing, and ant colony. A problem-specific implementation of a heuristic optimization algorithm is also a kind of metaheuristic. Metaheuristic algorithms are always heuristic by nature. This means that they differ from the exact methods mentioned above in that the optimal solution will be found in a finite amount of time (usually too large). On the other hand, metaheuristics are developed specifically for finding a solution that is acceptable (good enough) in a time frame that is short enough. Due to their heuristic nature, metaheuristics are not subject to combinatorial explosion.

The scientific community has demonstrated metaheuristics to be a viable alternative to exact methods such as branch and bound and dynamic programming. Notably, when a problem is complicated or there are many instances of it, metaheuristics are usually an acceptable tradeoff between computing time and solution quality. There are two major reasons that metaheuristics are more flexible than exact methods:

- Metaheuristics are defined in general terms and can be adapted to solve most real-life problems in terms of allowed computing time and solution quality, although they can vary among different problems and situations.

- Metaheuristics do not demand that the formulation of optimization problems follow the typical form, such as constraints or objective functions, or to be in the form of linear algebra functions of the variables.

Criticism of metaheuristics is based on the lack of universal application of the design of the methodology, the scientific ambiguity regarding the testing of different implementations, and the trend to create too complicated methods with many operators. (Sörensen & Glover, 2013)

2.1.5.3.Heuristics used in Combinatorial problems

2.1.5.3.1. Simulated Annealing

Simulated annealing is an algorithm that, even though it is a local search metaheuristic, has the capability of escaping from local optima. It is relatively easy to implement to take advantage of convergence properties and the use of hill-climbing to escape local optima, and that is the reason that simulated annealing has been a popular technique over the past two decades. Usually, it is used to solve discrete optimization problems and less often for continuous ones.

Simulated annealing is named after the analogy to the process that is used in the real-world process of physical annealing with solids, during which a crystalline solid is heated and then cooled very slowly until the most regular crystal lattice configuration is reached and the final product is as free as possible from crystal defects. Given a slow enough cooling rate, the final configuration is a solid with superior structural integrity. As described above, simulated annealing is the implementation of physical annealing, and it shows the connection between the search for global minima and thermodynamic behavior and provides an algorithmic way of exploiting this connection. (Gendreau & Potvin, 2010)

In every iteration, the algorithm compares two solutions, the current one and the new one. A solution that is better than the previous one is always accepted, as well as a fraction of the ones that are inferior, in the hope of escaping the local optima in search of the global. The decision whether an inferior solution is accepted is determined by a temperature parameter that is usually non increasing with each iteration of simulated annealing. The key feature of simulated annealing is that a mean to escape local optima is provided through hill climbing moves, which are moves that typically worsen the objective function solution. The temperature parameter that is described above is decreased in each iteration, and in this manner, the hill climbing is occurring less and less frequently, and the algorithm converges to a minimum that is either local or global. Simulated annealing can be modeled using theory based on Markov chains. A pseudocode is provided below with the steps involved in the algorithm:

Where:

S:	is the finite set of all solutions
f:	is a real valued function defined on members of S
T:	is the temperature

and the problem is to find a solution or state for $i \in S$ which minimizes f over S. (Eglese, 1990)

2.1.5.3.2. Tabu Search

Over the last two decades, many papers have presented applications of Tabu search, a heuristic method that was originally proposed by Glover in 1986, to many different combinatorial problems in the literature of operations research. In several cases, the implementation of the algorithm provides solutions that are close to the optimal one and among the best for addressing difficult problems. Through these successes, tabu search has become popular among those searching for good solutions to large combinatorial problems. Despite the abundance of literature, there seem to be many researchers who find it difficult to properly understand the concepts and limitations of tabu search and implement it effectively.

Tabu search is an extension of classical local search methods. A basic tabu search can be seen as a simple combination of local search and short-term memory. In other words, the two first basic components of tabu search are the definition of its search space and the structure of its neighborhood. The definition of the search space is the space of all possible solutions that can be visited during the execution of the algorithm. Likely, the structure of the neighborhood is the local transformations that can be searched during each iteration of the current solution.

Tabus are distinctive elements of tabu search in comparison to local search. Tabus are there to prevent cycling when the search is moving away from the local optima and choosing non-improving moves. The key point when cycling occurs is to declare tabu moves. These moves are prohibited moves that reverse the effect that recent moves have had. Tabus are also helpful when we want to search a wider area of the search space and to move away from the previous search area. Tabu rules are stored in the short memory, and only a small subset of information is recorded in the tabu list. There is no predefined procedure for the way the tabu list is created, but usually it stores the last few moves before the current solution and prohibits them from occurring again. Another strategy is to store only complete solutions, but that requires a lot of memory. Lastly, there are implementations that use key characteristics to choose what will be stored. While tabu lists can be immensely powerful, sometimes they are too powerful and prevent good moves even though cycling is not in sight. In this case, aspiration criteria can be implemented to delete some of the tabus and allow a move that enhances the solution of the objective function.

Even though a simple implementation of tabu search can, on occasions, provide successful solutions, some additional elements must be included to make the algorithm more effective. Some of them are:

- Intensification: Search a space more intensively if it seems more promising to ensure that the areas where the best solutions are found are being searched more thoroughly.
- Diversification: Make the algorithm search more search spaces and not focus only on a small subset of the search space. There are two major techniques. The first one is to force some rarely used components into the solution in hand and restart from this point, which is called restart diversification. The second is achieved by adding a small term to the objective function and thus biasing the possible moves; this one is called continuous diversification.
- Allowing infeasible solutions; If the constraints are too tight and the search space is too small to allow moves, the constraints are relaxed, and the search space is enlarged. This can lead to an infeasible solution, which is bypassed with the use of weights and self- adjusting penalties that increase if the local search is focused on infeasible solutions and decrease otherwise. (Gendreau & Potvin, 2010)

2.1.5.3.3. Genetic Algorithms

Genetic algorithm, a term that was first used by John Holland in 1975 and has evolved to be a flourishing field in terms of research and applications. The main idea for the development of genetic algorithms is the neo-Darwinian theory of evolution. (Gendreau & Potvin, 2010) The algorithm is analogous to biology for chromosome generation and variables as selection, mutation and crossover that make the genetic operations that would occur on a random population. The genetic algorithm aims to find solutions for the following generations. (Lambora, Gupta & Chopra, 2019)

Genetic algorithms are based on the concepts of heredity and natural selection. These algorithms are a subsection of a larger area of algorithms known as Evolutionary algorithms. Genetic algorithms are used to solve modeling systems with randomness, such as the stock market, and optimization problems such as scheduling and finding the shortest path. (Kumar, Husain, Upreti, & Gupta, 2010)

The methodology that is followed by the genetic algorithm is divided into four steps.

I. Initialization: Many random and individual solutions are generated as part of the initial population. The size is dependent on the nature of the problem, but usually consists of thousands of solutions. Even though typically the population is chosen at random with the aim of covering the whole search space, there are occasions where one can intervene and seed the initial solution in areas where it is known that optimal solutions can be reached.

II. Selection: In each successive generation, a part of the initial population is selected to breed. Individual solutions are determined through a process that is fitness-based, and the best solutions are most likely to be selected. Most of these fitness functions are stochastic and allow a small percentage of less-fit solutions to be selected, which helps with the diversity of the population. Other fitness functions rate only a random subsample of the initial population to shorten the process.

III. Reproduction: Now that the solutions are selected, the next generation is generated through mutation and/or crossover. For each successor, a pair of the selected solutions are chosen from the pool. The successor inherits many of its predecessor's characteristics through the processes of mutation and crossover. For each successor, a different pair of parents is selected, and this is continued until the appropriate size of the new population is reached. Due to biology-inspired reproduction methods, a pair of parents is used; however, research suggests that a larger count of parents yields better results. Through the repetition of these steps, fitness will increase due to the selection of the fittest in each iteration.

IV. Termination: The algorithm stops when one or more of the criteria is met. Some of the stopping criteria are:

a) The predefined number of generations has been reached.

b) A minimum criteria solution is found.

c) The time/computation resource limit has been reached.

d) Manual interruption (Kumar et al., 2010)

2.1.5.3.4. Ant colony optimization

Ant colony optimization is a metaheuristic algorithm that is used for solving combinatorial optimization problems. The principal that is used in the ant colony algorithm is the trail that real ants lay through pheromones, and it is used as a means of communication. Like real ants, this algorithm has its basis in the indirect communication within a colony of agents, which are called artificial ants, through artificial pheromone paths. (Gendreau & Potvin, 2010)

In real life, when ants are searching for food, they search the area in a random pattern. (Dorigo & Blum, 2005) When an ant finds a food source, it first evaluates the quality and quantity of the food and then carries a sample back to the nest. During this return trip, the ant lays phenomes depending on the quality and quantity of the food. The quantity of the phenome that is deposited is dependent on the food source and guides the other ants towards the food. This indirect communication helps the ants find the shortest path to the food and make the process as efficient as possible. This is the detailed procedure that is exploited in the ant colony optimization algorithm. (Gendreau & Potvin, 2010)

The algorithm was first introduced in the early 1990s by M. Dorigo and colleagues. The first implementation of ant colony optimization was named the "Ant System" and was applied to the traveling salesman problem. During this implementation, three algorithms were combined. The first was ant cycle, followed by ant density, and finally by ant quantity. Even though it could solve small problems of seventy-five cities heuristically, it was not competitive with other algorithms designed specifically for the traveling salesman problem. Because of this, a significant effort was made to enhance the algorithm with new features. Some of the new features that were added were:

- Elicit strategy: The best tour since the beginning of the algorithm is given extra weight when the pheromones are updated.
- Max-Min ant system: A lower and upper bound are introduced to the values of the pheromones that limit the range of the pheromones, which causes a wider search space for the algorithm.

• Ant colony system: Increases the importance of previous ants' information exploitation.

A brief algorithmic framework of the ant colony optimization algorithm is as follows:

• Initialize pheromone values: At the beginning of the algorithm, the pheromones are all initialized to a value of $c > 0$.

• Construct a solution: A set of ants finds a solution by extending the initially empty solution and adding a feasible solution to the currently partial solution until a complete solution is found.

• Apply local search: A local search may be applied to improve the solution that was found above. This step is optional but, in many experimental cases, it yields better overall performance.

• Apply pheromone update: The goal is to change the value of the pheromones so that the solution components that are being built and leading to a better solution are more likely to be chosen by the ants. There is a variety of algorithms that are applied to obtain the desired outcome in this step. (Dorigo & Blum, 2005)

2.1.6. Constraint Programming

"Constraint Programming is the study of computational systems based on constraints. The idea of constraint programming is to solve problems by stating constraints (requirements) about the problem area and, consequently, finding solution satisfying all the constraints". (Barták, 1999, p.1)

Constraint programming is a relatively novel approach to solving scheduling problems. Its origins are not in the Operations Research community but in the Computer Science community and specifically the Artificial Intelligence community in the 1970s. (Pinedo, 2016)

The approach that constraint programming uses to solve a problem is based on two branches. The first one is Constraint satisfaction which means that problems are defined over a finite domain, which includes a proportion of over 95% of real-life scheduling problems, and the second one is Constraint solving, where the problems are defined either on an infinite domain or a complex one. (Bockmayr & Hooker, 2005)

A constraint c (x1, x2, ..., xn) usually consists of a finite number of variables xj and each variable can have a value of vj defined in a finite set Dj. A constraint satisfaction problem is defined as a finite set of constraints C = {c1, c2, …, cm} imposed on a set of variables {x1, x2, …, xn}. The problem is feasible if a tuple {v1, v2, …, v3} exists that simultaneously satisfies all constraints C. An objective function that needs to be minimized or maximized over the set of feasible solutions can also be included, which fabricates a constraint optimization problem. (Bockmayr & Hooker, 2005)

2.1.6.1.Global Constraints

Arithmetic constraints are regular constraints like linear and nonlinear equations and inequalities. On the other hand, in constraint programming, there are constraints that are called global. These constraints, in essence, combine a number of simple constraints into new ones where all these variables are considered together. Some examples are:

- Alldifferent: Where all variables x1, x2, …, xn must take different values pairwise.
- Element: An element (i, l, v) means that a variable x in a list of variables l takes a value v, so for example, xi = v.
- Cardinality: Restricts the number of values that a variable can take each time. (Bockmayr & Hooker, 2005)

2.2. Types of scheduling problems in production environments

There are four main types of manufacturing environment scheduling problems, each with numerous variations. The four main categories that we are going to discuss in this paper are Single machine, Flow shop, Job shop, and Open shop. The problems usually fall into two bigger categories that are deterministic and stochastic. In this paper, we will analyze the deterministic approach, which means the processing time for each job is predefined and fixed from the beginning.

2.2.1. Single machine problems

Single machine scheduling problems are the simplest of all scheduling problems, as they involve the use of only one machine and are a special case of the others. The importance of single machine problems, despite their simplicity, is that they have properties that none of the other, more complex problems have, and that is the reason they can provide a basis for heuristic approaches that can be applied into more complex systems. In real-world applications, sometimes complex scheduling problems are divided to simpler subproblems that deal with single machines. One example is that sometimes when a bottleneck in the production chain occurs, it is dealt with as a single machine problem. (Pinedo, 2016)

2.2.2. Flow shop

Flow shop scheduling problems are a design where the machines are in series. Jobs that are scheduled start the process on an initial machine, go through intermediate machines, and finish on the final processing machine. The structure of the arrangement is linear. The order in which a job is processed and the intermediate machines that a job must go through may vary from one job to another in some variants. There is a continuous flow of jobs in sequence that is uninterrupted across a set of machines. It has been an important problem in the scheduling field, and due to the complex industrial and economic applications that it has, and the difficulties that it must overcome to present an effective schedule in a timeframe that is meaningful, the problem has been studied with various assumptions and various objective functions. The first research that provided the methodology to obtain the best sequence of n-jobs on two machines was conducted in 1954 by Johnson, who continued his research on solving 3-machines with constraints. (Zaied, Ismail & Mohamed, 2021)

2.2.2.1.General flow-shop description

The flow-shop scheduling problem is generally determined by a set of machines (M) and a set of unrelated jobs (J), where the number of jobs and machines is finite. Each job (J) has a predefined processing time (P) on each machine (M). The classic flow shop problem assumes that each job must be processed in the same order by each machine (all jobs must go from machine 1 to machine 2 until the last machine). There are a few constraints that must be considered when solving flow shop problems, such as:

- Each machine can process one job at a time.
- Each job can be processed by one machine at a time.
- All processes must be performed.
- The starting time of each job process on machine 1 is zero, as is process time.
- The processes are executed in a predetermined time frame and cannot be interrupted during the execution.
- Machines execute all job processes in the same order. (Parveen & Ullah, 2010)

2.2.2.2.Variants of flow shop

There are many variants of the flow shop problem, like permutation, non-permutation, no-wait, no-idle, hybrid, and flexible flow shops. Given the nature of the variant, some problems have a solution that is optimal for one variant but not for another. We will analyze the most common ones briefly.

2.2.2.2.1. Permutation flow shop

In this variant, it is assumed that all processes for the given jobs must go through the same sequence of machines. The number of possible combinations is n! (where n is the number of jobs), so, for example, if there are 6 jobs, the possible number of combinations is 6x5x4x3x2x1 = 720. As one can understand, the permutation flow shop problem is a combinatorial optimization problem. The annotation of this variant is F | prmu | Cmax, where F annotates that it is a flow shop scheduling problem, prmu that it is of the permutation variant, and Cmax that the objective criterion is captured by the makespan. (Zaied et al., 2021)

2.2.2.2.2. Non permutation flow shop

This variant is a generalization of the permutation flow shop problem, and it allows jobs to change order in the machines. Non-permutation flow shop problems have three specifications, in addition to those of general flow shop problems. The first and second are as described in the general flow shop description (each machine can process one job at a time and jobs have a predefined processing time on each machine), and the third is that the intermediate buffers between each machine are large enough to permit the reordering of the jobs. What makes this subcategory differ from the permutation flow shop is the third specification. In this subcategory, the possibilities that must be searched are n!m, where n is the number of jobs and m is the number of machines. This means that it needs massive amounts of computational power to solve this type of problem. The standard annotation of this variant is F | | Cmax, where F indicates that it is a flow shop scheduling problem and Cmax indicates that the objective criterion is captured by the makespan. Finally, there is a category of problems where the intermediate buffers have a limited storage capacity. (Rossit, Tohmé & Frutos, 2018)

2.2.2.2.3. No wait flow shop

In many real-life applications, there are constraints regarding the production process. In the plastics industry, for example, where the procedure cannot be stopped until the raw material has taken on its final form due to degradation, is a paradigm of no wait flow shop scheduling problem. These types of problems have been proven to be NP-complete. To satisfy the no wait constraint, the completion time of a given job on a given machine must be the same as the starting time of the same job on the next machine. (Sapkal & Laha, 2013; Engin & Güçlü, 2018)

2.2.2.2.4. No idle flow shop

Like in the previous variation, where a job cannot stop its process once it has started, the no idle flow shop scheduling problem has constraints regarding the machines that cannot stop operating once they have started. In real-life cases, once a machine has started operating, the cost of idle time is too high, so the machine must operate continuously. The main objective of this variation is to minimize the time interval between two jobs in the machine. This can be achieved by changing the objective function, which is not always the case since the main objective may be something else, or by making the problem a multi-criteria optimization problem or by adding constraints to the linear problem. (Goncharov & Sevastyanov, 2009)

2.2.2.2.5. Hybrid flow shop

Flow shop scheduling is one of the most common subtypes of flow shop problems. It is a flow shop problem with multiple parallel machines per process. This subproblem may have additional constraints combining the above cases. The main constraint that makes this type of problem special is that the number of machines in at least one stage of the production is greater than one. (Ruiz & Vázquez-Rodríguez, 2010) Machines at each stage can be identical, unrelated, or uniform. The processing complexity of these problems can be categorized into three separate groups:

- Two stage Hybrid flow shop: In this case, there is one machine at stage 1 and more than one in stage two.

- Three stage Hybrid flow shop: Here, there are three stages in the production line where there are parallel and non-identical machines at each stage.
- K-stage Hybrid flow shop: The last category is where there are more than three stages in the production process. Due to its complexity, the usual approach is to decompose the problem into smaller subproblems and deal with them independently. (Linn & Zhang, 1999)

2.2.2.2.6. Flexible flow shop

This is a special category of Hybrid flow shop problems where the machines that are allocated at each stage do not have an identical processing time for a given job. (Wang, 2005)

2.2.3. Job shop

In Job shop problems, the jobs can be processed by any machine in any order. In this type of problem, the general annotation is that machines (m) must process jobs (j). Jobs must be operated by all the machines available in a prespecified order, but this order can be different for each job. Job shop problems, as previously discussed, have many variations that are like flow shop problems, and we will not go over them again, such as no-wait, no-idle, hybrid, and flexible job shops. We will proceed with discussing another kind of constraint that can be applied to scheduling problems. (Parveen & Ullah, 2010)

2.2.3.1.Setup times

Setup time is the time needed to prepare the resources that are required to perform a process. For example, some machines need time to heat up before a process can start or need an interval between two jobs to get cleaned. As it is clear, in many real-life applications, the machines cannot be made immediately available when needed. There are two types of setup time:

- Sequence independent: This type of setup time is only dependent on the process at hand and does not consider its previous processes.
- Sequence dependent: This type depends both on the current and the previous process. For example, in the chemical industry, a machine may be used for manufacturing a chemical that, if mixed with another chemical, could explode, so extensive cleaning is needed. (Sharma & Jain, 2016)

2.2.3.2.Batches and Batch Splitting

Batch production has been used in many cases in mass production environments where jobs can be split into batches, and in this way, productivity efficiency can be improved. The general description of this kind of problem is Batch splitting scheduling problems. In this kind of environment, every batch contains identical jobs. Different batches contain different types of jobs. All machines operate a whole batch at any given time, and all the jobs that are included in the same batch must wait until the last job of a batch is processed to proceed to the next machine, and the machine is occupied until the whole batch is moved to the next machine. In many cases, the batch size can cause a great deal of idle time for the machines and increase the waiting time for jobs. For this reason, batch splitting can be used. This procedure divides the batches into smaller batches so that smaller amounts of jobs can be processed rather than whole batches. (Zhang, Tan, Zhu, Chen & Chen, 2020)

2.2.3.3.Availability constraints

In many real-life cases, there is the possibility that a machine may need maintenance. This constraint usually arises in flexible shop problems when a machine must go offline during the processing procedure. There are two types of availability constraints. The first one is when a machine has scheduled maintenance or reconfiguration. In this case, there can be a forecast, as seen in (Williams, 2013) where there are predefined machines that need to go offline for maintenance for a prespecified period of time. This can be solved by adding extra constraints and variables during the formulation of the problem and solving it with these extra parameters. The other type of availability constraint is when a machine breaks down during a processing procedure that is not scheduled. This type cannot be predicted in advance with certainty. (Zribi & Borne, 2008)

2.2.3.4.Buffer constraints

As discussed, in many cases, unlimited storage of intermediate buffers is taken for granted by the UIS. In real-world cases, though, this is not always true. There are problems where there is no intermediate storage, which is called NIS, and problems where the intermediate storage is finite, which is called FIS. In the unlimited buffer storage case, it is assumed that the intermediate buffer can hold n – 1 jobs, where n is the number of jobs, and there is no bottleneck in this situation. In the event of no intermediate storage, there is a dreadlock in the system. If the next machine is already processing a job and the machine of interest has finished its process, the machine must stay occupied until the next machine is available. (Rossit et al., 2018)

2.2.4. Open shop

Open shop scheduling problems are one of the most frequently encountered scheduling problems where a set of jobs must be processed by a set of machines, and the set of machines that each set of jobs may undergo can be different. One example of open shop scheduling is a medical facility (a hospital), where a patient (job) may have to visit various departments (machines), but the order in which he does so is irrelevant. The departments that each patient must visit are different in general. Each department can treat one patient at a time, and there is no order in which the patient visits each department. The objective of the scheduler is to get a feasible schedule so that a performance criterion is optimized. As discussed, there are a lot of variations in this type of scheduling problem, as well as many constraints that can be of concern, and we will not go through them once again. In this section, we will focus on some of the most common types of objectives that are used, such as makespan, due date, multi-objective, and economic objectives. (Ahmadian, Khatami, Salehipour & Cheng, 2021)

2.2.4.1.General notation of scheduling problems

The scheduling problems can be generally described using the following triplet: α | β | γ. The first field considers the machine environment and can only have one entry. Some examples are: Identical machines in parallel (Pm), Unrelated machines in parallel (Rm), Flow shop (Fm), Flexible flow shop (FJc) and Open shop (Om) where m is the number of machines and c is the number of identical machines in parallel.

The second field describes the details of the characteristics and the processing constraints that may occur in a problem. It can be blank, have a single value, or have multiple values. Some notations that can be used are: Release dates (rj), which means that job j cannot begin before its release date, Preemptions (prmp), which means that a job can be interrupted during its processing, No-wait (nwt).

The third field regards the objective function that is at stake, and we will discuss it in the following sections. (Pinedo, 2016)

2.2.4.1.1. Makespan

Makespan is the most common objective function used in literature. In a 2017 survey (Rossit et al., 2018), 55% of the papers that were reviewed considered makespan as a single objective. Usually, it is denoted as Cmax (Pinedo, 2016). Through this objective function, the goal is to find the minimum time required for the last job to exit the system. (Abdolrazzagh-Nezhad & Abdullah, 2017)

2.2.4.1.2. Total weighted completion time

This is a measure of the sum of the weighted completion times that are needed for the jobs and is notated as $\sum wjCj$ where w is the weight and C is the completion time of the last job j to exit the system. This type of objective function gives an indication of the inventory costs that are added by the schedule. It is also referred to as weighted flow time minimization. (Pinedo, 2016)

2.2.4.1.3. Maximum lateness

Measures the worst violation of the due dates. It is denoted as Lmax and the lateness of a given job i is Li = Ci - di where di and Ci are the due date and the completion time of the job i. If Li is positive, then the job is finished late, and if it is negative, then the job is finished early. (Abdolrazzagh-Nezhad & Abdullah, 2017)

2.2.4.1.4. Total weighted tardiness

This objective is also related to due dates, as mentioned above. The notation is $\sum wjTj$ where w is the weight and T is the tardiness of the last job to exit the system. The difference between lateness and tardiness is that lateness can be negative, but tardiness can have values that cannot be negative. (Abdolrazzagh-Nezhad & Abdullah, 2017; Pinedo, 2016)

2.2.4.1.5. Economic objective functions

The objective of this family of objective functions is to maximize the profit that the organization can gain or minimize the cost. One example is the minimization of cost that a changeover can have on the production plan. Another is the minimization of setup costs for the intermediate buffer. Also, another objective could be the minimization of maintenance costs. Lastly, an objective can be the maximization of profit that is accrued by the organization. (Rossit et al., 2018)

2.2.4.1.6. Multi-objective optimization functions

Vilfredo Pareto, a pioneer of multi-objective optimization, is famous for the Pareto principle (the 80-20 rule) in addition to the Pareto optimality that is relevant to this thesis. The first reference of the Pareto optimality is in his book (Pareto, 1906) which is translated as follows:

> *"We will begin by defining a term which is desirable to use in order to avoid prolixity. We will say that the members of a collectivity enjoy maximum ophelimity in a certain position when it is impossible to find a way of moving from that position very slightly in such a manner that the ophelimity enjoyed by each of the individuals of that collectivity increases or decreases. That is to say, any small displacement in departing from that position necessarily has the effect of increasing the ophelimity which certain individuals enjoy, and decreasing that which others enjoy, of being agreeable to some and disagreeable to others."* (Ehrgott, 2012, p.450)

Multi-objective optimization is the process of setting multiple objective functions that can sometimes be conflicting. For example, an objective function could be to minimize costs and minimize the makespan. For this purpose, in most cases, the way to solve these problems is by reducing them into a scalar problem with the form:

$$\min \hat{J} = \sum_{i=1}^{z} \frac{\lambda_i}{sf_i} J_i$$

where Ĵ is the aggregated, weighted sum of the individual objectives, and λi and sfi are the weight of the objective i and the scale of the vector of the same objective, Ji is the objective function vector, and z is the number of the objectives. (Ehrgott, 2012)

The multi-objective problem is usually broken down into single objective problems, and after solving them individually, a Pareto front (z = 2) or Pareto surface (z > 2) is formulated by combining these solutions. The weighted sum that we described is usually the way to parametrically change the weights through the objective functions and get the Pareto surface. (Kim & De Weck, 2005)

3. Brief presentation of data mining techniques

Data mining is the procedure that describes the extraction of knowledge from data. The term is an analogy borrowed from the mining industry, and it refers to the mining of minerals, as from a large amount of data (a "mine field") one tries to find the meaningful portions (the "minerals") that are of concern. Data mining uses various techniques, such as statistics, machine learning, and artificial intelligence, to achieve its goals. In order to extract useful information, data mining uses databases where existing data are stored, and through data processing, data manipulation, prescriptive statistics, predictive statistics, and other processes, the goals of the users are achieved. There are three main tasks that data mining can achieve. Prediction, association, and clustering. The algorithms that are used, depending on the desired outcome, can be, in most cases, either supervised or unsupervised or, more recently, reinforcement learning (Q-Learning). A typical workflow of machine learning for tabular data is first collecting the data to train the algorithm, then assessing the data using descriptive statistics, performing feature engineering, estimating the parameters, fitting the model, assessing it, and finally deploying it. (Sharda, Delen & Turban, 2017)

3.1. Data Collection

Data collection has been an issue for machine learning. The quality of the data with which an algorithm is trained is one of the major factors that affect the quality of the results of the algorithm. There are many concerns about the quality of the data. Some of them are the source reliability, which refers to the originality and appropriateness of the data storage. It is usually preferable to seek out the source of the data rather than using it through an intermediary in order to gain access to its true nature. Then there is data content accuracy, which means that the data are appropriate for the problem that we want to solve. There is also the issue of data accessibility, which refers to whether the data is readily and easily accessible. We must take under consideration the possibility that data may not be secured and the data privacy. This means that we need to have our data secured and anonymized. Lastly, the data must be consistent. This means that the data must be accurately collected and merged. (Sharda et al., 2017)

3.2. Assessing the Data

Following data collection, we must proceed with the assessment of the collected data. During this stage of the process, we must evaluate, analyze, and interpret the collected data. Some of the means to achieve this are descriptive statistics metrics such as calculating the mean, the median, the variance, the skew, and the kurtosis of each of the variables, which can give us valuable information about them. Also, through a box and whiskers plot, one can find outliers in the data or inconsistencies. After assessing the data individually, one must assess them collectively again through descriptive statistics, some of these tests are correlation analysis and chi square testing. (Sarkar, Bali & Sharma, 2018)

3.3. Data Manipulation – Feature Engineering

Some data manipulation techniques can be used before assessing the data and some after, dependenting on the data and the researcher. Some of the algorithms that we will study in the next chapter cannot handle missing values, are extremely vulnerable to outliers, cannot handle categorical values that are text-based, or need variables to be normalized.

3.3.1. Handle missing values

There are many ways of handling missing values, and the action to be taken is not always straightforward and is case dependent.

- One easy way is to delete the observations that contain missing data. This can usually be done in cases where the dataset is large, and the proportion of observations is small. For smaller datasets, we need to find other ways. Usually, the decision whether to delete the observations or not is made by the researcher who develops the machine learning workflow.
- Another way is by acquiring the missing values. This method, in most cases, is not cost effective and is avoided. (Saar-Tsechansky & Provost, 2007)
- The last way, which is most often used, is by imputing the missing data. This can be done in many ways. One way is by using the mean or median of the values of the other observations' attributes. One other way, not commonly used, is by replacing the missing value with the value that is directly above or below the observation that has the missing value. One last method that is more computationally expensive but yields the best results is training a model and setting the attribute of the missing variable as the target. As we will discuss later, this is done by training a supervised machine learning algorithm and setting as the dependent variable the attribute with the missing value and all the others as the independent ones. As stated above, this method yields the best results, but it is the costliest one.

3.3.2. Outliers

As an outlier, we define an attribute that is not consistent with the others. The outliers can usually be detected by a box and whiskers plot, a q-q plot, and other methods that are used in descriptive statistics. Usually, an outlier is identified when it is more than three standard deviations apart. These outliers can also be treated in many ways. One way is to delete the whole observation, yet another way is through data manipulation and transformation, as we will discuss later.

3.3.3. Categorical values

Categorical values that are represented in text, can usually not be interpreted by the machine learning algorithms. So, we have to replace them with numeric ones. We can use many methods, but the most usual one is assigning a number to each category.

3.3.4. Data normalization – scaling – data transformation

As stated earlier, some algorithms need the data to be normalized. One method for normalizing data is to use min-max scaling, which results in all attribute values being on a scale between [0, 1]. Another method is by standardization, where the following calculation is computed: $x - mean(x) \big/ sqrt(varx))$ where the mean of the feature is equal to 0, and the variance is equal to 1 given that the feature follows a Gaussian distribution. (Zheng & Casari, 2018)

To handle outliers and/or denormalized data (that do not follow the normal – Gaussian distribution), we can use some data transformation techniques. One of the most common transformations is the log transformation. This is used to compress the range of large numbers in our observations and expand the range of the smaller ones. This method is usually used when the distribution of our observations is heavy-tailed, meaning that a large proportion of our dataset is concentrated at the lower end of the distribution curve. Another similar transformation is power transformation, which includes square rooting and squaring the variables. Lastly, there is the box-cox transformation, which, like the square rooting transformation, can be used only when the data is positive. (Zheng & Casari, 2018)

3.3.5. Feature Engineering

Feature Engineering is the process of combining features that are already in the dataset or combining external knowledge with the dataset to create new features. Feature engineering requires knowledge of the domain of the dataset, handcrafted techniques, and mathematical transformations (that we studied above) with the intent to convert data into features to feed the machine learning algorithms. Good feature engineering can result in better representation of data, better performing models, and faster deployment. Some feature engineering techniques are:

- Counts: Instead of treating the whole dataset as individual observations, observations can be grouped – pivoted into single observations and added to a new column with the count of these observations. This can be used to easily identify which observations are the most frequent.
- Binarization: Data can be grouped into two categories, zero and one, for example, with zero representing values that are below a threshold and one above this threshold.
- Rounding: When the dataset has data that contains redundant information that is not needed, these observations can be rounded to represent the data in a manner that retains the needed information. For example, if we need a percentage of some value but the decimal value is not needed, we can round the value to the closest integer.
- Interactions: In a dataset, the interaction of the features can be valuable for understanding it better. One example can be to create a feature that is the multiplication of two existing ones, another is to raise a feature to a power of two and more.

- Binning: This method is used when transforming a continuous feature into a discrete one. These created bins can be thought of as groups to which each datapoint of the feature belongs. One example is with percentages, where we group them by tens. So, one bin would be 0-10%, the other 11-20% and so on. (Sarkar et al., 2018)

3.4. Model Training - Supervised Versus Unsupervised Learning

Supervised learning means that for every predictor measurement xi, there is an associated measurement yi. Supervised Learning can also be divided into two other categories: regression and classification. A regression problem is when one tries to predict a continuous associated measurement such as the price of a house. On the other hand, a classification problem is a problem that tries to predict a qualitative outcome, such as if a customer will churn or not. Whereas in unsupervised learning, one does not have an associated measurement. (James, Witten, Hastie & Tibshirani, 2013) There are many types of algorithms that can be used in each situation and we will analyze some of them in this chapter, including:

- Supervised Learning:
    - Decision Trees
    - Support Vector Machines
    - Naïve Bayes
    - Linear Regression
    - Logistic Regression
    - kNN – K-Nearest Neighbors
- Unsupervised Learning:
    - k – means Clustering
    - Association Algorithms

- PCA – Principal Components Analysis

3.4.1. Supervised Learning

As stated, supervised learning, can be further divided into classification and regression. Some of the algorithms that we presented can be used for either of these categories, so first we will analyze the ones that can only be used in one of the categories.

3.4.1.1.Linear Regression

Linear regression is a relatively simple approach to supervised learning. It is one of the oldest predictive algorithms and the basis, with other methods, of many other algorithms. The goal of the algorithm is to find a correlation between the dependent and independent variables, usually by minimizing the residuals. There are two types of linear regression: simple linear regression with only one independent variable and multiple linear regression with more than one dependent variable. Mathematically, the simple regression model is:

$$Y = b_0 + b_1 * X$$

Where Y is the dependent variable, b0 is the intercept, b1 is the slope, and X is the independent variable. b0 and b1 are also known as the coefficients that linear regression is trying to define so it can predict Y. The most common way that is used to estimate these coefficients is, as stated, by minimizing the residual sum of squares (RSS). Residuals (e) are defined as the difference between the predicted y (ŷ) and the actual y. This can be expressed mathematically as: RSS = $\sum_1^n e_i^2$ where i is the number of observations in the dataset. Multiple regression is the extension of simple regression and is represented as:

$$Y = b_0 + b_1 * X_1 + \cdots + b_n * X_n$$

where n is the number of the predictors.

When implementing a linear regression algorithm, we have to test if the predictors have an impact on the dependent variable or not. This is done by calculating the standard error (SE) for every coefficient b and forming a hypothesis test with n coefficients:

H0: There is no relationship between X and Y (H0: bn = 0)

and H1: There is some relationship between X and Y (H1: bn $\neq$ 0)

Then, through the computation of t-statistics in simple regression or f-statistics in multiple regression, we get the p-value of the coefficient, which shows us whether there is an association between the predictor and the response. A small p-value -lower than 5% or, in some cases, 1%- when the sample is larger than 30 observations suggests that there is a small probability that the relationship between the predictor and the response are associated by chance, and we reject the null hypothesis, whereas a large p-value shows the opposite. (James et al., 2013)

3.4.1.2.Logistic Regression

Logistic Regression is used for classification problems. The logistic regression algorithm is very similar to linear regression. When it was first developed, it could be used for binary classification, but nowadays it is also used in multiclass output variables, for example in multinominal logistic regression. It is used to develop a probabilistic model between the predictors and the response. The predictors can be either continuous or categorical variables, but the response must be categorical. In essence, logistic regression calculates the odds of the dependent variable falling into a category. For example, if the probability of a response being positive is greater than 0.5%, then it is positive otherwise, it is negative. (Sarkar et al., 2018)

3.4.1.3.Naïve Bayes

Naïve Bayes is a classification algorithm as well. It uses Bayes's theorem, and it calculates a set of probabilities using the sum of the frequencies and value combinations in the dataset. The algorithm assumes that all attributes are either independent or non-interindependent with the value of the class variable. Some advantages of the naïve bayes algorithm are that it can handle both quantitative and discrete data, that it is memory efficient and fast, and that it can handle irrelevant attributes. The greatest disadvantage is that it assumes that the variables are independent. (Peling, Arnawan, Arthawan & Janardana, 2017)

3.4.1.4.Support Vector Machines (SVM)

Support Vector Machines are an approach that was introduced in the 1990s and is used for classification. SVM is a generalization of a simple classifier called the maximal margin classifier, which assumes that there is a linear boundary between classes in a hyperplane. SVM, on the other hand, can be used in cases where the boundaries are not linear and can also be used in some cases where there are more than two classes. A hyperplane is a flat affine subspace of a p-dimensional space with p-1 dimensions. So, for example, in a two-dimensional space, the hyperplane is a line, in three dimensions, it is a plane and so on. The goal of the maximal margin classifier and SVM in general, in a two-dimensional space, is to find a line that distinguishes the two categories and has the maximum distance from the observations that are closest to the line. Support vector machines also have the ability, through kernels, to predict an outcome for non-linear responses. (James et al., 2013)

3.4.1.5.K Nearest Neighbors (kNN)

kNN can be used either for classification or regression, although it is mostly used for classification. This algorithm essentially divides the data into contiguous clusters and then classifies the new data into one of these groups based on the previous classification. A crucial hyperparameter that needs to be specified is the number of k-neighbors. When new data is entered in the algorithm for prediction, the label that will be assigned to the new data will be determined by the nearest neighbors, so the number of neighbors to be considered is of the essence. The metric that is used for determining the closed neighbors is usually the Euclidean distance, and the sum of the minimum distance from the neighbors is the one that is selected, and the label is assigned. KNN has many variants, such as Locally adaptive KNN, Weight adjusted KNN, Adaptive KNN, SVM KNN and more. One of the advantages of KNN is that it is simple, comprehensive, and scalable. It is easy to interpret and has a low calculation time. It is effective for large datasets, and it is effective for classification and regression. A disadvantage is that it is difficult and resource consuming to determine the optimal number of neighbors (k) that are optimal in large datasets. (Taunk, De, Verma & Swetapadma, 2019)

3.4.1.6.Decision Trees

Decision Trees are used either for classification or for regression. Decision trees are segmenting the predictor space into simple regions. They use splitting rules to further segment a space into smaller subspaces.

Decision trees are constructed using leaves and branches – nodes. A node is a split in the space, for example, in a dataset where we want to predict the price of a house, dependent variable, and the independent variables are the area of the house in square meters and the number of bedrooms that a house has. The first node would be if the house has more than two bedrooms or not, and if it has more than two, then the second branch would be if the house is larger than 60 square meters. The predicted price of a house would be the leaf, which means that the leaf is the prediction that is made based on the decision tree's branches. In this example, the number of bedrooms is the most important factor for the price of the house. (James et al., 2013) The decision about where a node should be split is one of the most important decisions that must be made. As decision tree algorithms such as CART (Classification And Regression Tree), ID3 (Iterative Dichotomizer 3) and C4.5 are mostly greedily searching for a tree, the algorithms are trying to find one of the following cost functions in each step:

- Mean Squared Error: MSE is used for regression and is the square difference between observed and predicted values.
- Mean Absolute Error: MAE is also used for regression and is the absolute difference between the observed and predicted values.
- Variance Reduction: It was introduced for the first time with the CART algorithm and is the standard variance formula, and we choose the split that gives us the least variance.
- Gini Impurity/Index: Gini is used mainly for classification trees and measures the chance that a randomly selected data point will have an incorrect label given that it was labeled randomly.
- Information Gain / Entropy: Entropy is primarily used for classification, and we choose to split a node based on the amount of information gained; the higher the entropy, the better.

As a stopping criterion for the decision trees, there are many different choices. As we will discuss, it is also an important aspect of the algorithm, as the right choice will prevent us from overfitting the model. One stopping criteria is the minimum count of data points. Implementing this criterion, the algorithm will stop when, for example, we have set that for a leaf, there must be at least a certain number of observations. Another criterion is the maximum depth, which we will look at in a later chapter, and which means we tell the algorithm to stop after a certain number of nodes have been reached. As a final note, there are many hyperparameters that we can specify during the design of the tree that can help us avoid overfitting-underfitting and get better results, such as maximum depth, minimum samples to split internal nodes, and more that we will discuss in a later chapter. (Sarkar et al., 2018)

Here we will introduce the concept of overfitting and underfitting in machine learning through tree pruning. In our example, we could divide the space into smaller segments and fit our model in the training set to obtain a result that perfectly predicted the outcome for all of our training datasets. This would lead to overfitting our trained model in the training dataset, and when a new test set is introduced that has new observations, it would perform badly due to the complexity of our model and lead to bad predictions. Overfitting can also occur due to the noise, outliers, or errors in a training dataset, which leads the algorithm to follow these issues too closely and keep reproducing the same error. (James et al., 2013) One way of overcoming this problem is by holding a subset of the original training dataset and performing cross-validation, which we will analyze later. Another pitfall – the opposite of overfitting – is underfitting, which means that our model is not trained properly and makes erroneous and very rigid assumptions about the data. (Sarkar et al., 2018)

This is part of the Bias – Variance tradeoff. Bias is introduced in the model when it makes wrong assumptions about the parameters of the data. The bias error is the difference between the actual value that we are trying to predict and the predicted one that arises from our model. For example, a linear regression model assumes that the relationship between the dependent and independent variables is linear and can miss crucial relationships between the features (independent variables) and the output (dependent variable). Variance, on the other hand, is the sensitivity of the model to fluctuations in the dataset that can be due to new features, randomness, outliers, noise, and other factors. For example, if we train a decision tree on a dataset that has a lot of outliers with a linear relationship between the dependent and independent variables, then the predictions will not be as good as a linear regression model due to the high variance that our tree will have. (Sarkar et al., 2018)

In this thesis, we use a decision tree-based algorithm named XGBoost, so we will focus in a later chapter on various methods that are used in trees such as bagging and boosting and some implementations of these techniques such as random forest, adaboost, and XGBoost.

3.4.2. Unsupervised Learning

Unsupervised learning is used when there is no label – dependent variable to predict but rather when we need to discover and recognize interesting things about the "independent" variables. There are three main categories of unsupervised learning: Clustering, Dimensionality Reduction, Anomaly Detection and Association Rule-Mining. (Sarkar et al., 2018)

3.4.2.1.Clustering

Clustering methods try to find patterns of relationships in the data. These methods divide the data into categories like groups that have some similarities between them based on their features or attributes, and cluster them together based on these features. They are very powerful if used in the right way, as they can give us a bigger picture of the data that we want to make sense of. There are many types of algorithms that can be used, such as (Sarkar et al., 2018):

- K-means which is centroid based. This algorithm requires the number of clusters into which we want to divide our observations as input, and it returns the cluster to which each observation belongs based on the observations. For the K-means clustering algorithm, the best outcome is the one that has the least variation inside each cluster. One of the biggest challenges is to find the optimum number of clusters into which the data should be divided. (James et al., 2013)
- Hierarchical clustering, which does not require a prespecified number of clusters. Also, another advantage of hierarchical clustering is that the result is a tree-based representation of the result, called a dendrogram, which is more appealing. (James et al., 2013)
- Distribution based clustering.
- Density based methods. (Sarkar et al., 2018)

3.4.2.2.Dimensionality Reduction Algorithms

Dimensionality reduction algorithms are usually used when a dataset is excessively large and there are a huge number of features. One of the most popular algorithms that are used to reduce the number of features is PCA – Principal Component Analysis which essentially groups features together, creating new features that include a portion of the original data and removing the old ones. PCA assumes that, for each observation, not all the attributes are equally important and groups them together into groups and, that each principal component (new feature) is a linear combination of the attributes that the observation has. (James et al., 2013)

There is another category of dimensionality reduction called Feature Selection where only some of the original features are selected to be analyzed and no new features are generated. (Sarkar et al., 2018)

3.4.2.3.Association Algorithms

This category of unsupervised learning tries to find associations between variables in large datasets. One of the most popular algorithms is Apriori, and one of the most common implementations is market-basket analysis. The main purpose of market-basket analysis is to find which products are bought together and have strong relationships with each other. The input in association algorithms is usually transactional data, where products that are purchased together are tabulated in the same transaction instance. Association mining uses two metrics: support and confidence. Suppose there are two objects, X and Y, that we want to find out if they are related. The hypothesis is that if a customer buys X, Y will be bought as well. Support is the number of baskets that contain both objects X, Y divided by the total number of baskets, and confidence is the Support found above divided by the value of the first object, X in our case.

As mentioned, Apriori is the most common algorithm. Given a set of items, the algorithm tries to find subsets of the itemset that are present in a minimum number of itemsets. It begins by creating one-item subsets, then two-item subsets, and so on. Those subsets that have a Support that is lower than a given number, which is usually provided by the user, are stopped by the algorithm from looking for further combinations. (Sharda et al., 2017)

3.4.3. Reinforcement Learning

Reinforcement Learning is a method that does not fall into either supervised learning or unsupervised learning. It uses an agent that is trained over a period of time and interacts with a specific environment. The agent improves as a result of a reward or penalty assigned to its most recent move. Usually, the agent starts with a predefined set of strategies and interacts with the environment. The agent then continues by updating the current strategies as needed using the reward/punishment method. Through this iterative process, the agent "learns" to select the optimal strategy given the circumstances. An example is an agent that is trained to play chess. First, some guidelines are provided, then the agent observes the current state, selects the optimal strategy, and performs the action, gets the reward or penalty, updates the strategy, and repeats until it learns the optimal policies. AlphaGo AI was created by Google's DeepMind to train an agent to play the game of Go. (Sarkar et al., 2018)

3.4.4. Neural Networks and Deep Learning

Deep learning and neural networks are algorithms that use multiple processing layers to train on data representations with abstraction. Neural network and Deep learning definitions refer to the same category of algorithms, with Deep learning being the new one. These algorithms try to imitate the learning mechanisms of biological organisms. The algorithm has the same structure as the components of biological nervous systems, such as neurons, axons, and dendrites. Weights are assigned to the connections of neurons in order to simulate the strength of the connections between the neurons. A function is used to compute the inputs and outputs of the neurons. In general, deep learning algorithms tend to provide predictions that are more accurate than traditional machine learning algorithms, but they still lack interpretability and usually need a large amount of training data to accomplish that. (Aggarwal, 2018)

There are many types of Neural networks, which we will shortly refer to.

First, there are single-layer and multi-layer Neural networks. Single-layer models use a set of inputs that is directly mapped to an output layer, usually using a generalized variant of a linear function. Single-layer networks are also known as perceptrons. This basic structure is used in multiple-layer networks, but there are one or more hidden layers between the input and output layers. (Aggarwal, 2018)

There are also various activation and loss functions through which the neuron is activated or deactivated. Inside a neuron, there are essentially two activation functions. One pre-activation is typically used in a backpropagation algorithm, and one post-activation is the neuron's output value. The activation functions that are most often used are the sigmoid, sign, tanh, hard tanh, and reLU. (Aggarwal, 2018)

Last, we will discuss some of the common Neural Network Architectures.

- Simulating basic Machine learning with shallow models: The classic machine learning algorithms that we studied earlier can usually be represented by neural networks with one or two layers.
- Recurrent Neural Networks: These are used for sequential data like text and time-series.
- Convolutional Neural Networks: These are used for computer vision tasks such as image classification or object detection. (Aggarwal, 2018)

3.5. Model Assessment

After a model is trained, we have to assess its performance. There are two main ways to assess a model. The first is by holding a part of the dataset to test the results. The dataset is usually split randomly into two proportions. Two thirds are used to train the model, and the rest is never presented during the training phase but is held to test and evaluate the model. This is called a simple train test split. In the case of a neural network, the dataset is split into three parts: training, validation, and testing, to prevent overfitting. This method is not used because, due to its randomness, it can lead to a train – test split that can be skewed, especially in classification problems. The other method is to use k-fold cross-validation. To reduce skewness and bias, the dataset is divided into mutually exclusive datasets, the number of which is defined by the number k, and the algorithm is trained iteratively with k - 1 datasets, with the final part serving as the test set. (Sharda et al., 2017) There is one variation of k-fold cross-validation that is used in this thesis that is called stratified cross-validation. This variation sneak peeks into the target value and tries to create balanced folds, and the dataset is divided into folds that have the same amount of zeros and ones in each fold.

The model's performance will be evaluated using metrics such as accuracy, sensitivity, F-measure (F-1 score), auc, and others. First, a confusion matrix must be introduced. A confusion matrix is used for evaluating a binary classification algorithm and is a comparison of true positives, which have a positive value and are predicted positive; true negatives, which have a negative value and are predicted negative; false positives, which have a negative value but are predicted positive; and false negatives, which have a positive value but are predicted negative, as shown in the table below: (Hossin & Sulaiman, 2015)

	Actual Positive	Actual Negative
Predicted Positive	*True Positive (tp)*	*False Negative (fn)*
Predicted Negative	*False Positive (fp)*	*True Negative (tn)*

- Accuracy: Accuracy is the sum of True Positives and True Negatives divided by all the observations. It is the simplest and most intuitive assessment method for classification problems. Although it is often used as a measure, it does not make a distinction between true positives and true negatives, and this can lead to false assumptions if the dataset is not balanced (has the same amount of positive and negative observations). (Japkowicz, 2006)
- Precision (p): Measures the ratio of true positives to the sum of true positives and false positives. Precision evaluates the extent to which the classifier correctly classifies observations as positives. (Japkowicz, 2006)
- Recall (r): Recall is the division of true positives by the sum of true positives and true negatives. Usually, precision and recall are calculated together to indicate the predictive capabilities of a model.
- F-Measure (FM): F-Measure, also known as F1-score, is a metric that combines recall and precision. It represents the harmonic mean between recall and precision and is given by the formula:

$$2 * \frac{p * r}{p + r}$$

- Geometric mean (GM): Is the square root of the multiplication between true positives and true negatives. It is used to maximize the true positive and true negative rates while keeping them relatively balanced. Both F-Measure and Geometric mean are considered to be better measures than the others mentioned in optimizing a binary classification algorithm.

4. Implementation of our proposal

4.1. Previous work

As we analyzed in the first part of this paper, scheduling problems are computationally expensive to solve. The idea behind this thesis is that in a production environment like a flow shop, an optimal or near optimal solution is provided to the machine learning algorithm, and through proper data manipulation, the algorithm is introduced with a binary solution where a certain job is performed before another (1) or after another (0). There are other implementations of this approach such as (Benda, Braune, Doerner & Hartl, 2019), (Shahzad & Mebarki, 2012) and (Li & Olafsson, 2019). To train our algorithm, (Taillard, 1993) benchmark instances were used and the solutions that were selected were found at <http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html> accessed on 20 December 2022.

In (Shahzad & Mebarki, 2012) the system that was proposed, consisted of a four-stage modular approach. The first module was the optimization module, where a tabu search algorithm provided the most relevant solution to a scheduling problem. Then a simulation module that was working in parallel with the learning module, which is the third module, transformed the solution of the first module into a set of attribute-pairs with decision variables indicating whether a job proceeds to the next stage or not. Then in the learning module, a C4.5 tree decision algorithm was used. Lastly, the third stage was a control module that generated the relevant scheduling problem instances.

In (Li & Olafsson, 2019), a flat file with the processing times of each job on each machine was needed. The proposal was to discover dispatching rules using data mining. The goal of the paper is to complement more traditional approaches to operations research. Their implementation, as previously mentioned, includes a C4.5 algorithm that is trained to learn new dispatching rules. Through data engineering and feature selection, the algorithm is able to improve its predictions.

In (Benda et al., 2019), a list of attributes of a real-life problem were selected through testing and were the input of a decision tree that uses a C4.5 algorithm to obtain a dichotomous tree. The tree is trained using a constraint programming solution and a shop floor simulator. Because the decision tree has the potential to overfit the training dataset, a random forest algorithm is used to counteract it. They used 80 instances that were randomly generated for each scenario in order to train the random forest algorithm according to the configurations that they set.

4.2. Implementation

4.2.1. Instances

As stated, in our training (Taillard, 1993) instances were used. Taillard proposed 260 randomly generated scheduling problems that correspond to the actual dimensions of industrial problems. In his work, permutation flow shop, open shop, and job shop problems are introduced. The problems are basic, meaning the processing times of the jobs are fixed, there are no set-up times, due dates, or release dates, and the objective is minimization of makespan. The problems, as stated in previous chapters, have the notations F||Cmax, J||Cmax, and O||Cmax, respectively. The proposed algorithm can be implemented in flow shop scheduling problems. As stated, in flow shop problems, each job must follow a sequential order, first processed in machine 1, then machine 2, etc. In his work, Taillard used Taboo search to get good solutions to the problems that he proposed because, as we discussed earlier, it is an easy to implement algorithm and provides good solutions. After using a random number generator to produce a large number of problems, Taillard used Taboo search to determine the 10 "hardest" and publish them. A list of these problems can be found at <http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html> accessed on 20 December 2022 and some optimum solutions can be found as well. We chose instances with an existing best solution rather than those with known lower and upper bounds to train our algorithm.

4.2.2. Machine learning algorithm

As mentioned above, the problem that we study will be a binary classification problem. The XGboost classification algorithm was selected to be trained. This algorithm was introduced in (Chen & Guestrin, 2016). Since then, it has been an award-winning machine algorithm with great results. The algorithm uses tree boosting which is a widely used and highly effective machine learning method. Before we analyze how XGboost is implemented, we will explain what bagging and boosting are. Bagging and boosting are part of the ensemble methods. Ensemble methods combine many simple, weak learner models in order to obtain one model that is potentially better than any individual. Decision trees are machine learning algorithms that can be vulnerable to high variance, so if we train the algorithm with two subsets of the same dataset, we can have results that vary greatly from each other. Essentially, what bagging does is split the dataset into smaller ones, train each of them separately, and then average them to obtain a single learning model with low variance. Bagging is a method that can be used in many regression algorithms, but it is mostly used in decision trees. In classification problems, a similar method is used. A voting mechanism is used to predict the class of the observation. Random forest is an improvement over bagged trees, as in every iteration only a portion of the predictors are used, allowing weak predictors to participate, and thus producing results that are even less variable and more reliable. Boosting, and specifically gradient boosting, can be used either for classification or regression. Boosting, like bagging, can be used for any machine learning algorithm but is usually used in decision trees. Boosting works in a similar way as bagging, but the main difference is that the trees are grown sequentially. This means that after the first tree is built, the next tree is ignorant of the initial dataset but is trained on the residuals of the previous tree, and each time we update the fitted function with the new findings. There are three parameters for boosting that need to be specified. The first is the number of trees that need to be created; l is the shrinkage parameter and controls the learning rate of the algorithm; and d is the number of splits in each tree. (James et al., 2013) XGboost is using shrinkage and subsampling as complementary methods to avoid overfitting. All the information related to the XGboost algorithm can be found in Chen and Guestrin, 2016. (Chen & Guestrin, 2016)

4.2.3. Tools

To develop our solution, we used the following tools. Anaconda, which is free software, includes many libraries that are used for data science and research. Anaconda is an I.D.E. (Integrated Development Environment) through which access is given to different environments and access to Python or R. Through Anaconda, access to the console is possible. The version of Anaconda used is 4.12.0.

Python is one of the most widely used programming languages, especially for data analysis and machine learning applications. This is due to the fact that it has a wide range of packages that are used for data analysis, such as scikit-learn, pandas, and many more, that are either preinstalled or easy to install. Another benefit is that Python is a programming language that is easy to learn and use, is easily modifiable, and is fast for prototyping. The version of Python used is 3.7.10. (Sarkar et al., 2018)

The packages that are used in python are:

- Pandas (Version 1.2.3): Pandas is a powerful package used for data manipulation, wrangling, and analysis. Pandas uses two main data structures: Series and Dataframes. Series are basically indexed one-dimensional arrays. Dataframes are the most important data structures in Pandas. A dataframe can contain heterogeneous data. Dataframes, through manipulation can perform all kinds of operations along the rows and columns of a dataset. Pandas also provide ways to import, retrieve, read, and export data. (Sarkar et al., 2018)

- Scikit-learn (Version 0.24.1): Scikit-learn (sklearn) is one of the most important packages for data analysis and machine learning. This package implements many of the machine learning algorithms that are used in machine learning. It was released for public use in late 2010 (Sarkar et al., 2018). Most of the data evaluation, preprocessing, and model selection were implemented with this package.
- XGboost (Version 1.5.0): XGboost was used to build the model as stated before. This implementation is part of the XGboost algorithm specifically developed for Python. XGboost is an algorithm that is not included in sklearn and has also implementations in R and JVM.
-

5. Data Analysis

In this chapter we will analyze the process used to extract the results that will be presented.

5.1. Data Preparation

As we previously stated, in this thesis, a schedule of flow-shop jobs is required. This can be obtained by any means, including using preexisting schedules, integer programming, simulation, etc. The data for the algorithm's training are imported as the time required to complete each job in each machine from Taillard instances. We will explain the process with a simple numerical example of 3 jobs and 2 machines. Let's assume that the optimal schedule is Job 2, Job 3, Job 1:

I. The data that is imported is in the form:

	Machine 1	Machine 2

Job1	45	24
Job 2	23	27
Job 3	27	42

In this example, Job 1 (J1) needs 45 time slots to be processed in Machine 1 (M1), Job 2 (J2) needs 23 time slots in Machine 1 (M1), etc.

II. We take the cartesian product of this dataset, with itself and the dataset and we delete rows that have excess data. The final dataset on which we train the algorithm has the following form:

Job1	Job1_M1	Job1_M2	Job2	Job2_M1	Job2_M2	Job1 Schedules first
J1	45	24	J2	23	54	No
J1	45	24	J3	19	42	No
J2	23	54	J3	19	42	Yes

III. Next, we create new features with feature engineering using the interactions between the variables.

IV. We train the algorithm with the data from a Taillard instance and then test it with the data from another.

V. Last, as our goal is primarily to obtain a schedule that will be instantly used in the manufacturing environment, we take a list of jobs that have a voting system to show the schedule that should be followed.

Many experiments were conducted. We will present instances of the following sizes:

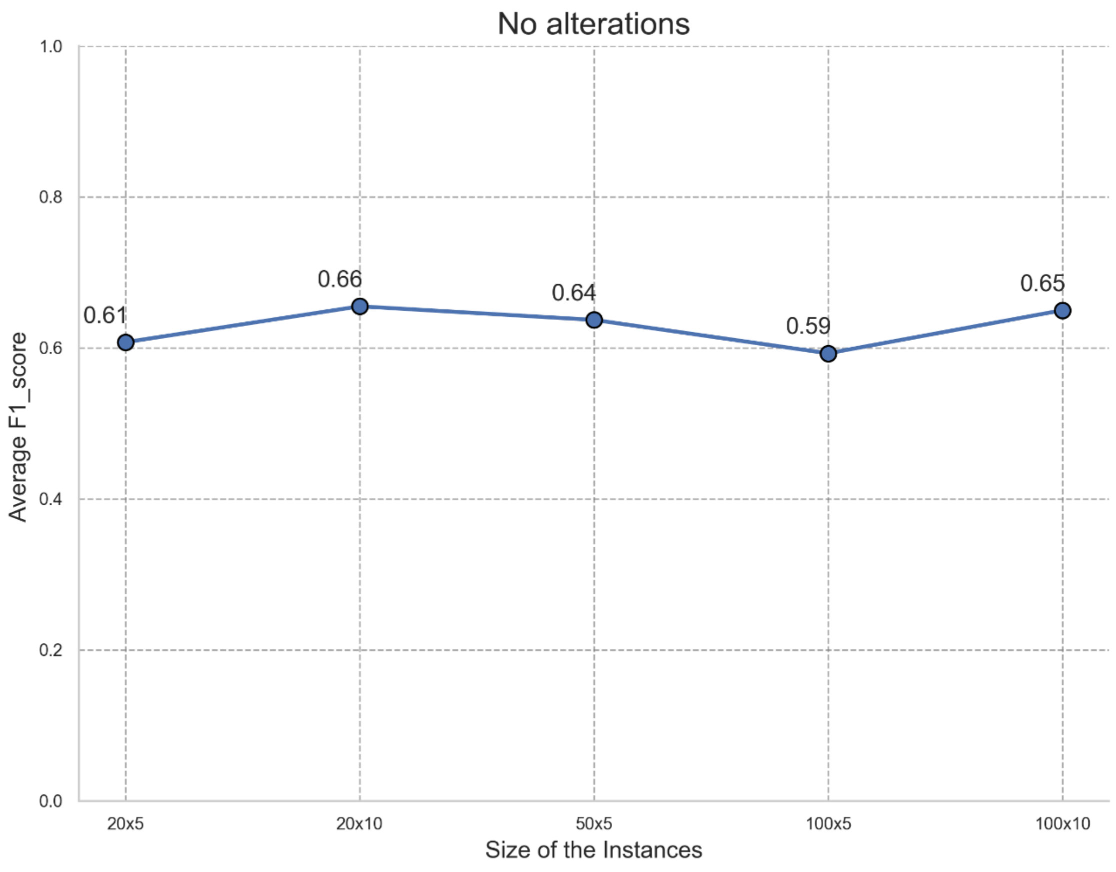
- 20 jobs 5 machines,
- 20 jobs 10 machines,
- 50 jobs 5 machines,

- 100 jobs 5 machines and
- 100 jobs 10 machines

The method that was used is to train the algorithm with an instance and then make a prediction on four other datasets of the same size and get the average of their f1-scores:

In the first experiment, the default settings of XGboost were used and no further features or Grid-Search was conducted.

Instance Size	20x5	20x10	50x5	100x5	100x10
Average F1_score	0.6075	0.655	0.637	0.59275	0.64975

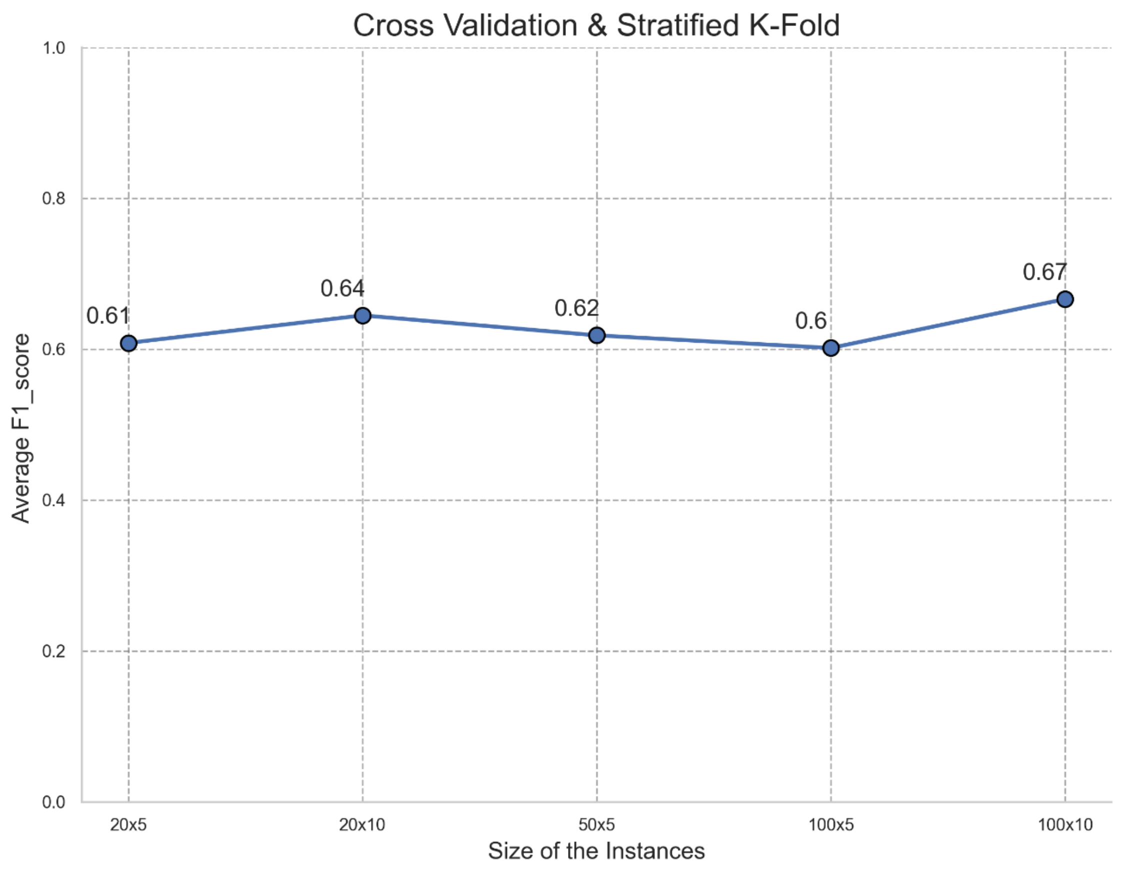


In the second experiment, Randomized Grid Search was used with the parameters given bellow with Stratified Cross-Validation. Since the datasets are varying in size and shape, different parameters were used. For the first four sizes we used a 10-fold cross validation and in Stratified Cross-Validation we used 50 iterations for Randomized Grid Search, which means that the algorithm used 50 different combinations of the following grid:

```
param_grid = {
   "max_depth": [3, 4, 5, 7],
   "learning_rate": [0.1, 0.01, 0.05],
   "gamma": [0, 0.25],
   "reg_lambda": [0, 1, 10],
   "scale_pos_weight": [1, 3, 5],
   "subsample": [0.7,0.8,0.9],
   "colsample_bytree": [0.5,0.6,0.7],
   'min_child_weight': [1, 3, 5, 7],
   'n_estimators': [100, 500, 1000, 2000, 3000],
   'reg_alpha': [0.01, 0.05, 0.1, 0.5, 1.0],
   "use_label_encoder" : [False]
}
```

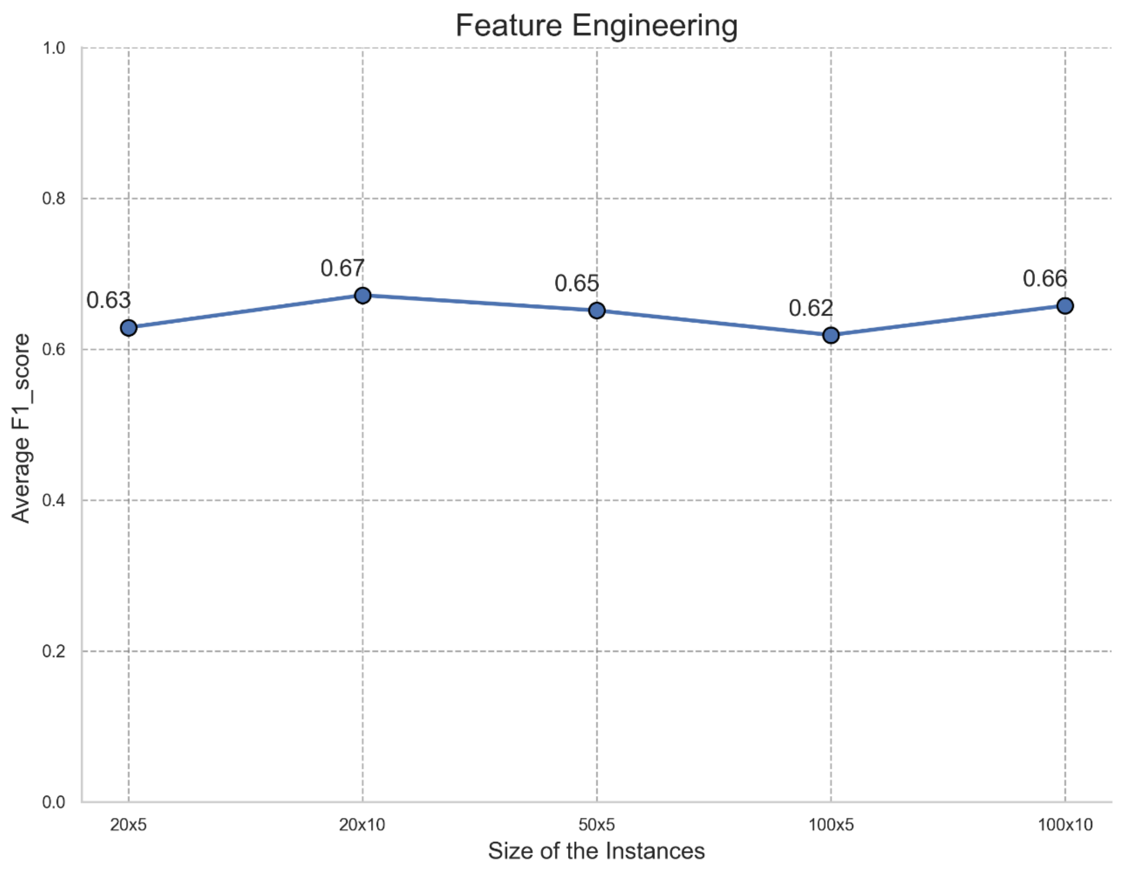
The results were:

Instance Size	20x5	20x10	50x5	100x5	100x10
Average F1_score	0.60825	0.64475	0.61825	0.6015	0.66525

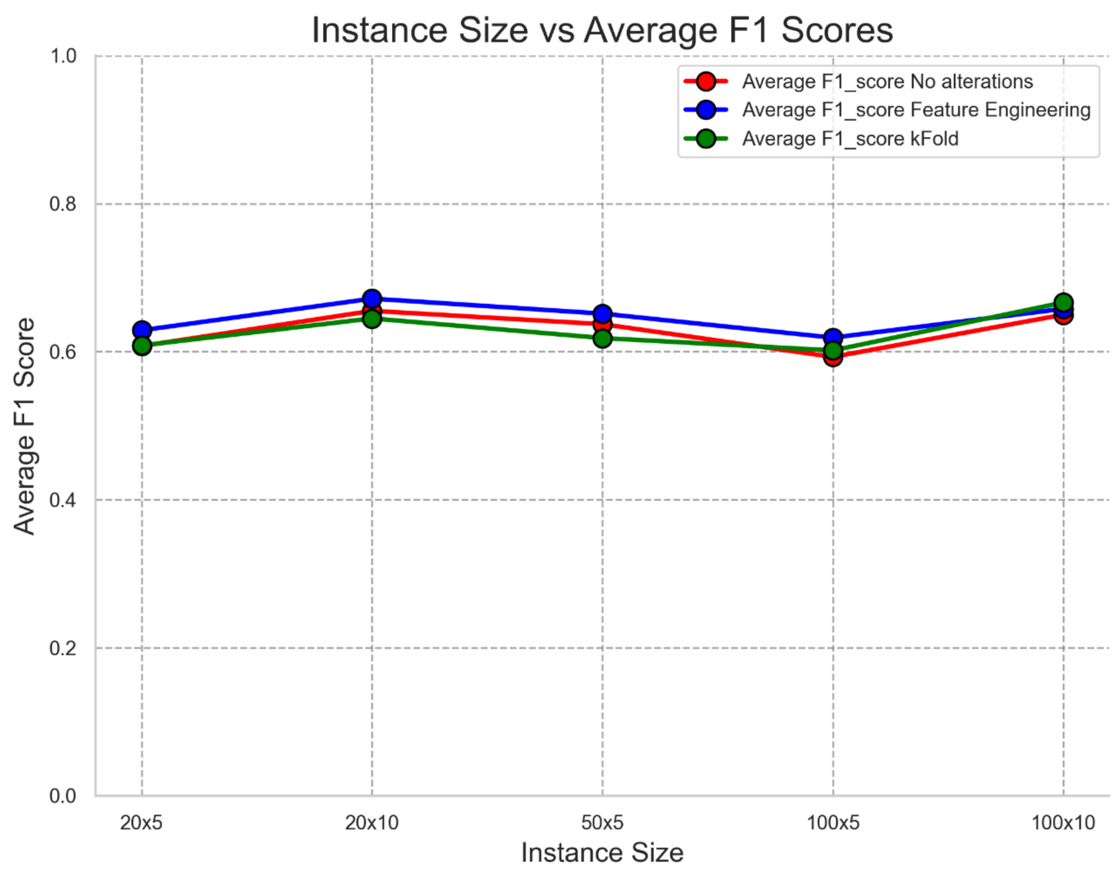


In the last experiment we used feature engineering. We used PolynominalFeatures from sklearn.preproc with default settings. The results were as follows:

Instance Size	20x5	20x10	50x5	100x5	100x10
Average F1_score	0.62875	0.6715	0.65125	0.61875	0.65775



In the following graph, we present the three experiments that we conducted in a comparative way.



6. Limitations and further research

As seen in the results, our algorithm does not perform well with the datasets that we had at hand. (Taillard, 1993) Instances, as stated before, are randomly generated. Machine learning algorithms are looking for patterns in the data to predict the outcome of new datasets. In our best knowledge, if a real-life dataset is introduced to the algorithm, it will perform better with higher scores, in particular the F1-score, and predict the outcome more accurately. Another drawback of the dataset is that the only information it provides is the time a job needs to be processed by a machine; no further information is provided, such as buffers, blockers, or setup times, that would give a realistic outcome. Another limitation is that in a real-life dataset, more expert knowledge would be available, and more features would be added that are not directly measured. Lastly, there was a limitation in resources, as the training of the model was time-consuming, so we could not conduct all the experiments that we would want in order to yield better results.

As stated, more experiments must be conducted to yield better results. The algorithm should be tested in a real-life environment, and all hyper-parameters of XGBoost should be explored to get better results. As seen in our results, as more features are added, the F1-score is improving.

7. Conclusion

In this thesis we explored an alternative way to solve scheduling problems. Aside from traditional integer programming techniques like branch and bound and cutting planes, which have a high computational cost, and metaheuristics, which can produce an answer but there is no way to know if it is the optimal or how close to the optimal it is, the proposed method is both time-efficient and provides an answer to how close the solution is to the optimal. So, in a production environment, it is possible to execute an integer program on an infrequent basis and then, for day-to-day activities, run the machine learning algorithm to have a near optimal schedule and know at what percentage the schedule is optimal.

Bibliography

Abdolrazzagh-Nezhad, M., & Abdullah, S. (2017). Job shop scheduling: Classification, constraints and objective functions. *International Journal of Computer and Information Engineering*, *11*(4), 429-434.

Aggarwal, C. C. (2018). Neural networks and deep learning. *Springer*, *10*, 978-3.

Ahmadian, M. M., Khatami, M., Salehipour, A., & Cheng, T. C. E. (2021). Four decades of research on the open-shop scheduling problem to minimize the makespan. *European Journal of Operational Research*, *295*(2), 399-426.

Anand, E., & Panneerselvam, R. (2015). Literature review of open shop scheduling problems. *Intelligent Information Management*, *7*(01), 33.

Barták, R. (1999, June). Constraint programming: In pursuit of the holy grail. In *Proceedings of the Week of Doctoral Students (WDS99)* (Vol. 4, pp. 555-564). Prague: MatFyzPress.

Benda, F., Braune, R., Doerner, K. F., & Hartl, R. F. (2019). A machine learning approach for flow shop scheduling problems with alternative resources, sequence-dependent setup times, and blocking. *OR spectrum*, *41*(4), 871-893.

Bockmayr, A., & Hooker, J. N. (2005). Constraint programming. *Handbooks in Operations Research and Management Science*, *12*, 559-600.

Chandru, V., & Rao, M. (1998). Integer programming. *IIM Bangalore Research Paper*, (110).

Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 785-794).

Dean, Walter, "Computational Complexity Theory", *The Stanford Encyclopedia of Philosophy* (Fall 2021 Edition), Edward N. Zalta (ed.), [Online] Available from: <https://plato.stanford.edu/archives/fall2021/entries/computational-complexity/>. [Accessed: 25 December 2022].

Dorigo, M., & Blum, C. (2005). Ant colony optimization theory: A survey. *Theoretical computer science*, *344*(2-3), 243-278.

Eglese, R. W. (1990). Simulated annealing: a tool for operational research. *European journal of operational research*, *46*(3), 271-281.

Ehrgott, M. (2012). Vilfredo Pareto and multi-objective optimization. *Doc. math*, 447-453.

Engin, O., & Güçlü, A. (2018). A new hybrid ant colony optimization algorithm for solving the no-wait flow shop scheduling problems. *Applied Soft Computing*, *72*, 166-176.

Gendreau, M., & Potvin, J. Y. (Eds.). (2010). *Handbook of metaheuristics* (Vol. 2, p. 9). New York: Springer.

Genova, K., & Guliashki, V. (2011). Linear integer programming methods and approaches– a survey. *Journal of Cybernetics and Information Technologies, 11*(1).

Goncharov, Y., & Sevastyanov, S. (2009). The flow shop problem with no-idle constraints: A review and approximation. *European Journal of Operational Research*, *196*(2), 450-456.

Hossin, M., & Sulaiman, M. N. (2015). A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, *5*(2), 1.

Ibaraki, T. (1976). Theoretical comparisons of search strategies in branch-and-bound algorithms. *International Journal of Computer & Information Sciences*, *5*(4), 315-344.

Jaffe, A. M. (2006). The millennium grand challenge in mathematics. *Notices of the AMS*, *53*(6), 652-660.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112, p. 18). New York: springer.

Japkowicz, N. (2006, July). Why question machine learning evaluation methods. In *AAAI workshop on evaluation methods for machine learning* (pp. 6-11).

Jünger, M., & Naddef, D. (Eds.). (2001). *Computational combinatorial optimization: optimal or provably near-optimal solutions* (Vol. 2241). Springer.

Kim, I. Y., & De Weck, O. L. (2005). Adaptive weighted-sum method for bi-objective optimization: Pareto front generation. *Structural and multidisciplinary optimization*, *29*(2), 149-158.

Kumar, M., Husain, D., Upreti, N., & Gupta, D. (2010). Genetic algorithm: Review and application. *Available at SSRN 3529843*.

Lambora, A., Gupta, K., & Chopra, K. (2019, February). Genetic algorithm-A literature review. In *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)* (pp. 380-384). IEEE.

Li, X., & Olafsson, S. (2005). Discovering dispatching rules using data mining. *Journal of Scheduling*, *8*(6), 515-527.

Linn, R., & Zhang, W. (1999). Hybrid flow shop scheduling: a survey. *Computers & industrial engineering*, *37*(1-2), 57-61.

Loui, M. C. (1996). Computational complexity theory. *ACM Computing Surveys (CSUR)*, *28*(1), 47-49.

Mitchell, J. E. (2002). Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, *1*(1), 65-77.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., & Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, *19*, 79-102.

Müller-Merbach, H. (1981). Heuristics and their design: a survey. *European Journal of Operational Research*, *8*(1), 1-23.

Pareto, V. (1906). L'ofelimità nei cicli non chiusi. Giornale degli economisti, 33, 15-30.

Parveen, S., & Ullah, H. (2010). Review on job-shop and flow-shop scheduling using. Journal of Mechanical Engineering, 41(2), 130-146.

Peling, I. B. A., Arnawan, I. N., Arthawan, I. P. A., & Janardana, I. G. N. (2017). Implementation of Data Mining To Predict Period of Students Study Using Naive Bayes Algorithm. *Int. J. Eng. Emerg. Technol*, *2*(1), 53.

Pinedo, M. L. (2016). *Scheduling* (Vol. 29). New York: Springer.

Pryor, J., & Chinneck, J. W. (2011). Faster integer-feasibility in mixed-integer linear programs by branching to force change. *Computers & Operations Research*, *38*(8), 1143-1152.

Roh, Y., Heo, G., & Whang, S. E. (2019). A survey on data collection for machine learning: a big data-ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, *33*(4), 1328-1347.

Rossit, D. A., Tohmé, F., & Frutos, M. (2018). The non-permutation flow-shop scheduling problem: a literature review. *Omega*, *77*, 143-153.

Ruiz, R., & Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. *European journal of operational research*, *205*(1), 1-18.

Saar-Tsechansky, M., & Provost, F. (2007). Handling missing values when applying classification models.

Sapkal, S. U., & Laha, D. (2013). A heuristic for no-wait flow shop scheduling. *The International Journal of Advanced Manufacturing Technology*, *68*(5), 1327-1338.

Sarkar, D., Bali, R., & Sharma, T. (2018). Practical machine learning with python. *A Problem-Solvers Guide To Building Real-World Intelligent Systems. Berkely: Apress*.

Scheduling instances Published in E. Taillard, "Benchmarks for basic scheduling problems", EJOR 64(2):278-285, 1993. [Online] Available from: <http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html> . [Accessed 20 December 2022]

Shahzad, A., & Mebarki, N. (2012). Data mining based job dispatching using hybrid simulation-optimization approach for shop scheduling problem. *Engineering Applications of Artificial Intelligence*, *25*(6), 1173-1181.

Sharda, R., Delen, D., & Turban, E. (2017). *Business intelligence, analytics, and data science: a managerial perspective*. pearson.

Sharma, P., & Jain, A. (2016). A review on job shop scheduling with setup times. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, *230*(3), 517-533.

Sörensen, K., & Glover, F. (2013). Metaheuristics. *Encyclopedia of operations research and management science*, *62*, 960-970.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *european journal of operational research*, *64*(2), 278-285.

Taunk, K., De, S., Verma, S., & Swetapadma, A. (2019, May). A brief review of nearest neighbor algorithm for learning and classification. In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)* (pp. 1255-1260). IEEE.

Wang, H. (2005). Flexible flow shop scheduling: optimum, heuristics and artificial intelligence solutions. *Expert systems*, *22*(2), 78-85.

Weyl, H. (1950). The theory of groups and quantum mechanics. Courier Corporation.

Williams, H. P. (2013). Model building in mathematical programming. John Wiley & Sons.

Zaied, A. N. H., Ismail, M. M., & Mohamed, S. S. (2021). Permutation flow shop scheduling problem with makespan criterion: literature review. *J. Theor. Appl. Inf. Technol*, *99*(4).

Zhang, M., Tan, Y., Zhu, J., Chen, Y., & Chen, Z. (2020). A competitive and cooperative Migrating Birds Optimization algorithm for vary-sized batch splitting scheduling problem of flexible Job-Shop with setup time. *Simulation Modelling Practice and Theory*, *100*, 102065.

Zheng, A., & Casari, A. (2018). *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.".

Zribi, N., El Kamel, A., & Borne, P. (2008). Minimizing the makespan for the MPM job-shop with availability constraints. *International Journal of Production Economics*, *112*(1), 151-160.