POLTECNICO DI TORINO

Corso di Laurea in Engineering and Management

Tesi di Laurea

Scheduling a single batching machine with incompatibility and precedence constraints

Relatore
prof. Fabio Salassa

Candidata
Elena Rener
matricola: 251511

Anno accademico 2018-2019

Abstract

This thesis investigate the scheduling on a single crane performing dual command-cycles of storage and retrieval requests in an automated storage and retrieval system. The system is assumed to operate online and heuristic algorithms are developed, able to solve the problem in a short amount of time. As the warehouse is usually connected to other manufacturing process, jobs have to be processed on time and are subject to penalties if a delay on the whole production process occurs; thus, the maximum lateness is considered as objective function to be minimized. Jobs to be processed are considered as components or items of a production process and therefore subjected to precedence constraints, i.e. the requests can refer at the same item and the retrieval and the storage have to be performed in a logical sequence. In addition, the crane is assumed to process at most one storage and one retrieval request at a time, thus the existence of incompatibilities between jobs of the same type. Two solving heuristic algorithms for this specific problem based on different approaches are presented and evaluated.

Acknowledgements

Contents

List of Tables

List of Figures

Chapter 1

Introduction

At the core of this work is a scheduling problem coming from the Automated Storage Retrieval System (AS/RS). Warehouses represents a part of the production or distribution process and their management affects the system's overall performances. AS/RS are largely adopted, because they can improve performances compared with non automated warehouses, reducing inventory and work force costs, increasing the throughput and optimizing operations [Roodbergen and Vis (2009)]. It is particularly important in just-in-time environments, where the management of the inventory has to be efficient and reliable[1], which a fully automated system can better fulfill. Still, the improvement of performances has to be worth the high investment needed and every factor influencing the value gained has to be carefully treated. The design in the first place but also other features related to the system configuration - e.g. racks size, number of cranes, number and position of input/output (IO) points - have an impact on many performance measures such as throughput and total travel time. Besides system design and configuration features, also management and control decisions such as the way cranes travel times are modelled, items are disposed on the shells and the ordering inventory requests are processed influence system's performances [Gagliardi et al. (2011), Randhawa and Shroff (1995), Manda and Palekar (1997)]. Every dimension has to be adapted to the performances needed in the company and other specific needs of the surrounding system - such as production facilities connected to the warehouse.

[1]Womack et al. (1994) first defined basic just-in-time concepts to reduce waste in processes; among them, the need to reduce inventory level that requires as a consequence a better management of the warehouse in order to fulfill all external requirements.

The design and management of such a system is not easy at all. First, because of the dynamism of production and distribution processes that often requires high flexibility and adaptability to external changes and secondly, because of the uncertainty deriving from the variability of requests. Demands are hardly predictable, the availability of components depends on the reliability of suppliers, production processes depend on many different variables and availability on time of different resources. This results in a set of constraints that restricts the freedom in the design phase, which as mentioned above - depends on many connected decisions that have to be kept in consideration as a whole. Gagliardi et al. (2011) and Roodbergen and Vis (2009) present a review of all main assumptions to be considered and of past studies related to the subject.

Within this framework, we consider here the problem of scheduling storage and retrieval requests in order to minimize the maximum lateness of items on the production phase. This decision problem affects the total travel time of cranes and its related costs and the reliability of the system in terms of processing tasks on time and returning items to the connected production or distribution system when needed and possibly not late. It is assumed that the system has one crane performing dual command cycles, being able to first storage one item and then retrieving another one without going back to the I/O point. This means that a storage and a retrieval request can be batched together. The problem can be summarized as a scheduling of jobs with incompatibilities and precedence constraints on a single batching machine - $1|batch(b), prec, incomp|L_{max}$ with Graham's three field notation [Graham et al. (1979)]. Incompatibilities derive from the fact that two storage or two retrieval requests cannot be batched together and precedence constraints relate to the case of dealing with the same physical item, that has first to be stored and then retrieved.

Storage and retrieval requests arrive in most cases in a continuous flow, storages coming from deliveries of external/internal suppliers and retrievals from the demand of external/internal customers. As such, there are two possible ways of proceeding: One could either re-schedule all requests (tasks) a new, whenever a new request is received (*dynamic sequencing*) or select each time a set of tasks, schedule them and then proceed with the next set (*block sequencing*) [Han et al. (1987)]. The performance of both approaches differs according to the situation but the second is simpler and more intuitive and it is the one considered here.

The same problem is presented and exactly solved by Emde et al. (2019), where the authors use a logic-based Benders decomposition (B&BC) and show it outperforms in terms of computational time previous solution methods [Cabo et al. (2015)]. The algorithm finds most of optimal values for instances with 20 and 100 jobs in the time range of few seconds and many optimal values for instances with 200 jobs within half an hour. Being an exact method, it can provide optimal values for relatively small problems but cannot address the same problem when growing in size within a useful time period. The problem, being a generalization of the problem considered by Brucker et al. (1998) - scheduling of n jobs on a batching machine to minimize due date based scheduling criteria with batch restricted size - is in fact known to be NP-hard. In other words, no algorithm is known that can optimally solve the problem in polynomial time; instead, the time to solve the problem increases exponentially with respect to the number of jobs considered, becoming soon unfeasible. The aim of the work is therefore to analyze performances of some heuristics thought for the specific problem in comparison with the B&BC algorithm considering a real time system, where the responses have to be fast, let's say within a minute. Heuristic methods do not guarantee optimality but can still provide some good and useful solutions for industry purposes and improve the solution quality with respect to some other simpler scheduling rule as the largely adopted First In First Out (FIFO) or the Earliest Due Date (EDD) rule.

The reminder of the work is structured as follows: in chapter 2 it is presented a general background on scheduling problem and solving methods with definitions, notation, application frameworks and examples from the literature. In part II the specific problem coming from the context of automated storage and retrieval systems is addressed starting with chapter 3 where main issues is such sytems modelling are mentioned; in chapter 4 the problem is described in detail and a Mixed Integer Program formulation is presented; in chapter 5 we have the description of proposed heuristics to solve the problem; in chapter 6 results from the evaluation tests are discussed; finally, conclusions are summarized in the last chapter to review the work done and present possible future improvements.

Chapter 2

Scheduling problems

Dealing with scheduling problems is a common issue for most industries, from manufacturing to service providers [Pinedo (2016)]. The general idea behind the scheduling process is to assign, in the best possible way, resources to several tasks with respect to the firm's processes needs and objectives. Applications are many and various; objectives can be easily achievable or as difficult as reaching the limit of computational feasibilty.

Scheduling problems often lead to combinatorial optimization problems (COP) that are hard to be solved. The trade-off between the achievement of the optimal solution and the necessity to obtain solutions in a feasible time is at the core of scheduling. Problems solvable by polynomially growing time algorithms are distinguished from those solvable by exponentially growing time algorithms, the first being "easy" problems and the second "difficult".

In the AS/RS context, the system is required to accomplish all the storage and retrieval requests in such a way as to satisfy on time, with respect to some due dates, the retrieval of the parts needed. The resources available to process tasks are one or multiple cranes and the objective is to assign a sequence of items to each crane in such a way that the overall performances of the AS/RS would satisfy the best the requests of the manufacturing process requiring the items.

There is a broad variety of frameworks in which scheduling is applied. The objectives can vary a lot depending on which framework is taken into account. In the next sections, a brief overview of the most common deterministic scheduling models is presented, followed by an overview on how algorithm's

complexity is computed in order to understand the main issue around finding the optimal schedules for most problems due to computational capacity limits and next, some traditionally used scheduling rules and some other solutions found in the literature. In the next chapter, the focus will be on the use of heuristic algorithms to solve scheduling problems.

2.1 Scheduling models

Given a collection of jobs requiring processing in a certain machine environment, the issue in a scheduling problem is to sequence these jobs and assign them to certain machines, subject to given constraints, in such a way that one or more performance criteria are optimized. The objectives can be many and various, depending on the production environment. Sometimes, the objective may be the minimization of the completion time of the last task - when the interest is posed on the duration of the whole process - and another may be the minimization of the number of tasks completed after their respective due dates - when emphasizing the need of having jobs ready on time. We now first define the notation used to point out all different environments.

2.1.1 Notation

In deterministic scheduling problems we have a finite number of jobs and machines, each of them associated to some data describing their characteristics that we assume being deterministic. First of all, we have a number of jobs denoted by n and a number of machines denoted by m. A processing step or operation of job j on machine i is described by the pair (i, j), where we use the usual notation where the subscript j refers to a job while the subscript i refers to a machine. Each job j can be associated to some additional data and those are:

- the processing time p_j as the time requested to complete the process on the job j;
- the due date d_j as the time where job j is expected to be finished, otherwise a penalty is applied;
- the release time r_j as the earliest date at which job j can be processed;
- the weight w_j as the relative importance of that job.

Depending on the final schedule, each job has some other related information as:

- completion time C_j as the time where job j will be completed on the last machine where it needs to be processed;
- lateness L_j computed as C_j - d_j as the measure of how late is job j;
- tardiness T_j computed as the maximum between L_j and 0, s.t. tardiness is never negative and is positive if job j is late;
- unit penalty U_j that takes the value 1 if job j is late, 0 otherwise.

2.1.2 Scheduling frameworks

A scheduling problem can be generally described by a triplet $\alpha|\beta|\gamma$ following the notation as first introduced by Graham et al. (1979). The α field describes the machine environment - such as single machine, m parallel machines, flow shop - and contains just one entry. The β field provides details of processing characteristics and constraints - such as precedence constraints, preemptions, batch processing - and may contain no entry at all, a single entry, or multiple entries. The γ field describes the objective to be minimized and often contains a single entry. Some of them are: the makespan C_{max} as the completion time of the last job being processed; the maximum lateness L_{max} as the maximum of the lateness of all jobs; the total weighted tardiness ($\sum w_j T_j$) as a cost function that somehow measure the total cost of having late jobs. As example the problem investigated in this work is taken. In the notation described above it is denoted by $1|batch(2), prec, incomp|L_{\max}$ meaning that there is a single batching machine that can process 2 jobs at a time, that jobs have some incompatibilities and are subject to precedence constraints and that the function to be minimized is the maximum lateness.

Each combination of some of the above mentioned characteristics describes a scheduling framework with different parameters and objectives. Once the model is defined, there will be the need for some scheduling rules to be applied to the input tasks. One possible way would be of course to try out all possible combinations of tasks and to choose the best one according to the objective function, i.e. to solve the problem through complete enumeration. However, this would usually lead to an exponentially increasing number of computations that cannot be afforded in the case of an increasing number

of tasks. Therefore, considerable efforts have been made in the literature to find some solving algorithms that can operate polynomially increasing time while still achieving the best solution. In the next section, the complexity classes of solving algorithms are presented in detail.

2.2 Algorithm Complexity

Computational complexity serves as a mean to approximately indicate the number of elementary operations contained in an algorithm. It is used in computer science to classify algorithms according to how their running time grow as the input size grows. If an algorithm performs one elementary operations, one for all n elements, its running time is said to be of the order of n. n defines the size of the problem and by scheduing algorithms, it is usually identified as the number of elements to be scheduled. This is an approximation since the number of elementary operations also depends on how the solution is represented. For instance, a schedule can be represented as a sequence of objects but also as a binary vector as long as the amount of all possible elements containing a 1 or a 0 in position i whether element i is included in the solution or left apart. However, the approximation to the number of elements is generally accepted and useful to evaluate computing times.

In general, the *bigO* notation [Bachmann (1894) and Landau (1909)] is used, where just the fastest-growing term is represented. Formally, let f be a real or complex valued function and g a real valued function, both defined on some unbounded subset of the real positive numbers, such that $g(x)$ is strictly positive for all large enough values of x. One writes

$$f(x) = O(g(x)) \text{ as } x \to \infty \tag{2.1}$$

if and only if for all sufficiently large values of x, the absolute value of $f(x)$ is at most a positive constant multiple of $g(x)$. That is, $f(x) = O(g(x))$ if and only if there exists a positive real number M and a real number x_0 such that

$$|f(x)| \leq \ Mg(x) \text{ for all } x \geq x_0 \tag{2.2}$$

or simply

$$f(x) = O(g(x)). \tag{2.3}$$

For instance, in the case of an algorithm performing $n^2 + n$ elementary operations, it is said to perform in a time $O(n^2)$, meaning that the time required grows approximately by the square when increasing the number of elements. A problem with complexity $O(n^x)$, $x \in N$ is told to have a polynomially increasing time, whether a problem with complexity $O(2^n)$ or $O(n!)$ is told to have an exponentially increasing time by the size of job elements. It is of uttermost importance to understand the computational complexity because the computational feasibility of an algorithm's solution depends on it.

For most combinatorial optimization problems such as many scheduling problems - and also our problem - no algorithm with polynomially increasing time is known. Many of those problems are known to be NP-hard. Modern computational capability cannot afford an exhaustive search into whether other kinds of problems - solvable in polynomial time - can easily be solved in an optimal way. Moreover, the achieved quality of the solution can vary significantly on a case-by-case basis.

Complexity classes In computational complexity theory, the following main classes of problems are distinguished: problems solvable in polynomial time (P), in a non-deterministic polynomial time (NP)[1], in exponential time (Exp) and finite time (R). Besides these categories, there are those problems that cannot be solved in finite time.

In particular, the set NP differs from P in that it includes such problems that could be solved in polynomial time via a "lucky" algorithm that could choose the right path at each step of the decision tree. In other words, once a solution is obtained, this can be verified in polynomial time but not the same to find it.

P is clearly a subset of NP but it is not clear if $P \neq NP$ and it is a major unsolved problem in computer science. Nonetheless, it is meaningful to define the NP class because it computationally differs from P.

[1]The statement of P versus NP problems was introduced by Cook (1971) but the proof of $P \neq NP$ is still to be found and it remains a major unsolved problem in computer science. However, most scientists agree that it probably holds and we assume here the same.

Reduction Reduction is used to compare different problems in terms of complexity. A problem C is told to *reduces to* a problem C' if C is a special case of C' and this can be written as $C \propto C'$. Being C a special case of C', if we know a polynomial algorithm that solves C', then we can say that this works also for C, i.e.

$$\text{if } C' \propto C$$

$$\text{and } C' \in P,$$

$$\text{then } C \in P.$$

Reduction is very useful to compare problems and many researches focused on finding relations between problems using reduction. An example presented by Pinedo (2016) follows.

Example. We have two very well known problems: the knapsack problem (C) and the partition problem (C').

In the knapsack problem, we deal with n items. Each item j is associated to a size p_j and a benefit w_j and we want to fill a knapsack of total size d, such as to maximize the total profit z of items contained in the knapsack. Every solution can be represented as a binary vector with length n and a 1 in position j if item j is included in the knapsack and a 0 if not. This is equivalent to the problem of minimizing of the total weighted unit penalty ($U_j = 1$ if job comes after d, i.e. if it exceeds the knapsack size) of a set of jobs with a common due date d.

In the partition problem, we have the same number n of positive integers $a_j,\ j = 1, ..., n$. We want to partition them in two disjoint subsets $S_i,\ i = 1,2$, such that for each subset the sum of contained integers is the same and equal to b, i.e.

$$\sum_{j \in S_i} a_j = b.$$

By taking specific values for the knapsack problem, we see that it reduces to the partition problem; that is, if

$$p_j, w_j = a_j$$

$$\text{and } d, z = 1/2 \sum_{j=1}^{t} a_j = b,$$

$$\text{then } C' \propto C$$

NP-hard problems A special subset of NP problems are those NP-complete. They are the most difficult problems in NP. Formally, a problem is NP-complete if any other problem in NP polynomially reduces to it. The fact is, that if a polynomial algorithm would be found for a NP-complete problem, then this would mean that all of them could be solved in polynomial time.

NP-complete problems define a border line around the NP set. Beyond this, we have the so-called NP-hard problems.

The concept of NP-hardness refers to all problems that are at least as hard to solve as every problem in NP. Some of them can be solved by a polynomial algorithm for a certain solution encoding; these are called NP-hard in the ordinary sense or simply NP-hard. Some other cannot be solved in polynomial time in any encoding; these are told to be strongly NP-hard.

Exploring exhaustively all combinations of the components of NP-hard or strongly NP-hard problems often leads to computational times increasing exponentially in respect of the number of components. By real problems, this could easily mean days or years of computation and it is of course unfeasible to wait such a long time and the optimal solution may remain unknown.

Our problem is a specific case of the sequencing of jobs on a single batching machine with restricted batch size (BMRS problem). As this has been shown by Brucker et al. (1998) to be NP-hard, the problem considered is also contained in this category. Hence, the need of some heuristic algorithms able to solve the problem in a short amount of time but through the acceptance of solutions that are just nearly-optimal, in order to prioritize the finding of some feasible solution in limited time instead of the best one. Heuristics have found large use in all optimization problems indeed to avoid the non feasibility of many such problems and to come out with one - at least - feasible solution.

In the following section, the most common approaches to solve them - some leading to an optimal solution and some other not - are presented.

2.3 Solution approaches

As shown in the previous sections, every scheduling problem has to be approached specifically, depending on the framework. Every change in the problem design can affect the search procedures for a solution; In any case, the first decision to be made when approaching a scheduling problem is whether

to use an exact method - usually just possible for small sized problems - or a heuristic method. Among exact methods, apart from exact enumeration, we distinguish mathematical, dynamic and constraint programming methods, Branch & Bound procedures and others. If the problem has certain characteristics, also some simple dispatching rules can be enough to determine some good - or optimal in very specific cases - solutions.

2.3.1 Dispatching rules

Dispatching rules are the simplest and most intuitive method to approach a problem. They are rather simple ordering rules, whose aim is to prioritize tasks on machines according to job-related data, e.g. the processing time, or machine-related data, e.g. jobs queue at each machine. Dispatching rules are based on some logical intuition, e.g. nearest due date ordering when trying to minimize jobs lateness, which is a due date-related function. In some cases, they lead to optimal solutions (if the problem belongs to class "P") but of course - as they are purely greedy algorithms - they cannot be applied to complex optimization problem hoping in very well performances.

However, dispatching rules are broadly used because they are the simplest rules that can be implemented and work usually very fast. Thus, also in the cases in which they do not lead to good solutions, they can still provide an initial basis from which more satisfying solutions can be developed by means of more complex methods. Also, various or combined ordering rules are used as steps of complexer algorithms. Later, an application of dispatching rules to evaluate differences in problem instances will be also mentioned.

Clearly, a dispatching rule can be also used as simple standardized ordering method without any attempt of optimization. The widely adopted First Come First Served rule (FCFS) order jobs according to their arrival time. The Service in Random Order (SIRO) establishes as prioritizing rule a simple random ordering. Efficiency of these rules is often poor and an attempt of some kind of optimization is in most cases worth the effort.

2.3.2 Exact methods

Many efforts were made by researchers to develop mathematics and logic-based algorithms to exactly solve problems reducing number of computations required by a complete enumeration. Complete enumeration requires to list

all possible combinations of components of a problem. In some combinatorial problems, the ordering may be considered, in some other may be not. In any case, it is not difficult to see that the number of combinations grows exponentially as the number of components increases. For instance, to schedule n items, we should consider $n!$ different solutions and choose the best one. By 3 items, this requires considering 6 solutions, by 10 items, 3,628,800 solutions and by 100 items, to a number of solutions that has 157 cifers. Exact methods try to find some logical way to reduce the number of solutions to be considered.

Optimization problems are usually formulated as Linear Programs (LP) - otherwise called Integer Programs (IP) or Mixed-Integer Programs (MIP) if respectively all or some of the variables are integers. The formulation as LP, IP or MIP simply gives the mathematical formulation for a problem with an objective function to be minimized and some constraints to be fulfilled.

A formulation of a LP usually looks like the following:

$$\text{minimize } c_1x_1 + c_2x_2 + ... + c_nx_n$$

subject to

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n \leq b_1$$

$$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n \leq b_2$$

$$...$$

$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n \leq b_m$$

$$x_j \geq 0 \text{ for } j = 1, ..., n.$$

or simply, using the matrix form

$$\text{minimize } \overline{cx}$$

subject to

$$\mathbf{A}\overline{x} \leq \overline{b}$$

$$\overline{x} \geq 0$$

A first option is to use some available softwares that solve the MIP formulations of the problem. Broadly used is the commercial IBM's software CPLEX or the open source SYMPHONY.

There are many procedures that systematically solve some classes of problems in a more sophisticated way. For Linear Programs the most common methods are the simplex methods and the interior point methods; for the IP the most common are cutting plane (polyhedral) techniques, branch-and-bound and branch-and-price techniques.

Branch and bound Branch and bound techniques consider a partitioning of the solution space in a branching tree and for each part, i.e. at each node, compute a lower bound that gives an idea of the value of the best reachable solution in that branch. By comparing the bounds in each region, it cuts off the branches that surely do not contain the best solutions. For instance, if in a branch the lower bound is higher than a solution already found, then this branch can be diregarded as every solutions in it will be worse than the already found solution. The lower bound is usually an approximation of the best achievable solution. How it is computed, depends on the specific problem. A lower bound is said to be *tight*, the more it is close to the optimal solution. The tighter the lower bound, the more branches can be cut off and the solution found in less time.

A successfully applied version of the branch-and-bound is the branch-and-cut, where some cutting-plane techniques are applied at each node to generate stronger lower bounds. The cutting-plane techniques basically generate additional constraints that are injected in the original problem.

Branch and price combines branch and bound with the pricing problem that comes from the theory of linear programming, where variables are priced with a so-called *reduced cost*. The pricing optimization problem is then used to compute lower bounds.

Dynamic Programming Dynamic programming is used to solve a problem by recursively solving all subproblems, starting from the smallest up to the original problem. It can be applied if the main problem can be decomposed in smaller and identical subproblems.

Every dynamic programming procedure is composed by an initial condition, a recursive function and the optimal value function.

For instance, for the problem $1||\sum_i T_i$ with n elements, we first consider all subproblems made of one element and we choose element k such that

$$f(k) = \max(p_k - d_k,0),\ \text{k=1,...,n.}$$

The initial condition is set and the recursive procedure starts. For every subset S, we can compute the total tardiness as

$$f(S) = \min_{i \in S}\{f(S - \{i\}) + max[p_i - d_i,0]\}.$$

The procedure stops and the optimal value function for the problem is found when $S = 1, ...n$, i.e. it contains all elements.

2.3.3 Heuristic methods

Given the whole solution space of a problem, i.e. all feasible combinations of tasks on machines, the core point of heuristic methods is exploring this space and then just searching in the most promising regions without knowing where the real optimal solution lies and if it can even be reached.

As they often represent the only way to reach a computationally feasible solution, they are extensively applied to a large number of problems and there are a lot of examples of heuristics that were successfully applied in various fields - e.g. genetic algorithms in machine learning [Goldberg and Holland (1998)] or the tabu search in artificial neural networks optimization [Sexton et al. (2011)].

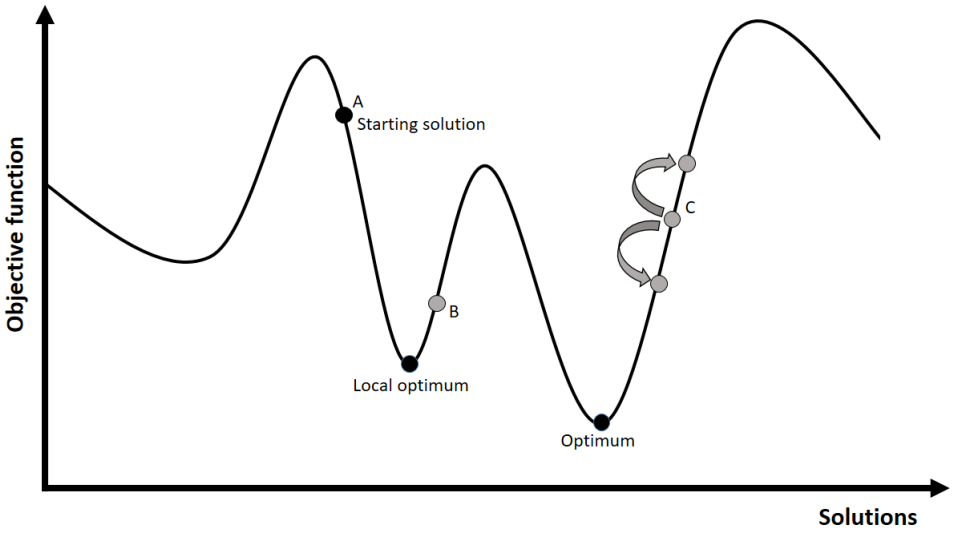

Figure 2.1. Intensification and diversification through the solution space

Definitions The word *heuristic* comes from the Ancient Greek and literally means "find", "discover". The term heuristic methods, or simply heuristics, was first introduced by Foulds (1983). Generally, it refers to the fact that a heuristic method simply tries to achieve a good solution in an easy and intuitive way without any proof of optimality. On the contrary, exact methods systematically search through the whole solution space, until a provable optimal solution is found. An exact method always finds the best solution, however long it does take; on the contrary, heuristics take advantage of some problem dependent properties and find some approximately good solutions in a faster way.

In practice, heuristic methods are often preferred to exact ones as in many cases they are the only possibility to reach a solution in a limited time.

In general, heuristic search principles can be defined by some simple priority rules or some common sense-based strategy built upon known information.

The definition of *metaheuristic* was instead introduced later by Glover (1986) to define a high-level strategy which builds upon a particular basic search principle[2]. The main underlying concept is to realize a balance between *intensification*, i.e. the exploitation of a specific region of the search space, and *diversification*, i.e. the exploration of the whole solution space. In other words, a good metaheuristic method should both explore different regions of the solution space to broaden the view and find where good solutions could lie and then focusing in finding the best solution in each region.

As an example, let's suppose to have a problem with a certain solution space as represented in Fig.2.1, where the x-axes represents all possible solutions and the y-axis the value of the objective function produced by each of them. A metaheuristic method can start by choosing - randomly or with some simple basic heuristics - a *starting solution* A. To broaden the view on the solution space, some other different and distant solutions will be chosen, let's say B and C and after that, the decision would be to better look at the region around the solutions found, as done by solution C. In such a way, we both explore different regions of the solution space by still analyzing them deeply.

A very common search procedure is the local search - being part of the search methods using solution modification explained in the next section -

[2]Metaheuristics Network Website URL: http://www.metaheuristics.org

that exactly implements the intensified search in the region of a solution. For instance, starting from solution C, it would look at its so called *neighbours* and choose the one with the lowest objective function value and repeat the procedure until the achievement of the optimum. It is easy to see that by repeating the search starting from solution A or B, the search would stay stuck at the *local optimum* without possibly never reaching the global optimum. This is why diversification plays an equally important role as intensification.

In a few words, metaheuristics define general and problem-independent frameworks that help to efficiently apply a heuristic search throughout the whole solution space.

The main issue by heuristic methods lies on the fact that they are often strictly problem-dependent in terms of performances, i.e. by changing some small parameters, the quality of solutions could worsen a lot. This is why metaheuristic methods are much more preferable in many cases: they are a sort of high-level strategy that "control" the performances of a particular underlying heuristic method and can be easily adapted to many different problems.

However, the distinction between heuristics and metaheuristics is often neglected and it is common to refer to both as heuristics.

Methods Classification The definition of heuristic is very general and simply refers to the fact of being a non-exact method but the underlying search principles can be summarized together and grouped in some classes as shown in Fig.2.2, where a general taxonomy of all solution methods is represented. After the first distinction between exact and heuristic methods, the latter are themselves split in constructive and search heuristics, the first building a unique solution, the second starting from a solution and exploring its "nearest" solutions. Considering how the nearest solutions are built, search heuristics distinguish then the search by repeated solution construction, modification or recombination. Here, an overview on how they generally work.

Constructive heuristics try to construct one single solution with the best possible quality by selecting promising solution elements.

Search heuristics conceptually differ from the constructive heuristics, because they examine many different solutions of the solution space in order

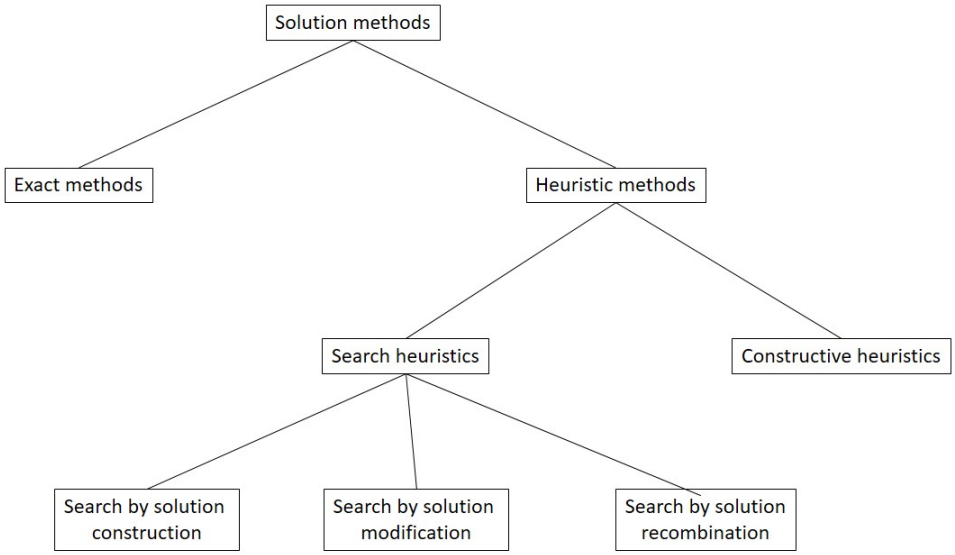

Figure 2.2. Taxonomy of solution methods [Zäpfel and Braune (2010)]

to find the best possible one. Depending on how solutions are generated, we have three different categories, as follows.

- *Search by solution construction*, it implies a repeated solution construction by slightly changing added elements, e.g. by partially randomizing the adding process or by keeping track of elements properties.

- *Search by solution modification*, it generates solutions starting from an initial one - previously generated - by for example exchanging or inserting some elements. Under this principle lies the concept of local search, defined as the search through the whole neighbourhood of a starting solution - where the neighbourhood contains all solutions that differs from the first one by a small modification gained in a systematic way - in order to find the nearest local minimum - supposing a minimization problem. This kind of search guarantees the achievement of the best solution given a specific region of the solution space.

- *Search by solution recombination*, it takes some from a previously generated pool of solutions and recombine their elements in order to create a new "generation" of solutions.

In all three cases, the method generates some final solutions and chooses the best one.

Upon these general search principles, some metaheuristic methods can be built by designing a framework that control and addresses in the right way the underlying heuristic algorithms. In fact, there is the need of an entity that can preserve an overlook on the whole process performances and find a way of not being stuck in the first apparently good solution and instead keeping exploring other regions of the solution space, fact above referred as creating a balance between intensification and diversification. In the next section, we look at some of the most popular metaheuristics that were successfully applied to various optimization problems.

Metaheuristics Here, it is presented a list of the most popular metaheuristics used in the literature emphasizing how intensification and diversification are performed in each of them.

Zäpfel and Braune (2010) exhaustively discuss all principles underlying metaheuristic methods and present in more detail all following methods.

The *Greedy Randomized Adaptive Search Procedure (GRASP)* is based on a solution construction search and the idea behind it, is to insert some randomness while building the solution in order to bring some kind of diversification in the process. On each partially random solution, a local search is performed to reach the nearest local optimum.

The *Ant Colony Optimization (ACO)* is another metaheuristics based on solution construction. The idea is to explore as many different paths between elements as possible, leaving track on each path of the quality of solutions found and by repeating the process many times. Thereby it will always choose the path depending on how good the "traces" are by keeping exploring many different paths. ACO will therefore slowly converge to the best path thanks to the progressive accumulation of positive tracks.

The *Tabu Search* algorithm is based on solution modification. The idea behind it lies in the fact that by performing a simple local search, the probability of being stuck around a local minimum is very high. Therefore the need arises of keeping track of past solutions in order to avoid them and keeping exploring other regions of the space also if this implies a worsening

of the solution. A so-called tabu list memorizes past solutions or elements contained in them and they are simply avoided if found in the neighbourhood during the local search.

The *Treshold Accepting* is another metaheuristics based on solution modification. The algorithm generates a neighbour from a starting solution and accepts it, if it does not exceed a previously defined threshold. The threshold is usually reduced at every iteration, so that even if at the beginning most of the neighbours are explored only the best one will be accepted at the end.

The *Simulated Annealing* works similarly to the Threshold Accepting algorithm; it generates a neighbour and accepts it with a certain probability that depends on both the distance from the previous solution and a so-called temperature that is reduced at each iteration.

The *Genetic Algorithm* works based on solution recombination. Starting from a pool of previously generated solutions, it - partially randomly and partially evaluating them - chooses two "parents" and recombine their elements by exchanging all elements that stay over some cross-over point. Sometimes some "child" solutions are mutated, in order to bring more diversification. The idea is that by repeating the process many and many times, only the best "features" of the solutions will survive and the best solutions will be found.

The *Scatter Search* is similar to Genetic Algorithms but performs intensification and diversification in a more visible and systematic way. From a starting solution, it first generates some diversified solutions and intensifies them with a local search. Starting out from this, it builds a ”reference set” that contains both the best and the most diverse solutions; it recombines all pairs to create the new starting pool of solutions.

2.3.4 Hybrid methods

There are, apart from exact and heuristic methods, some other directions in which researches are focusing with more and more attention. A set is roughly identified and includes so-called *matheuristics*. Matheuristics are informally referred to as hybrid methods, where the hybridization refers to the combination of different methods to come up with a new approach that exploits advantages of each of them.

Matheuristics rose from some applications of metaheuristics by introducing elements of mathematical programming or more generally, exact approaches. Authors combined heuristic and exact methods in different ways. Della Croce et al. (2011) addressed the minimization of the total completion time in a 2-machine flow shop and combined a Recovery Beam Search method for a first heuristic search through the solution space and a neighbourhood search based on a MILP formulation to solve it with a commercial solver. Hansen and Mladenović (2005) developed a decomposition-based approach that mix exact and heuristic procedures built upon a Variable Neighbourhood Search metaheuristics to solve large instances of various problems.

Maniezzo et al. (2009) gives an overview on the state of the art of matheuristics classification and presents many example applications.

2.4 Examples from the literature

The literature about scheduling problems is really broad and the approaches are various. The simplest approaches refer to some simple ordering rules that prioritize tasks according to the objective function; some split jobs in subgroups and order them by different rules according to the subgroup; some others use dynamic programming to adapt continuously the objective function or some kind of logic reasoning in order to "cut" some paths that would not lead to better solutions. To stay in line with the present work, key literature mainly regarding the single machine environment and some in line with our specific problem are mentioned. We start with some dispatching rules that simply order jobs (tasks) by their attributes and focus then on some more problem-related strategies that could be of some interest for our specific problem of modelling an AS/RS.

The Weighted Shortest Processing Time (WSPT) rule orders jobs by non decreasing weighted processing time, i.e. w_j/p_j. It is shown to optimally solve (Pinedo, 1994) the minimization of the total weighted completion time $1||\sum w_{\mathrm{j}}C_{\mathrm{j}}$. The simpler variant derived from it is the Shortest Processing Time rule (SPT), optimally solving the minimization of the makespan $1||\sum C_{\mathrm{j}}$.

Lawler (1973) developed an algorithm to minimize the maximum cost (MMC) that optimally solves the problem $1|prec|h_{\max}$, where $\mathrm{h_{max}}$ is the maximum of a general due date related cost functions $h_{\mathrm{j}}(C_{\mathrm{j}})$, $j = 1, ..., n$

such as jobs lateness and tardiness. It is a backwards algorithm that sequence the jobs from last to first, always choosing next, from among the jobs which are currently available, i.e. jobs none of whose successors remain unchosen, a job with the latest possible deadline [Lawler (1973)]. There are n steps needed to schedule the n jobs. In each step at most n jobs have to be considered. The overall running time of the algorithm is therefore bounded by $O(n^2)$. The simpler variant is the Earliest Due Date (EDD) rule, ordering jobs by non decreasing due date that optimally solve the problem $1||L_{\max}$.

Johnson (1954) addressed the problem of scheduling jobs in a flow shop. Flow shops refer to those processes where a number of operations have to be done on all job. Every job goes through multiple machines set up in series and has to follow the same route. Jobs can have different processing times on different machines and we refer to the processing time of job j on machine i as p_{ij}. Buffers between machines are often considered to be unlimited if the products to be processed are very small; otherwise, a limit to buffer capacity has to be specified.

The so-called *Johnson's rule* finds an optimal schedule for the problem $F2||C_{max}$ that is, the minimization of the makespan in a flow shop with two machines. Jobs are first partitioned into two subset, the first containing all jobs having $p_{1j} < p_{2j}$ and the second all others. An optimal schedule is generated scheduling first jobs in the first subset ordered by increasing p_{1j} (SPT) followed by jobs of the second subset ordered by decreasing p_{2j} (LPT). Such a schedule is also referred to as an $SPT(1) - LPT(2)$ schedule.

This procedure - that leads to a polynomial algorithm - works only for a flow shop with two machines. On the contrary, it can be proved that increasing the number of machines the problem becomes strongly NP-hard [Pinedo(2016)].

Han et al. (1987) evaluated the travel-time savings that can be achieved by first pairing storage and retrieval requests using the Shortest-leg (SL) and Nearest-neighbour (NN) heuristics compared with the scheduling with the First Come, First Served (FCFS) rule being up to 60% of the time. The NN heuristic selects a pair $P = (s \in S, r \in R)$ with the minimum interleaving time from the non-empty set of available retrieval requests (R) and the non-empty set of available storage requests (S). The shortest-leg (SL) heuristic sequence retrievals and considers both the travel time to a storage location s and the interleaving time from s to a retrieval location r as one leg, and executes the pair with the shortest leg.

Possani (2001) defines a local search heuristics based on an exponentially sized neighbourhood that can be searched in polynomial time to minimize maximum lateness on a batching machine with restricted batch size (BMRS problem), i.e. to solve the problem $1|batch(b)|L_{\max}$. The procedure considers splitting the initial solution in two disjoint subsequences and then merging them by performing multiple insert moves. Given a split procedure, a feasible batching to minimize maximum lateness can be achieved in polynomial time with a dynamic programming algorithm.

Emde et al. (2019) developes a logic-based benders decomposition cut algorithm to exactly solve the specific problem addressed in this work. Authors decompose the problem into a master problem, where jobs are first assigned to batches and a slave problem, where those batches are sequenced with respect of all constraints minimizing the maximum lateness. In the slave problem, feasibility and optimality cuts are generated and injected into the master problem's branch and cut tree as lazy constraints, following the idea of the so-called branch and Benders cut approach. The master problem is formulated as an Integer Programming model and whenever it finds an integer solution, the slave problem is solved and new cuts injected until no more feasible or unfathomed solution remains.

2.5 Knowledge discovery techniques: a support for scheduling

Aligned with the idea of hybridization, there are research lines that combined the scheduling process with some methods coming from the field of knowledge discovery, i.e. the field that contains support methods ranging from statistical control charts to machine learning and artificial intelligence methods.

The reason lies in the fact that scheduling is the general process of sequencing tasks within a certain framework and this applies to such distant environments that the scheduling process can take the most various forms. Given the hypothesis that scheduling aims to optimize some kind of performance, with small changes in the framework some solving methods could return really poor performance if they are strictly problem specific. Especially simple heuristic procedures are often thought for specific problems and cannot tolerate big changes without loosing solution quality. On the other

side, complex methods that are able to solve problems well with more general conditions are harder to implement and are often computationally more expensive.

A support coming from data analysis and the understanding of their pattern was therefore introduced to improve solutions in complex and diversified environments and facilitate decision making. Harding et al. (2006) present a review of data mining techniques used as decision making support in the manufacturing industry from the beginning of 1990s. Ismail et al. (2009) review Artificial Intelligence techniques such as the knowledge based method, artificial neural networks, fuzzy logic, genetic algorithm and regression tree used in production planning and, particularly, in job shop scheduling and control.

2.5.1 Knowledge discovery

Knowledge discovery is the science of studying, analysing and finding information or patterns about data. When dealing with a huge amount of data that relate to the same flow of information as scheduling requests but also http packages in a network, environmental statistics or personal data about a community, it becomes important to find some common characteristics in order to extract some knowledge from the whole quantity of data. Among all features of a certain type of data, some are more relevant than others and the goal is to understand which ones are of some meaning and which ones can be, on the contrary, disregarded. The fact is that often the amount of data required is too big to be analysed at whole and the extraction of the sole important features is necessary or can still strongly reduce computational efforts.

Data mining is in general the process of discovering patterns and get information from a large quantity of data that involves methods built around Artificial Intelligence, Machine Learning, statistics and database systems. It is defined [Honghua Tan. (2012)] as the automatic or semi-automatic analysis of a large quantity of data to extract previously unknown interesting patterns, such as groups of data, records (cluster analysis), unusual relationships (anomaly detection) and dependencies (association rule mining). This step helps identifying the more meaningful features, possible classes of the data and serves as a basis to effectively use other knowledge discovery tools.

Data mining tasks are divided in predictive, i.e. that use some characteristics to predict future data, using regression, classification, decision tree, and descriptive, i.e. that describe task-relevant data in a concise way, using segmentation, clustering, association, outliers analysis, classification, characterization, clustering, discrimination. These techniques aim at constructing, through machine learning models, a knowledge function that can map together data set values and extracted information.

Let's suppose we want to monitor a network traffic in order to identify possible intruders. Each network package represents an instance of the data and is characterized by many features, such as content, sender, receiver, path, size, format and so on. After collecting a certain amount of data, we analyse them and extract some information that could for example be that to identify intruders it is enough to look at the sender and the path of the package. Based on this information, we build a model that classifies packages in desirable and undesirable in respect of the sender and path. In this way, we can train the computer to automatically extract information from a big quantity of data.

Data mining in scheduling

It was already mentioned that this methodology can also be applied to scheduling. There are past researches that consider solving scheduling problems using simple assignment/dispatching rules supported by knowledge discovery methods to both select the most appropriate rule and to discover new ones.

Sha and Liu (2005) studied the due-date assignment problem and developed a data mining approach for extracting knowledge on different due-date assignment rules. They considered dispatching rules as a predictor for due-date assignment rules and as a result, they obtained a decision tree that makes a decision by selecting a due-date assignment rule for a given dispatching rule, number of jobs in the system, processing time requirement of jobs and other parameters.

Li and Olafsson (2005) instead, proposed a datamining-based approach for discovering new dispatching rules. They similarly used a set of dispatching rules to extract knowledge depending on their performances. The final output is a decision tree that is referred to as a new dispatching rule and which is

later used to make a decision for releasing or not releasing a job to the machine based on attribute values such as processing time requirement and processing time difference. Geiger et al. (2006), and Geiger and Uzsoy (2006) approached the same problem of new dispatching rule discovery by combining simulation and GA techniques.

Metan et al. (2010) consider a job shop problem minimizing average tardiness and develop a model that selects dispatching rule through a decision tree extracting knowledge from data using simulation, data mining and statistical process control charts. In addition, the model updates the decision tree whenever the manufacturing conditions change. They define a number of attributes such as total remaining processing time, maximum queue length at time t, average remaining time until due-date, etc. that represent the general characteristics of the manufacturing system and its status in time. They identify attribute selection as a main issue because attributes affect the quality of the tree in the construction phase as well as in the decision phase concluding that increasing the number of attributes in the subset does not necessarily improve the quality of the decision tree and that the effect of each attribute in the subset on the performance of the generated decision tree also depends on the other attributes in the subset, i.e. they are correlated and their importance is hard to be determined. These techniques aim at constructing through machine learning models a knowledge function that can map together data set values and extracted information.

Chapter 3

Automated storage and retrieval systems

3.1 Automated storage and retrieval systems modelling

Automated storage and retrieval systems (AS/RS) are warehousing systems used in production and distribution systems for the storage and retrieval of products or components.

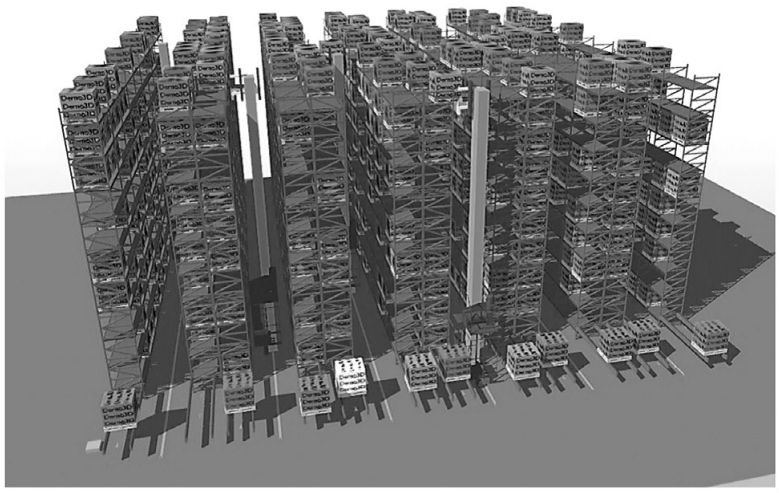

Figure 3.1. A unit-load AS/RS example [Gagliardi et al. (2011)]

Fig. 3.1 schematically shows how they are structured. An AS/RS is composed by some racks. Single or multiple cranes move along the racks storing and retrieving components that lie on the shells. Components are picked or returned to some I/O point, for instance, in front of the racks.

Their large adoption is mainly related to the savings in labour costs and floor space, increased reliability and reduced error rates but they need to be carefully managed to get all advantages and improve returns despite their high investment costs.

Gagliardi et al. (2011) and Roodbergen and Vis (2009) present two literature reviews about the topic over the past decades. They point out the main issues involved in AS/RS modelling, being system design and configuration, travel time estimation, storage assignment and request sequencing. System design and configuration deals with the determination of matters like the number and size of the racks, number of cranes, where they can move and other system-related decisions; travel time estimation poses the assumptions on how travel times are modelled and their restrictions; storage assignment relates on rules about where to store items; finally request sequencing is about scheduling tasks in order to optimize some measure of system performances.

System design and configuration An AS/RS must be specifically designed depending on the needs of the production or distribution system connected. The capacity is the first decision to be met but also the shape of the whole system, racks height, length and depth, number and capacity of the cranes, number and position of I/O points have to be establish. About cranes, there are a lot of possible configurations: number, capacity, degree of freedom. Usually in a fully automated AS/RS they are unit-load, meaning that components are batched in a pallet and the whole pallet is treated as a inseparable unit and cranes are subsequently distinguished in single, dual or multi-shuttle depending on the number of units they can contain. We distinguish aisle-captive or aisle-changing systems weather cranes can move through a single or multiple aisles; single command cycle or dual command cycles cranes weather they can complete a request at a time, every time returning to the I/O point or they can store an item and then retrieve another without first returning to the I/O point. A lot of studies covered some of these system configuration issues; for instance Randhawa and Shroff (1995) studied system performances through simulation based on storage racks and number and location of I/O points; Rosenblatt et al. (1993) studied system performances depending on racks size, number of cranes, number and length of aisles and number and location of I/O points both analytically and through

simulation; Chang and Wen (1997) analytically studied system performances based on storage racks.

Travel time estimation Travel time estimation influences the reliability of the AS/RS model and its evaluation quality. Many analytical models of travel times were studied in the literature; Bozer and White (1984) presented in their work some of these models and estimated with a statistical approach their expected travel times for single and dual command cycles in an AS/RS for a discrete rack and a continuous model and compared them with the MHI model originally presented by the AS/RS Product Section of The Material Handling Institute, Inc. (MHI). They also examined some different strategies in I/O and dwell-point location. These models can provide a general idea on the system average performances but they are based on very strict assumptions, such as independent cranes drives on both axes, that allow them to travel horizontally and vertically simultaneously (i.e. travel time follows a Chebyshev distance metric), physical capability to store any item at any time of all storage locations, possibility to neglect acceleration and deceleration of cranes and many others; therefore, simulation based models were often adopted and resulted closer to the reality. For instance, Schwarz et al. (1978) used a discrete rack simulation to extend the analytical models of Hausman et al. (1976) and Graves et al. (1977) relating storage assignment to travel-time performances.

Storage assignment It refers to policies adopted to assign items or class of items to the shells. Main rules are the dedicated storage location, the random storage location, the closest open location, full-turnover-based storage assignment and the class based storage assignment. Based on the rule items are respectively assigned to a fixed or random location, to the closest location with free space, to a location assigned based on item demand frequency or to the location assigned to the class the item belongs to. Gagliardi et al. (2010) and Hausman et al. (1976) studied for instance the effect on travel times of a turnover-based in comparison with the class-based and the pure random storage assignment policy. The results showed that a turnover-based policy improves the travel time performances in respect of the random storage location.

Request sequencing Request sequencing is actually a dynamic problem that has to deal with the continuous flow of new storage/retrieval requests.

Since this means continuously re-sequencing tasks and can be computationally and timely expensive, it is often preferable to use the so called block sequencing (Han et al., 1987) and sequencing one block of requests at a time. Request sequences can have a great impact on system performances. Linn and Xie (1993) concluded that prioritizing jobs with nearest due date can decrease 55%of the stockouts in a JIT environment. Han et al. (1987) studied the effect of a request sequencing based on first pairing storage and retrieval requests and then sequence them on the system performances, showing that the throughput can increase of 10-15% in comparison with the FCFS (First Come First Served) rule.

3.2 Problem description

The problem analyzed comes from the logistics and is originally motivated by the scheduling of requests on a single crane that moves in an automated storage/retrieval system (AS/RS). However, the model is applicable also to other systems dealing with other kind of batching machines. For instance, some applications of the model without incompatibility and precedence constraints were studied in relation of semiconductors burn-in operations by Lee et al. (1992) and Cabo et al. (2015).

Our logistic problem deals with a set of retrieval and storage requests that need to be scheduled on a single crane performing a dual command cycle.

For every request (job), some items have to be stored in or retrieved from the warehouse. The warehouse is composed by some racks and items are disposed on shells. A single crane moves all racks long both horizontally and vertically storing and retrieving items. In most automated system, the crane cannot extract a specific number of items from a pallet but works instead with unit-loads, meaning that if the production system requires 3 components and unit-loads are made of 10, the crane picks up the whole pallet of 10 items and the specific number of required components is selected by some employee at the shells or at the input/output (I/O) point. In practice, cranes neglects the exact number of components needed and every storage or retrieval request is associated to one single pallet. We speak of storage requests if the unit-load arrives at the I/O point and has to be brought to its storage location on the shells and of retrieval request if the items are requested at the I/O point and has to be picked up from the racks.

The crane performs a dual-command cycle, meaning that a unit-load can

be stored and another retrieved without needing the crane to return to the I/O point. The crane picks a unit to be stored from the I/O point, brings it to the shells, retrieves a unit required and transports it back to the I/O point. The crane is therefore referred to as a batching machine that can processe at most two compatible requests at a time.

Incompatibilities rise from the capacity of the crane that can contain one unit-load at a time. That is, just one storage and one retrieval request can be processed in the same dual-command cycle.

Items can be first stored and later retrieved. Precedence constraints are generated to fulfill the need of respecting the logical sequence of tasks on same physical items.

As the AS/RS is usually connected to a production system and furnishes it with required components, we want parts to be delivered possibly on time. An objective function that minimizes the maximum lateness of jobs is therefore used. The problem is described by Graham's three field notation as $1|batch(i), prec, incompatibility|L_{max}$ (see chapter 2).

Let's call J the set of requests (jobs) $J = 1, \ldots, n$. Each job is characterized by a processing time p_j which refers to the time needed to complete the storage/retrieval request and a due date d_j which represents the time where the job has to be completed, penalty a delay in the process.

Jobs can be incompatible to each other if they are of the same type (two storage or two retrieval requests) and cannot be processed by the crane in one cycle. Therefore we define the tuples containing incompatible jobs as $(j, j') \in I$, where I is the set which contains all of them.

Jobs are also subject to precedence relationships when referring to the same physical items that have to be, for instance, first stored and lately retrieved. These constraints are represented by the tuples $(j, j') \in F$, where j is the job that has to be processed before j' and F is the set that contains all precedence relationships.

The objective is to generate a schedule of batches, i.e. a partition of jobs into r subsets, to minimize the maximum lateness. Let's call B_{k} the *k-th* batch processed on the single machine and C the whole batch set. Each batch has a maximum size b=2; its processing time $P(B_{\mathrm{i}})$ is approximated with the longest processing time of the jobs contained in that batch and its completion time τ_{i} equals its processing time plus the sum of the completion times of batches previously scheduled, i.e. $\tau_{\mathrm{i'}} = \sum_{i'=1}^{i} P(B_{\mathrm{i}})$. The due date of a batch $D(B_i)$ equals the minimum due date of batched jobs. The maximum

lateness is computed as

$$L_{\max} = \max_{j \in J} \{\tau_{\pi(j)} - d_j | \tau_{\pi(j)} > d_j\} \tag{3.1}$$

where π_j is the position into the sequence of the batch job j is assigned to.

For the sake of simplicity, it is assumed that machine processing times are discrete, neglecting crane's acceleration, deceleration and its real technical capabilities, and that shell access time is much smaller than travelling time and can be therefore neglected.

Example. Let's suppose that we have six requests j $i = 1, ...,6$ to be scheduled having respectively a processing time p_i and a due date d_i as showed in Table 3.1. The set of incompatibilities is defined by $I = \{(1,4), (1,5), (4,5), (2,3), (2,6), (3,6)\}$ and the set of precedence constraints by $F = \{(1,3), (2,6)\}$.

j	1	2	3	4	5	6
p_j	3	5	8	1	11	7
d_j	5	6	10	13	16	17

Table 3.1. Example problem data

A feasible solution could be, for instance, a schedule made of the following three batches: $B_1 = \{1,2\}, B_2 = \{3,5\}, B_3 = \{4,6\}$ with a resulting maximum lateness of 10 (see Fig. 3.2). Instead of batching last two jobs, they could be processed separately and the resulting solution would be optimal and made of the following four batches: $B_1 = \{1,2\}, B_2 = \{3,5\}, B_3 = \{4, \backslash\}, B_4 = \{6, \backslash\}$ with a resulting maximum lateness of 7 (see Fig. 3.3).

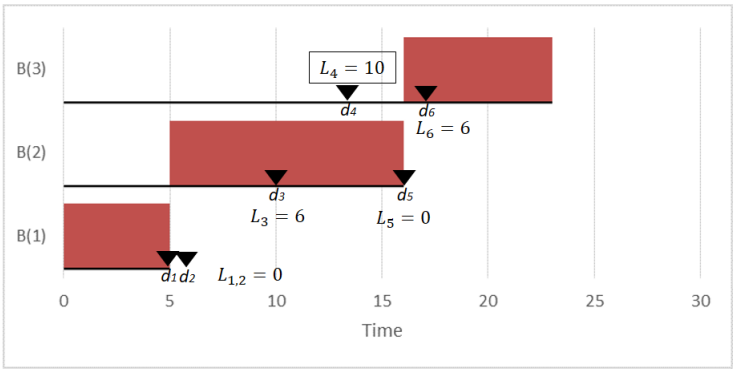

Figure 3.2. A feasible solution

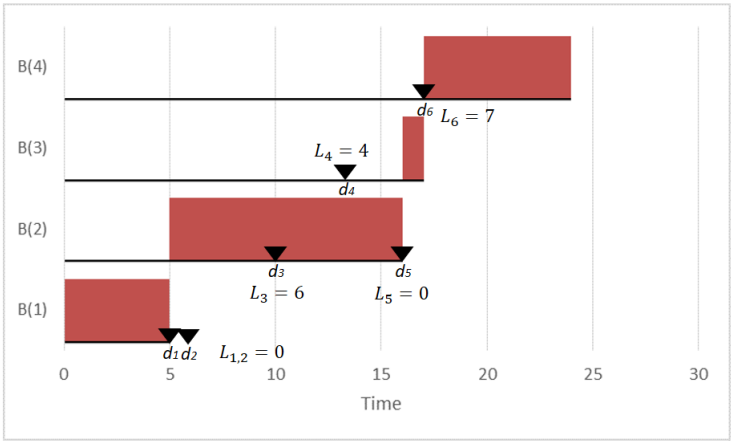

Figure 3.3. An optimal schedule

3.3 MIP model

With the data described in the previous section and some additional variables (see Table 3.2 and 3.3), a MIP programming model for the problem is

Notation	Description
J	set of jobs
C	set of batches
F	set of precedence constraints: each tuple (j, j') indicates that j has to be processed before j'
I	set of incompatibilities: each tuple (j, j') indicates that j cannot be processed with j'
p_j	processing time of job j
d_j	due date of job j
b	batch maximum size
M	large integer value

Table 3.2. Input data

Notation	Description
$x_{c,j}$	binary variable: 1, if job j is assigned to batch c; 0, otherwise
τ_c	completion time of batch c
P_c	processing time of batch c
L_{max}	maximum lateness

Table 3.3. Variables

presented as originally proposed by Emde et al. (2019).

Minimize L_{max}

subject to

$\sum_{c \in C} x_{c,j} = 1$	$\forall j \in J$	(1)
$\sum_{j \in C} x_{c,j} \leq b$	$\forall c \in C$	(2)
$P_c = \sum_{j \in J} p_j * x_{c,j}$	$\forall c \in C$	(3)
$\tau_1 = P_1$		(4)
$\tau_c = \tau_{c-1} + P_c$	$c \in C \setminus \{1\}$	(5)
$L_{max} \geq \tau_c - d_j - M \cdot (1 - x_{c,j})$	$\forall j \in J, c \in C$	(6)
$\sum_{c \in C} c \cdot x_{c,j} \leq \sum_{c \in C} c \cdot x_{c,j'} - 1$	$\forall (j, j') \in F$	(7)
$x_{c,j} + x_{c,j'} \leq 1$	$\forall c \in C, (j, j') \in I$	(8)
$x_{c,j} \in \{0,1\}$	$\forall j \in J, c \in C$	(9)
$L_{max} \geq 0$		(10)

First two constraints (1) and (2) ensure respectively that jobs are scheduled once and that batch size is not exceeded. Constraints (3) to (5) define processing and completion times of batches. Condition (6) ensures that the maximum lateness always exceed or equals lateness of all jobs. Conditions (7) and (8) take in account precedence constraints and incompatibilities. Condition (9) define the dummy variable representing the batch each job is assigned to and finally condition (10) impose that just latenesses greater or equal to zero are considered.

This MIP leads, as previously said, to a NP-hard problem. In addition, it contains the $big - M$ constraint (constraint (6)) that makes the MIP particularly difficult for solvers to solve it at the optimum. The value of the $big - M$ has to be sufficiently large to ensure the correct working of the model but the larger it is, the harder it is to solve the problem.

Chapter 4

Solution approaches

Keeping in mind the goal of searching for some online-solving algorithm, we propose in this chapter two heuristic approaches to find a feasible solution for the problem. They were selected as a result of testing of various simple and intuitive ordering rules as well as combinations of greedy approaches and neighbourhood search.

Dealing with simple heuristic methods, means also dealing with high uncertainity with regard of solution quality. In particular by greedy algorithms, it can happen that they return good solution for some instances of a problem and very bad for others. It is therefore important that a sufficient solution quality stability is guaranteed in order to show them to be somehow useful. Here, two heuristic based on different approaches are presented that were shown to maintain a certain average gap with respect to a lower bound. However, there are also some recent works that try to use the different behaviour of several greedy algorithms, e.g. dispatching rules, to differently schedule problem instances. This kind of research deals with knowledge discovery techniques, such as machine learning and artificial intelligence, and use them to support the scheduling process. With regard of this, at the end of this chapter also a possible application in this direction is mentioned.

Selected heuristics are respectively based on solution construction - one feasible solution is generated - and on a local search applied on a solution. The complexity is polynomial and quality of solutions are evaluated in computational tests presented in the next chapter.

Beside heuristics, we developed a procedure to compute a lower bound to compare our solutions with. The procedure is based on the consideration of a partial schedule of just storage or retrieval requests. By considering just one kind of jobs, we reduce our problem to the scheduling of a set of jobs

subject to precedence constraints with no batching minimizing the maximum lateness for which an algorithm exists that optimally solves it in $O(n^2)$.

The chapter is organized as follows. First, the procedure to compute the lower bound is presented; then, the selected heuristic algorithms are discussed in details; finally, some algorithm variations are presented and a possible application with knowledge discovery methods is presented for future developments.

4.1 Lower bound procedure

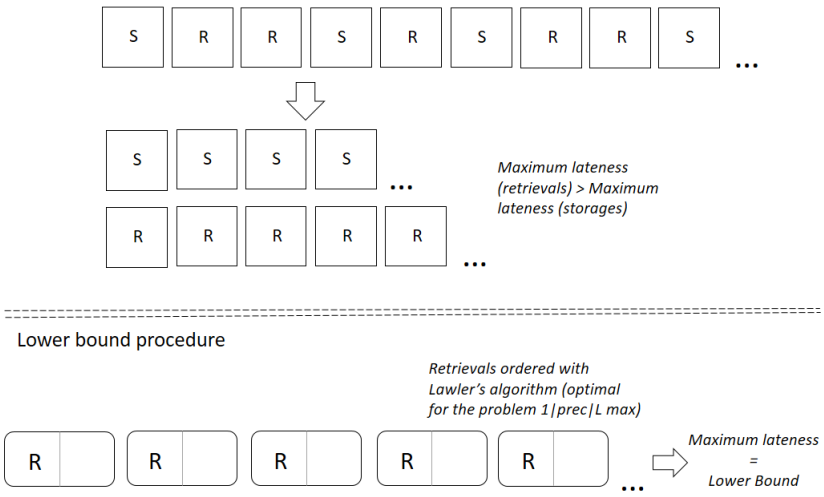

Figure 4.1. Lower Bound construction procedure

To compute a lower bound, we develop a scheduling procedure that considers just one type of request (job) and can therefore disregard batching. The procedure works as follows and it is illustrated in Fig. 4.1:

- we consider separately the set of storage and retrieval requests
- the two sets are scheduled following Lawler's algorithm that minimizes the maximum cost (MMC), optimal for the problem $1|prec|Lmax$
- two batch schedules are created, one with just storage and the other with just retrieval requests, respecting the MMC ordering and always

leaving empty the second place in batch (all requests of the same type are incompatible one to another)

- the maximum lateness for both batch schedules is computed and the largest is taken as lower bound.

Rationale. The final schedule of all requests set is composed by a sequence of batches, each filled by one or two compatible jobs. Let's first consider just one type of jobs - let's say retrieval requests - and schedule them on the batching machine. The resulting schedule would be a sequence of batches, each with one job inside and with a certain maximum lateness, depending on their ordering. By considering the other type of jobs - let's say storage requests, we would have different scenarios.

First, storages are less than retrievals and all of them can be batched together with retrievals without altering the maximum lateness, i.e. every storage request finds a place in a batch where the processing time of the current job is longer than its processing time and the due date larger or the difference in both is such not to increase the maximum lateness of the whole schedule. The maximum lateness is that of the first partial schedule.

Secondly, storages are less than retrievals but not every storage request finds place in a batch without increasing the maximum lateness. The maximum lateness increases with respect of the maximum lateness of the partial schedule.

If storage are more than retrieval requests, some storages would be batched separately in additional batches and the maximum latenes can whether stay the same or increase.

Therefore, we can say that the maximum lateness could increase but never decrease with respect of the maximum lateness of a partial schedule of one type of requests if the ordering of the partial schedule does not change.

To prove that the MMC ordering generates a lower bound, let's suppose that we find the optimal schedule and that we extract retrievals and they are not ordered by the MMC ordering. As MMC is optimal to schedule one type of request, the maximum lateness generated by the MMC ordering would always be less or equal than that generated by this partial schedule of the optimal schedule.

If we consider for the reasoning both retrieval and storage requests, than the procedure holds and the lower bound never overcomes the maximum lateness of the final schedule.

4.2 Heuristics

In order to keep computing time small, proposed algorithms give little weight to the basic principles that improves heuristic solutions in the form of meta-heuristic methods - only a basic local search is used and no diversification at all. As any greedy algorithm, the main critical aspects lies in the fact that there is little information about the final solution quality during the construction and the selection of items to be added to the solution can lead to some contradictions. In our case, it is difficult to evaluate if it is preferred batching together whenever possible compatible jobs - reducing completion time of following jobs and subsequently their lateness - or prioritizing the minimizing of the lateness of the current job at any stage of the solution construction, i.e. choosing to batch a job alone when a second compatible job chosen for the batching increase the lateness of the first.

Example. Consider two jobs j_1 and j_2, having respectively processing times $p_1 = 3$, $p_2 = 15$ and due dates $d_1 = 7$, $d_2 = 22$. By trying to optimize the completion time (to reduce completion time and maximum lateness of jobs coming next), the algorithm would batch them together with respective lateness $L_1 = 8$, $L_2 \leq 0$ and $L_{\max} = 8$. On the other hand, by considering uncoupling them in two different batches, we would instead have $L_1 = 3-7 \leq 0$ and $L_2 = 18 - 22 \leq 0$, with a resulting $L_{max} \leq 0$.

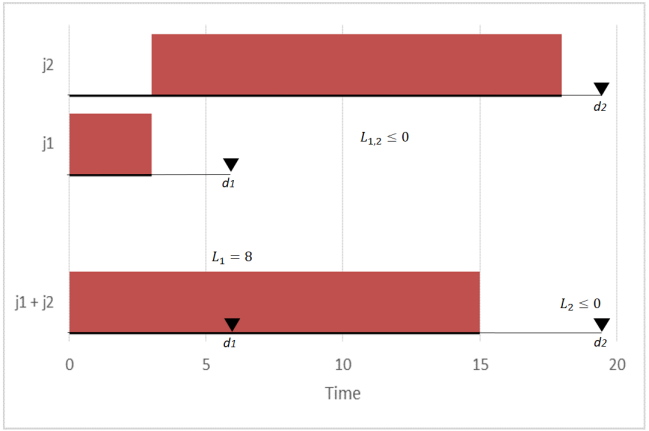

Figure 4.2. Batching together vs processing jobs separately

Therefore, two algorithms are developed, where the first takes the whole requests set and builds a solution considering batching jobs whenever possible - let's call it *systematic batching procedure* (see Fig.4.3) - and a second one, which schedule separately retrieval requests, considering first a partial schedule to get a clue of the final value of the objective function and then completing batches with storage requests in a second stage - let's call it *complementary batching procedure* (see Fig.4.4).

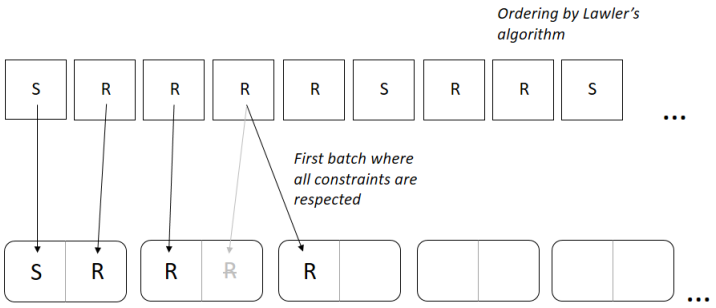

Figure 4.3. Systematic batching procedure

4.2.1 Systematic batching procedure

Algorithms based on this procedure are built upon an initial solution that basically orders the list of jobs to be scheduled following Lawler's algorithm that minimizes the maximum cost (MMC), which in our case is job's lateness and represents our objective function (see Chapter 2). This is done in order to guarantee a first feasible solution in a limited time, despite the existence of many constraints in the form of incompatibilities and precedence relations. The algorithm that minimizes maximum cost would lead to an optimal solution in the case of the same problem without batching; thus we consider it as a first rough good initialization of the job set.

After having ordered the whole job set, a solution is generated as follows. Starting from the beginning, jobs are either batched or not batched together

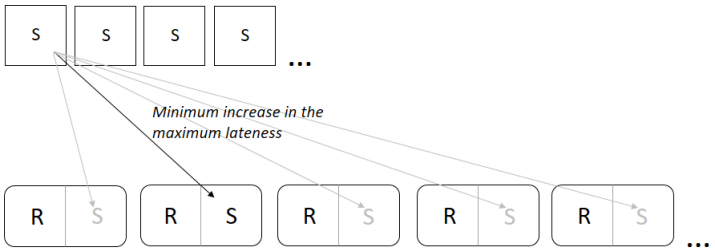

Figure 4.4. Complementary batching procedure

depending if the difference of the processing times of the jobs to be batched together overcomes a certain threshold or not. Algorithm 1 shows how it works.

The solution generated in this step is passed to a method that intensifies the search in that region of the solution space through a local search in its neighbourhood. The neighbourhood is defined as a 2-swap neighbourhood, in which every job is swapped with all others, whenever it respects all constraints. Each new solution is evaluated and, if better, replace the previous solution. At each iteration of the whole job set, the best solution is taken, i.e. a best-improvement method is applied, and the search goes on until a local minimum is reached or the time limit is overcome.

Parameters The performance of the algorithm is influenced by two parameters: a threshold t and the time limit T. t affects the number of jobs that are batched together and the quality of the batching in terms of how much processing times of jobs in the same batch differ at most. It is defined as the percentage of the average processing time of the job set that is tolerated as maximum distance between processing time hence, the higher is the threshold, the more jobs will be batched together.

The time limit T serves as a limit to the local search, thus we don't know how long would it take to reach a local minimum.

```
input  : The list l of jobs to be processed
output: The best found admissible batch schedule
order list l by MMC;
while time limit is not overcome do
    while list is not empty do
        take the first job j1 from l;
        while all jobs j2 ≠ j1 have been considered do
            take job j2;
            if j1, j2 have no predecessors still to be scheduled and j1, j2
             are compatible then
                if p2 − p1 > t · avg(p) then
                    add j1 alone to an empty batch;
                    remove j1 from l;
                else
                    add j1, j2 to an empty batch;
                    remove j1, j2 from l;
                end
                add batch to the batch schedule s;
            else
                proceed in the list l;
            end
        end
    end
    localSearch();
end
```

Algorithm 1: Systematic batching procedure

4.2.2 Complementary batching procedure

The complementary batching procedure starts from a partial schedule obtained by a subset of jobs. The subset contains whether all storage or retrieval requests depending on which of them - once optimally scheduled - generated a larger value of the objective function. The purpose is to obtain a clue of the final maximum lateness that is possibly close to the real value of the final schedule before jobs are batched together. This procedure initializes the problem and it is the same used to compute the lower bound. The structure of the algorithm is described by Algorithm 3. Once the partial

```
input  : A batch schedule s
output: The best found batch schedule in the neighbourhood

L_max*=maximum lateness of schedule s;
s*=input schedule;
while time limit is not overcome do
    L_max* = L_max;
    s* = s;
    while all swaps have been tried do
        pick a job in the first batch;
        pick a job from another batch;
        if the swap doesn't violate constraints then
            swap the jobs;
            if currentL_max < L_max then
                L_max =current L_max;
                s=current s
            end
        end
        undo swap;
    end
end
```

Algorithm 2: Search in a 2-swap neighbourhood

```
input  : requests set

take all retrieval requests;
take all storage requests;
create a batch schedule with just retrievals r ordered by MMC;
create a batch schedule with just storages s ordered by MMC;

if L_max(s) > L_max(r) then
    swap schedules name;
end
return r;
```

Algorithm 3: LB procedure / initialization for the CBP

schedule is defined, the algorithm proceeds with the batching of jobs. For simplicity, we call jobs contained in the partial schedule retrievals and those to be batched storages.

The batching procedure takes the partial batch schedule with just retrievals and the storages list as inputs and works as follows: job k, k = 1,2,...,n_s, with n_s the number of storages, is taken and batched with retrievals where there is a free place left and the maximum lateness is minimized. At the beginning of the procedure all places available for storages are free and the job can be placed in all batches where the current job has a longer - or the less shorter - processing time.

The core point is that every time the maximum lateness of the whole partial schedule (at the beginning our lower bound) is taken in account and not the maximum lateness of the current job. It is pretty different from understanding how to minimize the maximum lateness while creating a schedule from the beginning.

Example. Let's suppose that first retrieval requests are j_1, j_3, j_4, having repectively a processing time $p_1 = 3, p_3 = 1, p_4 = 7$ and due dates $d_1 = 10, d_3 = 12, d_4 = 13$. The first storage request to be placed is j_2, whose processing time and due date are $p_2 = 8$ and $d_2 = 11$. Because d_2 is in

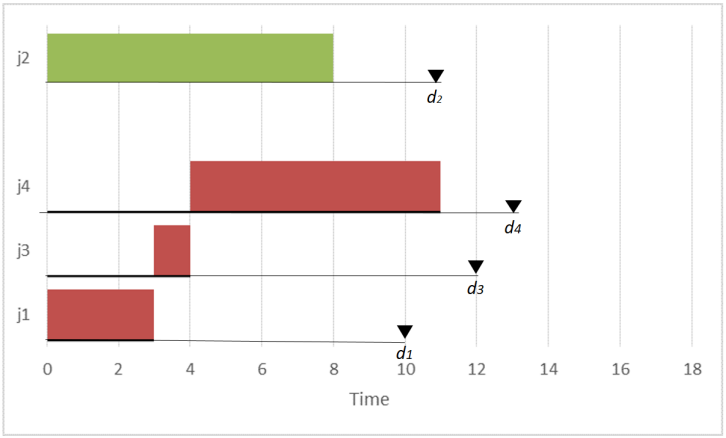

Figure 4.5. Example of a batch assignment decision (a)

between d_1 and d_3, would be reasonable to schedule the job with one of them since the due dates of first two jobs are still larger than p_2 and in order not to reduce due date of j_4 that is already pretty tight. However, it is more convenient to batch j_2 with j_4 because they have nearer processing times. In fact, batching it with first two jobs (for instance, see batching with first job in Fig. 4.6) would increase the lateness of all following jobs respectively of 5 and 7 (the difference between processing times), which is more than 1 which

would be the increase when if batching it with j_4.

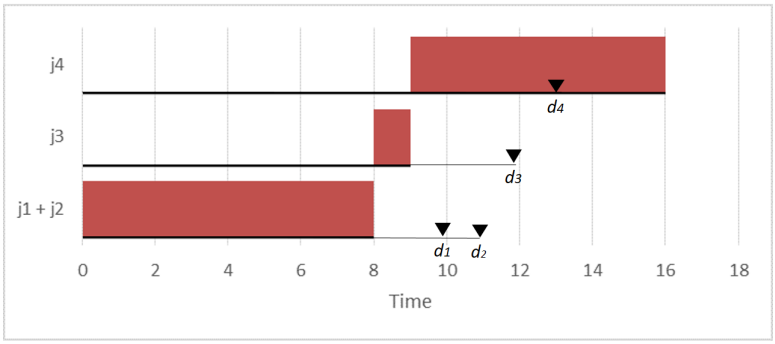

Figure 4.6. Example of a batch assignment decision (b)

This is why it is helpful to have a first clue of the value of the final objective function. It helps to overcome the trade-off that has to be made when batching jobs together since it is very difficult to evaluate the impact in some other greedy way.

The main issue is that the quality of batching, i.e. how much processing times of jobs batched together differ is not taken in account once it is verified that the increase of the maximum lateness of the partial schedule is minimized. First jobs are placed in the first batch where the current job has a greater processing time.

Many ways could avoid inefficient batching but it not always clear how this would influence the batching of following jobs. The selected procedure basically adds a level of recursion on the creation of its solution. That is, during the batching procedure it is considered admissible to replace a storage request already scheduled if the new job has a processing time, which is nearer to the retrieval request in the same batch. If this is the case, the job replaced is scheduled once again and if no better place than the previous is found, i.e. there is no position for which the maximum lateness is not increased, the algorithm returns to the original solution and keeps going on with the usual procedure. A complete recursion would improve the solution but it would also require an exponentially increasing time. That is why only a first level is permitted. In addition, a time limit is imposed at this step meaning that once overcome, the basic procedure completes the solution construction.

N.B. The time limit is imposed to the recursive procedure and not as a limit to the whole algorithm. This last step of completing the solution construction, i.e. without the recursion level, can still require some time and that is why in computational results of next chapter, the time limit is overcome of some seconds by larger instances.

4.2.3 Combining algorithms via knowledge discovery techniques

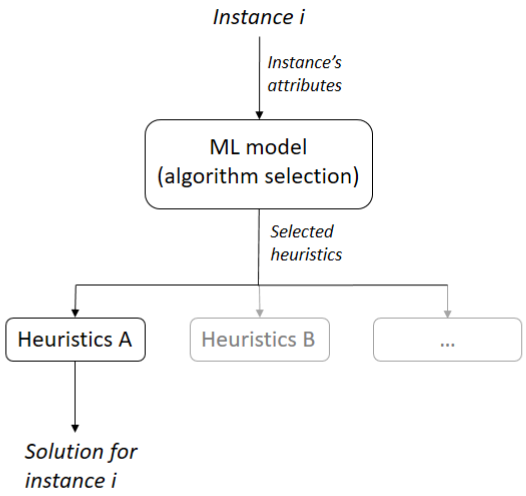

Figure 4.7. Model scheme

The goal of this section is to briefly present another possible approach to use simple heuristic algorithms to schedule jobs and improve solutions. The purpose is to give an overview and propose future developments on the subject. Some tests were run but as it is out of the scope of this work, it is not further discussed.

In line with the goal of keeping solving time short, instead of developing a single - and more time consuming - effective heuristics for all requests sets

(instances), those could be analyzed through data mining techniques from a set of sample data in order to find some common patterns and identify some sub categories of instances to be solved with an easier and more effective algorithm for the specific instance.

This way of proceeding has its roots in the field of knowledge discovery that generally aims in extracting knowledge from large quantities of data. When dealing with problem instances, we deal in practice with a set of data identified as processing times, jobs id, etc. That is, it is not considered all various forms the problem can take but instead, it is here taken in account one specific concretization, i.e. a set of request assiociated with a defined processing time, due date, the set of incompatibilities and so on. On each concretization of the problem, simple heuristics achieve different performances. For instance, the CBP - that recursively replace already scheduled shorter jobs - works better in the case of instances where jobs at the beginning of the request set have longer processing time than those of jobs at the end of the set. For the opposite case, the same algorithm implemented backwards would lead to the same solution in less time.

With regard on this, a model was tested, that considers various heuristics and a Machine Learning model that supports the selection of the proper heuristics for each instance. Fig. 4.7 shows the general scheme of such a model, where an instance is received by the Machine Learning (ML) model, evaluated according to its characteristics (attributes), assigned to a heuristics and scheduled returning the found solution.

In other words, an amount of different heuristic algorithms able to find an admissible schedule for a set of jobs have to be developed. No matter how good the quality of the solution can be, once receiving a request of scheduling a set of jobs, each algorithm returns a feasible schedule. The ML classification algorithm chooses which of them should receive the request. Its work is looking on the sequences of job requests given as input and assign them - depending on their characteristics - to the best solving heuristics. The "intelligence" is all assigned to this higher level and the quality of the output depends mostly on it. The underlying heuristics return good solutions just in specific cases and without a well working algorithm that can recognize some patterns in the requests and assign them to the best performing heuristics, this becomes pretty useless.

Parameters describing their characteristics are assigned to each instance as *attributes* while the best solving heuristics describes the *class* which the instance is assigned to. The two components are input data for the machine

learning tools that construct a model on them that can be re-used to classify future instances.

The quality of the model developed depends on various factors but mainly on the set of attributes that are selected - and on which heuristics selection is based - and on the algorithm used to construct a ML model. For the tests, a multi-class classifier algorithm implemented in the Weka libraries for Java was used. The strength of such a model is that it could be easily adapted to similar problems because of the simplicity of the heuristics.

Attributes As attributes, we need some measures such as statistics on data or representation of unusual characteristics, to effectively summarize all relevant information in order to be able to distinguish different types of instances. As Metan et al.(2010) point out, each attribute should capture some portion of the important information about the system and it is important to define attributes so that their values can be calculated easily. This is due to the fact that when dealing with heuristic procedures it is often a key issue to keep time short and hence the time required to select a new dispatching rule should be negligible.

Attributes considered as characterizing our instances were the tightness of due date, the expected number of successors, the difference between number of storage and retrieval requests, the position of the batch generating the maximum lateness on the partial scheduled and other similarly defined attributes.

Algorithms set The simplest way to build a pool of algorithms from which the ML model chooses the best one for every instance type would be to use different dispatching rules or a combination of those.

Let's suppose that we have two instances of our problem and we want to schedule them. The first having all jobs with similar processing times and the second one with processing times ranging on a wide interval. For the first one, it would be better to create a schedule based on an earliest due date scheduling, while for the other an additional sub-ordering by longest processing time, e.g. in every subset of k jobs, with $k << n$ would probably lead to a more interesting solution.

The more algorithms would reflect real instances characteristics, the more they would be suitable to better solve the specific problem. In addition, the more algorithms would be clearly diversified in the solution, the easier would be to rightly assign them as solver to specific instances.

As a test, some variants on CBP were developed that first schedule longer jobs and to limit maximum difference between processing times of jobs and on SBP by changing the batching threshold and reducing the local search to the most critical portion of the schedule. The general idea of algorithms is shown in Table 4.1, where p_1 is the processing time of the job in the currently considered batch and p_2 the processing time of the job that comes next to be scheduled and for which the decision has to be taken, to batch it or not with the other job.

Tests were run on instances from the literature [Emde et al. (2019)] and showed some dependencies between best solving algorithm and some of instance attributes and the model was shown to improve average solutions in comparison with single heuristics. As it is out of purpose of this work, further evaluations are left apart and the model is kept as a basis for future developments.

```
input : batches - a batch schedule with only retrievals, storages - a
        list of storage requests, best - the best current batch schedule
best = retrievals;
M = big integer;
while k < size(storages) do
    L_max* = M;
    take job j_k;
    while i<size(batches) do
        take batch b_i;
        if b_i contains successors of j_k then
            add a batch with job j_k at position i − 1;
            if currentL_max < L_max* then
                L_max* < currentL_max;
                best = current schedule;
            end
            remove batch created;
        else
            if b_i has empty place and predecessors of j_k are scheduled in
             previous batches then
                add j_k to b_i;
                if current L_max < L_max* then
                    L_max*<current L_max;
                    best = current schedule;
                end
                remove j_k from b_i;
            end
        end
        i++;
        if no place was found then
            add j_k in a new batch at the end of the schedule;
            batches = best;
            k++;
        end
    end
end
return batches;
```

Algorithm 4: Complementary batching procedure

Heuristics	**Description**
	systematic batching procedure-based heuristics
A	batch first ten jobs separately and optimize subschedule through local search; batch all other jobs whenever possible
B	batch all jobs separately
C	batch all compatible jobs when $p_2 - p_1 < 0.1t$
D	batch all compatible jobs when $p_2 - p_1 < 0.3t$
E	batch all compatible jobs when $p_2 - p_1 < 0.5t$
F	batch all compatible jobs when $p_2 - p_1 < 0.7t$
G	batch all compatible jobs when $p_2 - p_1 < 0.9t$
	complementary batching procedure-based heuristics
H	storage requests scheduled only if the difference of processing times between jobs is minimal, i.e. less than 1/10 of maximum processing time
I	to MMC ordering of storages, follows LPT ordering for every subgroup of 20 jobs
K	first third of longest storage requests is scheduled first by LPT, the rest by MMC
L	EDD ordering prevails and it is maintained during the batching process, i.e. storage requests are batched only if their due date doesn't overcome that of current job in batch
M	order storage requests by LPT

Table 4.1. Algorithms

Chapter 5

Computational study

5.1 Test data

Instances used to test the algorithms are generated following the generation scheme presented by Emde et al. (2019) that studied the same problem.

Each instance contains n jobs - each associated to an integer processing time uniformly distributed from 1 to 100 and a due date. Depending on the tightness of the due date λ, the due date of each job is a randomly drawn (uniform distribution) integer from the interval $[1 : (\lambda/b) \cdot \sum_{j \in J}[p_j]]$, with b=2, hence the lower is λ, the tighter is the time window. Each job has 0.5 probability of being a storage or retrieval request. Each instance is additionally characterized by a degree of disagreement between due dates and precedence relations δ and the expected number of successors per job ρ. Depending on these two parameters, precedence relations are determined as follows: a random permutation of jobs $S = (s_1, ..., s_n)$ is generated and for each pair of distinct sequence positions $k, k' \in 1, ..., n, k \neq k'$ jobs are switched if $d_{s_k} < d_{s_{k'}}$ and $k - k' > \delta \cdot n$; given the new sequencing, each job is successor of previous with a certain probability, such that the expected number of successors of each job is ρ. In practice, δ determines how having an early due date for a job matches the fact of having few predecessors, i.e. a low value implies that jobs with an early due date tend to have few predecessors and vice versa and ρ determines the number of precedence relations. We finally call the real number of instance's precedence relations θ.

The data correspond to 300 instances, 5 for each combination of λ, ρ and δ, where the values taken by them are shown in Table 5.1.

parameters	description	values
n	number of jobs	100, 200, 300, 400, 500
λ	tightness of due dates	0.5, 1
δ	degree of disagreement between due dates and precedence relations	0.125, 0.5
ρ	expected number of successors per job	0.125, 0.25, 0.5

Table 5.1. Parameters values

Heuristics are implemented in Java 8.201 and tested on a x64 PC equipped with a 1.7 GHz Intel i3-4005U CPU and 4 GB of RAM. Results are shown in the next section.

5.2 Algorithms performance

The two algorithms presented in chapter 5 are tested on all sample instances and results of different tests are compared in this section. Complete results can be found in appendix A.

All tables show first instances id, then the value of the lower bound LB (see Chapter 5) followed by the values of the objective function f, the percentage gap in respect of the lower bound computed as $(f - LB)/LB * 100$ and time needed to get the solution in CPU seconds. Instances are named with parameters values, following the scheme $n\_\lambda\_\delta\_\rho$ and all data are averages of the 5 instances per each combination of parameters n, λ, δ, ρ. Tests are run with different time limits.

Results within 60 CPU seconds

Table 5.2 and 5.3 respectively summarize results of the complementary batching procedure (CBP) and the systematic batching procedure (SBP) when $\lambda = 0.5$ and $\lambda = 1$. A time limit of 50 CPU seconds is assigned depending on the instance size with a resulting computing time that stays within 60 CPU seconds (once the time limit is overcome, algorithms still go on finishing some operations).

Within the given time limit, CBP returns pretty stable performances for all instances size, having an average gap on the lower bound of ...% for $\lambda = 0.5$ and of ...% for $\lambda = 1$. Instead, SBP returns comparable solutions within the

instance	LB	CBP	gap%	CPU sec.	SBP	gap%	CPU sec.
100_0.5_0.125_0.125	1889.8	1999.2	5.9	0.0	2023.0	7.1	0.0
100_0.5_0.125_0.25	1740.0	1902.8	9.7	0.0	1870.2	7.0	0.0
100_0.5_0.125_0.5	1859.8	2077.8	10.6	0.0	2013.8	6.8	0.0
100_0.5_0.5_0.125	1789.6	1906.6	4.8	0.0	1920.8	4.9	0.0
100_0.5_0.5_0.25	2151.8	2230.8	3.9	0.0	2252.4	4.9	0.0
100_0.5_0.5_0.5	1816.6	1959.0	7.9	0.0	1957.8	7.9	0.0
average			7.1	0.0		6.4	0.0
200_0.5_0.125_0.125	3439.8	3663.8	7.1	2.2	3676.8	7.1	9.4
200_0.5_0.125_0.25	3602.0	3818.6	5.9	1.2	3864.0	6.6	6.8
200_0.5_0.125_0.5	3707.8	3935.2	5.8	2.0	3997.0	7.4	8.4
200_0.5_0.5_0.125	3642.2	3881.8	6.8	2.0	3863.4	5.3	8.6
200_0.5_0.5_0.25	3688.2	3826.8	4.4	1.2	3911.4	6.4	8.6
200_0.5_0.5_0.5	3775.0	3899.0	3.4	0.8	3951.2	4.8	8.2
average			5.6	1.6		6.3	8.3
300_0.5_0.125_0.125	4923.0	5312.8	8.2	19.4	5324.8	8.3	57.4
300_0.5_0.125_0.25	5263.8	5614.0	6.5	26.8	5655.8	6.6	51.0
300_0.5_0.125_0.5	5217.8	5506.4	5.4	18.2	5727.6	8.0	59.6
300_0.5_0.5_0.125	4981.8	5528.2	8.4	32.6	5426.2	7.1	57.4
300_0.5_0.5_0.25	5221.4	5582.0	7.2	16.8	5653.2	8.3	50.8
300_0.5_0.5_0.5	5500.4	5760.6	5.0	13.0	5879.8	7.2	58.4
average			6.8	21.1		7.6	55.8
400_0.5_0.125_0.125	6753.4	7010.6	4.0	55.4	10670.4	58.5	62.2
400_0.5_0.125_0.25	6531.2	7060.6	8.3	65.6	10350.0	58.5	62.0
400_0.5_0.125_0.5	6743.4	7344.6	8.9	54.2	10587.0	57.0	61.4
400_0.5_0.5_0.125	6772.0	7299.0	8.0	65.6	10849.2	60.4	62.6
400_0.5_0.5_0.25	6334.8	6965.8	10.0	60.8	10379.6	63.9	61.8
400_0.5_0.5_0.5	6796.6	7197.4	6.1	59.8	10473.2	54.5	61.6
average			7.5	60.2		58.8	61.9
500_0.5_0.125_0.125	7927.4	8907.2	12.3	68.0	14104.8	77.9	64.6
500_0.5_0.125_0.25	8390.6	9276.2	10.8	68.4	14745.8	76.1	63.4
500_0.5_0.125_0.5	8762.0	9199.6	5.1	68.2	15131.4	72.6	62.8
500_0.5_0.5_0.125	8095.6	9307.4	15.1	70.2	14149.0	74.8	62.2
500_0.5_0.5_0.25	8522.4	8970.4	5.5	66.8	14324.8	68.3	63.2
500_0.5_0.5_0.5	8037.4	9578.8	19.3	70.4	14015.4	74.6	65.0
average			11.3	68.7		74.1	63.5

Table 5.2. Average results ($\lambda = 0.5$)

same time limits just for instances with 100, 200 and 300 jobs, whereas for larger instances it does not reach useful solutions on time. Up to 300 jobs

instance	LB	CBP	gap%	CPU sec.	SBP	gap%	CPU sec.
100_1_0.125_0.125	1244.4	1430.4	15.1	0.0	1468.4	18.8	0.0
100_1_0.125_0.25	1263.0	1471.6	17.4	0.0	1449.4	15.0	0.0
100_1_0.125_0.5	1352.2	1439.6	7.2	0.0	1456.4	8.2	0.0
100_1_0.5_0.125	1211.0	1346.8	11.9	0.0	1384.4	14.9	0.0
100_1_0.5_0.25	1264.0	1362.0	9.2	0.0	1395.2	11.4	0.0
100_1_0.5_0.5	1287.6	1438.2	12.1	0.0	1466.0	14.3	0.0
average			12.2	0.0		13.8	0.0
200_1_0.125_0.125	2386.2	2692.0	13.6	6.6	2750.6	16.0	6.6
200_1_0.125_0.25	2402.8	2605.0	3.6	2.8	2783.8	10.6	6.8
200_1_0.125_0.5	2351.8	2590.6	11.1	1.4	2727.4	16.5	7.0
200_1_0.5_0.125	2438.6	2642.0	8.5	3.8	2620.0	7.5	6.8
200_1_0.5_0.25	2376.2	2675.6	12.5	6.8	2707.2	14.1	5.6
200_1_0.5_0.5	2258.6	2597.6	15.7	3.0	2611.6	16.1	5.8
average			10.8	4.1		13.5	6.4
300_1_0.125_0.125	3252.4	3793.2	17.9	43.2	3709.2	14.9	47.6
300_1_0.125_0.25	3522.8	3970.0	6.2	58.8	3948.0	8.2	52.2
300_1_0.125_0.5	3739.0	3940.6	6.0	20.8	4125.2	10.8	47.4
300_1_0.5_0.125	3364.2	3701.0	10.4	55.0	3870.6	15.4	45.6
300_1_0.5_0.25	3187.4	3792.4	19.4	44.0	3749.6	17.7	48.4
300_1_0.5_0.5	3548.2	3820.2	8.0	42.2	3932.2	11.1	46.8
average			11.3	44.0		13.0	48.0
400_1_0.125_0.125	4322.6	4976.0	15.0	69.2	7856.2	82.2	62.4
400_1_0.125_0.25	4581.2	5143.0	13.2	71.2	7754.4	70.4	62.0
400_1_0.125_0.5	4282.6	4763.0	11.3	67.6	7912.6	85.2	61.0
400_1_0.5_0.125	4271.4	4614.2	8.1	69.8	7725.0	81.1	61.4
400_1_0.5_0.25	4548.6	4842.6	6.8	68.8	7937.8	74.7	61.4
400_1_0.5_0.5	4270.2	5039.2	18.1	68.6	7778.4	82.4	61.8
average			12.1	69.2		79.3	61.7
500_1_0.125_0.125	4729.0	5597.6	18.5	66.8	10086.0	113.7	63.2
500_1_0.125_0.25	5876.0	6179.6	5.4	62.2	10688.6	82.4	63.4
500_1_0.125_0.5	5313.6	6181.0	16.0	65.4	10785.8	103.1	63.2
500_1_0.5_0.125	5102.0	5969.0	17.7	64.8	10227.8	100.8	63.0
500_1_0.5_0.25	5871.6	6256.2	7.1	63.0	11035.6	89.4	65.0
500_1_0.5_0.5	5468.8	5846.6	7.0	64.0	10722.4	96.6	64.2
average			12.0	64.4		97.7	63.7

Table 5.3. Average results ($\lambda = 1$)

also, algorithms return comparable solutions and there is no evident pattern that identifies which performs better in which case.

The difference average percentage gap between instances where $\lambda = 0.5$ and $\lambda = 1$ can be explained by the fact that by distant due dates instances are more difficult to schedule. The latter means that there are more possible combinations of jobs without changing the value of the maximum lateness and it is therefore more difficult to schedule them with simple combination of heuristic procedures.

There is no visible pattern or dependence with regard of other instance parameters, i.e. δ and ρ meaning that the algorithms tend to maintain equal performances under some different request conditions.

Comparison with an algorithm with linear time

n	CBP	SBP	EDD
100	7.1	6.4	34.8
200	5.6	6.3	37.0
300	6.8	7.6	39.5
400	7.5	58.8	45.3
500	11.3	74.1	47.4

Table 5.4. Average percentage gap for each instance size ($\lambda = 0.5$)

n	CBP	SBP	EDD
100	12.2	13.8	51.2
200	10.8	13.5	55.1
300	11.3	13.0	55.7
400	12.1	79.3	61.6
500	12.0	97.7	64.4

Table 5.5. Average percentage gap for each instance size ($\lambda = 1$)

For further comparison, instances are evaluated with an algorithm $O(n)$. Those are often inefficient but are still sometimes used in industry environments where no scheduling optimization is used for various reasons.

For the problem on a single machine minimizing the maximum lateness without batching, the simplest dispatching rule would be the Earliest Due Date (EDD) ordering (optimal without constraints). Therefore, we schedule jobs ordering jobs by EDD and systematically batching together whenever

possible, i.e. whenever all constraints are satisfied. The exact procedure is the one used for the SBP (chapter 5) and let's refer to it as EDD.

EDD finds all solutions in negligible time - less than 1 CPU second - and average solutions and percentage gap with respect to the lower bound of CBP, SBP and EDD are shown in Table 5.4 and 5.5. Percentage gaps slightly increase the larger are instances ranging from 34.8% for 100 jobs and 47.4% for 500 jobs. Both CBP and SBP largely overcome EDD results.

5.3 Time comparison

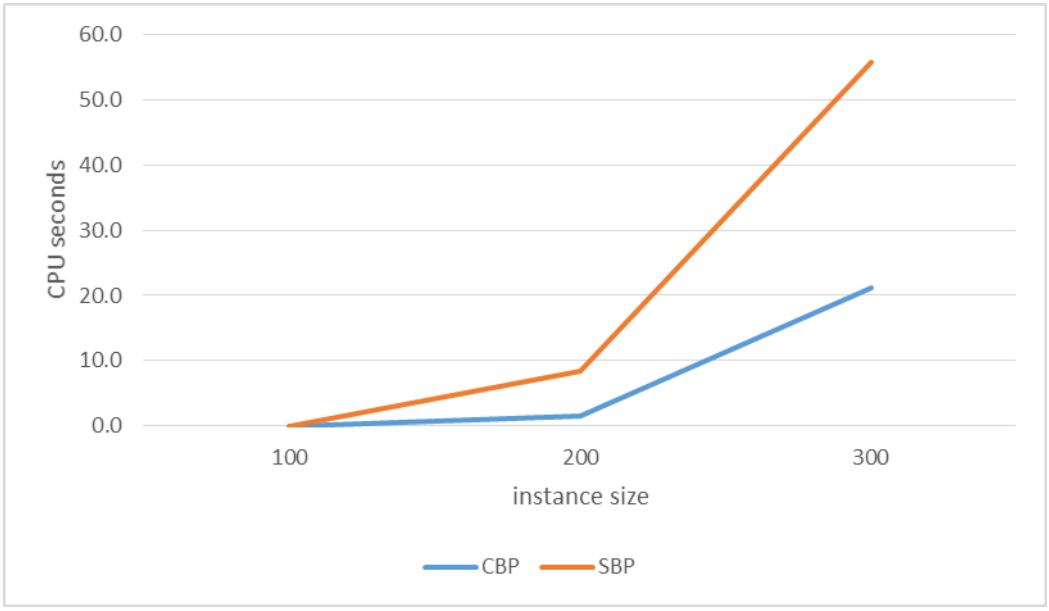

Figure 5.1. Algorithms behaviour vs CPU time ($\lambda = 0.5$)

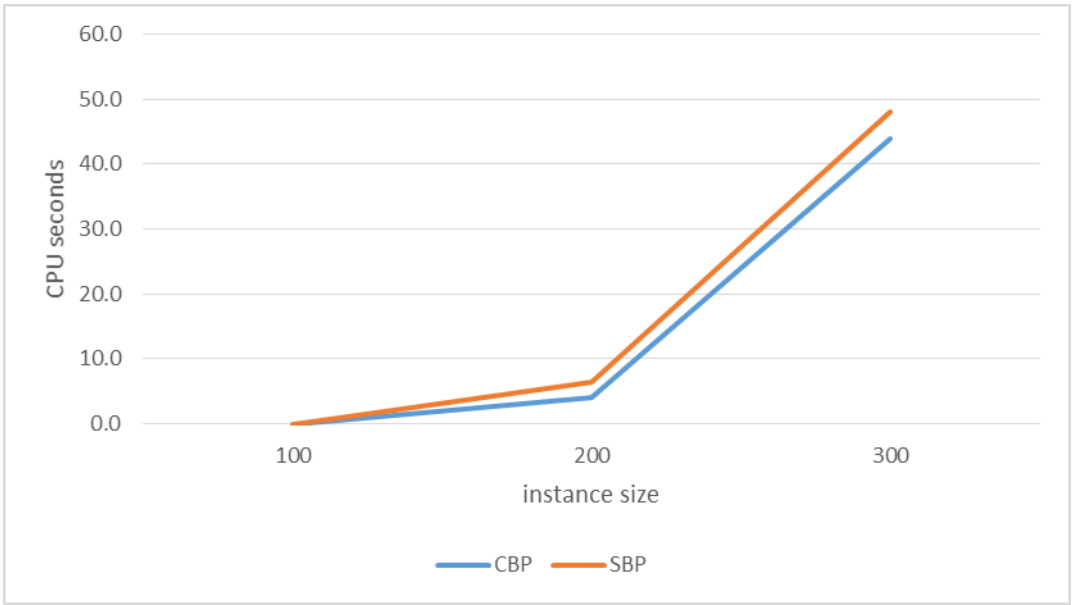

Figure 5.2. Algorithms behaviour vs CPU time ($\lambda = 1$)

In graphs contained in Fig.5.1 and 5.2 we compare the time behaviour of the two heuristic procedures considered as the size of the problem increase. We already saw that within the time limit of 60 CPU seconds, instances up to 300 jobs can be solved at the best of algorithms capability. CBP is shown to be much faster than SBP for instances where due dates are in average tighter, while in the other case, the time required by the two algorithms to find their solutions is similar (still, CBP returns better solutions).

It is clear that the computing time grows more than linearly. A rough evaluation of computing steps for each procedure is therefore made to understand the growth in time with respect of size of the problem.

Main steps of the CBP are (we call here n_r the number of retrieval requests, i.e. jobs contained in the subset of jobs scheduled first by the CBP):

- the partitioning of requests in storage and retrieval requests that requires n steps,
- the MMC ordering that was proved working in $O(n_r^2)$,
- the batching procedure that it is complexer to be evaluated in advance. Finding the best place for every storage request approximately requires $(n_r(n_r+1)/2)$ steps that is the sum of $n_r + n_r - 1 + ... + 1$ steps to place every storage request. But in many cases the sum stops earlier than trying all available place and this happens both if storage request are less than retrievals and whenever a successor of the job to be placed is encountered. If the recursive is considered, we add a quadratic factor of $(n_r(n_r+1)/2) - 1$ leading to an algorithm bounded in the worst case by a n_r^4 factor. If we consider the number of $n_r = n/2$, this is equal to $O((n/2)^4)$.

As a result, we have that CBP works in $O((n)^4)$.

Main steps of the SBP are instead:

- the MMC ordering, working in $O(n^2)$,
- the batching procedure, working in $O(n)$ (it goes through all jobs and simply decide if it is batched with previous job or separately),
- the local search working in $O(n_l^2)$ for every new best neighbour encountered. Since the final schedule considers also partially filled batches, the

number of elements considered for the local search n_l is larger than the real size of the problem n, i.e. $n_l > n$. The difference depends on the initial sequence of jobs. If this had pairwise all jobs having similar (i.e. not overcoming the batching threshold) processing times, then $n_l = n$. Viceversa, in the opposite case, $n_l = 2n$.

The resulting computing time is an algorithm $O(n^2)$.

Both algorithms work in a polynomially increasing computing times bounded respectively by n^4 or n^2. CBP is more time-expensive by very large n. Anyway, the difference in the basis - CBP considers approximately the half of jobs and SBP could hypotetically consider up to the double of the jobs - is significant when looking at the computing time in terms of seconds. We see in fact from the graphs above and from results from Table 5.2 and 5.3 that both algorithms complete their procedure in a negligible time - less than 1 CPU second - by instances with 100 jobs. By 200 jobs, CBP reaches its solutions within 5 CPU seconds and SBP within 10 and the more the size increases, the larger the difference. By larger problems both procedures do not complete their normal algorithm and computing times are determined by the time limit. Anyway, as by SBP the local search is essential part of the scheduling process, leading otherwise to a really bad schedule, CBP still leads to acceptable solutions when the recursive procedure is applied just to first jobs (see results in previous section).

Chapter 6

Conclusions

In AS/RS environments such as in other manufacturing or distribution environments dealing with a large quantity of requests to be processed, the need of fast algorithms for scheduling them is essential. A trade-off between solution quality and the time to get a response is a core point and has to be determined by the system management. Quality of final schedules have usually an impact on operating costs and lead to system inefficiencies but on the other side, when continuously receiving new sets of requests, it is not plausible to reserve a lot of time just for the scheduling.

Within this trade-off, find place plenty of heuristic methods that serve for this purpose. Heuristics cover a broad variety of problems and cases and stay on the opposite side of exact methods that find optimal solutions but often lead to excessively large computing times. Heuristics instead, are methods that find approximate solutions within a relatively short period of time.

In this work, two fast heuristic algorithms are developed using different approaches based on simple dispatching rules, neighbourhood search and a combination of both. Sample instances were generated as proposed in the literature to compare and evaluate the two heuristics. Both algorithms work in a polynomially increasing time with respect of the problem size. They solve the problem - deriving from the AS/RS context - of scheduling of jobs on a single machine with incompatibilities and precedence constraints minimizing the maximum lateness.

The problem is a generalization of the batching machine with restricted batch size problem (BMRS) without constraints known from the literature to be NP-hard. As for any NP-hard problem, no algorithm is known that could solve it optimally in polynomial time and it leads soon to unfeasibility

as the problem size increases.

Emde et al. (2019) studied the same problem and presented an exact algorithm based on branch and benders-cut that can solve instances up to 100 jobs optimally within few seconds but cannot afford larger instances in a short amount of time. Proposed heuristics are tested in comparison with a lower bound and shown to return useful solutions within few minutes for instances size up to 500 jobs.

As a review, we have two heuristic procedures that work as follows.

The first one - referred in the report as *systematic batching procedure (SBP)* - orders all requests by Lawler's algorithm (MMC) and then systematically batches them when all constraints are respected and the processing time of the two jobs is similar; on this initial solution a local search is performed.

The second one - referred to as *complementary batching procedure (CBP)* - divides the problem into two steps by first scheduling just one type of request and then optimizing batching of the other type of requests.

The first is based on the classical concept of intensification on an initial solution, the second schedules a subset of jobs and than uses the information of the maximum lateness of this partial schedule to batch the rest of the jobs.

Results from computational tests showed that the latter results in better performances. It often overcomes solutions found by the first one; however, on smaller problem size, the first one improves many results. CBP maintained a pretty stable percentage gap with respect of a lower bound and can be therefore considered to solve also larger instances by a feasible time and still largely improving solutions returned by other dispatching rules-based algorithms, e.g. EDD ordering adapted to the batching procedure.

Appendix A

Evaluation Results

All results from the computational test are listed below. As in chapter 6, we have first the instance ID that follows the name scheme $n\_\lambda\_\delta\_\rho\_i$, where i is an integer from 0 to 4 used to univocally identify instances with same parameters value. Then, we have the value of the lower bound, the maximum lateness of the solution found with the CBP algorithm, the percentage gap with respect to the lower bound and the CPU time. It follows the same for the SBP algorithm. Finally, in last two columns, there is the value of the maximum lateness found with an EDD-schedule and its percentage gap.

Instance ID	**LB**	**CBP**	**gap%**	**CPU sec.**	**SBP**	**gap%**	**CPU sec.**	**EDD**	**gap%**
100_0.5_0.125_0.125_0	1859	1980	6.5	0	2032	9.3	0	2657	42.9
100_0.5_0.125_0.125_1	1805	1865	3.3	0	1889	4.7	0	2528	40.1
100_0.5_0.125_0.125_2	2056	2056	0.0	0	2144	4.3	0	2610	26.9
100_0.5_0.125_0.125_3	1812	2019	11.4	0	1959	8.1	0	2356	30.0
100_0.5_0.125_0.125_4	1917	2076	8.3	0	2091	9.1	0	2514	31.1
100_0.5_0.125_0.25_0	2046	2188	6.9	0	2170	6.1	0	2934	43.4
100_0.5_0.125_0.25_1	1660	1757	5.8	0	1756	5.8	0	2385	43.7
100_0.5_0.125_0.25_2	1516	1742	14.9	0	1681	10.9	0	2099	38.5
100_0.5_0.125_0.25_3	1811	1814	0.0	0	1876	0.0	0	2437	0.0
100_0.5_0.125_0.25_4	1667	2013	20.8	0	1868	12.1	0	2541	52.4
100_0.5_0.125_0.5_0	1885	1974	4.7	0	1997	5.9	0	2543	34.9
100_0.5_0.125_0.5_1	1706	1825	7.0	0	1834	7.5	0	2510	47.1
100_0.5_0.125_0.5_2	1936	2079	7.4	0	2040	5.4	0	2681	38.5
100_0.5_0.125_0.5_3	2020	2168	0.0	0	2183	0.0	0	2856	0.0
100_0.5_0.125_0.5_4	1752	2343	33.7	0	2015	15.0	0	2560	46.1
100_0.5_0.5_0.125_0	1762	1799	2.1	0	1816	3.1	0	2458	39.5
100_0.5_0.5_0.125_1	1716	1794	4.5	0	1836	7.0	0	2279	32.8
100_0.5_0.5_0.125_2	1891	2077	0.0	0	2123	0.0	0	2737	0.0
100_0.5_0.5_0.125_3	1593	1847	15.9	0	1760	10.5	0	2414	51.5
100_0.5_0.5_0.125_4	1986	2016	1.5	0	2069	4.2	0	2612	31.5
100_0.5_0.5_0.25_0	2226	2334	4.9	0	2328	4.6	0	2718	22.1
100_0.5_0.5_0.25_1	2101	2115	0.7	0	2168	3.2	0	2731	30.0
100_0.5_0.5_0.25_2	1730	1838	6.2	0	1891	9.3	0	2679	54.9
100_0.5_0.5_0.25_3	2573	2588	0.6	0	2645	2.8	0	3069	19.3
100_0.5_0.5_0.25_4	2129	2279	7.0	0	2230	4.7	0	3064	43.9

100_0.5_0.5_0.5_0	1973	2037	3.2	0	1986	0.7	0	2797	41.8
100_0.5_0.5_0.5_1	1866	2035	9.1	0	2031	8.8	0	2814	50.8
100_0.5_0.5_0.5_2	1748	1809	3.5	0	1853	6.0	0	2305	31.9
100_0.5_0.5_0.5_3	1799	2086	16.0	0	2063	14.7	0	2592	44.1
100_0.5_0.5_0.5_4	1697	1828	7.7	0	1856	9.4	0	2304	35.8
100_1.0_0.125_0.125_0	1486	1596	7.4	0	1594	7.3	0	2044	37.6
100_1.0_0.125_0.125_1	1118	1280	14.5	0	1435	28.4	0	2118	89.4
100_1.0_0.125_0.125_2	1374	1685	22.6	0	1572	14.4	0	1986	44.5
100_1.0_0.125_0.125_3	1142	1341	17.4	0	1443	26.4	0	2142	87.6
100_1.0_0.125_0.125_4	1102	1250	13.4	0	1298	17.8	0	1923	74.5
100_1.0_0.125_0.25_0	1197	1287	0.0	0	1284	0.0	0	1811	0.0
100_1.0_0.125_0.25_1	1480	1620	9.5	0	1586	7.2	0	2148	45.1
100_1.0_0.125_0.25_2	906	1388	53.2	0	1292	42.6	0	1710	88.7
100_1.0_0.125_0.25_3	1555	1731	11.3	0	1777	14.3	0	2248	44.6
100_1.0_0.125_0.25_4	1177	1332	13.2	0	1308	11.1	0	1755	49.1
100_1.0_0.125_0.5_0	1527	1527	0.0	0	1602	4.9	0	1967	28.8
100_1.0_0.125_0.5_1	1283	1323	3.1	0	1360	6.0	0	1832	42.8
100_1.0_0.125_0.5_2	1345	1445	7.4	0	1463	8.8	0	2207	64.1
100_1.0_0.125_0.5_3	1109	1356	22.3	0	1297	17.0	0	1805	62.8
100_1.0_0.125_0.5_4	1497	1547	3.3	0	1560	4.2	0	2229	48.9
100_1.0_0.5_0.125_0	1256	1576	25.5	0	1650	31.4	0	2291	82.4
100_1.0_0.5_0.125_1	1388	1419	2.2	0	1443	4.0	0	1910	37.6
100_1.0_0.5_0.125_2	1079	1132	4.9	0	1130	4.7	0	1546	43.3
100_1.0_0.5_0.125_3	1015	1285	26.6	0	1295	27.6	0	1750	72.4
100_1.0_0.5_0.125_4	1317	1322	0.4	0	1404	6.6	0	1737	31.9
100_1.0_0.5_0.25_0	1647	1651	0.2	0	1700	3.2	0	2106	27.9
100_1.0_0.5_0.25_1	976	1269	30.0	0	1210	24.0	0	1646	68.6
100_1.0_0.5_0.25_2	1227	1245	1.5	0	1304	6.3	0	1615	31.6
100_1.0_0.5_0.25_3	1278	1326	3.8	0	1426	11.6	0	1850	44.8
100_1.0_0.5_0.25_4	1192	1319	10.7	0	1336	12.1	0	1738	45.8
100_1.0_0.5_0.5_0	1345	1448	7.7	0	1453	8.0	0	1700	26.4
100_1.0_0.5_0.5_1	1146	1307	14.0	0	1445	26.1	0	2071	80.7
100_1.0_0.5_0.5_2	1403	1403	0.0	0	1512	7.8	0	1980	41.1
100_1.0_0.5_0.5_3	1294	1447	11.8	0	1440	11.3	0	1865	44.1
100_1.0_0.5_0.5_4	1250	1586	26.9	0	1480	18.4	0	1859	48.7
200_0.5_0.125_0.125_0	3498	3567	2.0	1	3740	6.9	10	5020	43.5
200_0.5_0.125_0.125_1	3819	3833	0.4	3	3926	2.8	10	5055	32.4
200_0.5_0.125_0.125_2	3193	3477	8.9	3	3486	9.2	10	4719	47.8
200_0.5_0.125_0.125_3	3066	3757	22.5	3	3385	10.4	7	4861	58.5
200_0.5_0.125_0.125_4	3623	3685	1.7	1	3847	6.2	10	5320	46.8
200_0.5_0.125_0.25_0	3475	3909	12.5	2	3998	15.1	5	5353	54.0
200_0.5_0.125_0.25_1	3456	3824	10.6	1	3806	10.1	3	4822	39.5
200_0.5_0.125_0.25_2	3659	3811	4.2	1	3820	4.4	10	5170	41.3
200_0.5_0.125_0.25_3	3830	3880	0.0	1	3987	0.0	6	5238	0.0
200_0.5_0.125_0.25_4	3590	3669	2.2	1	3709	3.3	10	5312	48.0
200_0.5_0.125_0.5_0	4230	4246	0.4	1	4385	3.7	8	5483	29.6
200_0.5_0.125_0.5_1	3800	3982	4.8	0	4153	9.3	9	5452	43.5
200_0.5_0.125_0.5_2	3425	3921	14.5	5	3977	16.1	9	5010	46.3
200_0.5_0.125_0.5_3	3519	3635	0.0	2	3621	0.0	8	4951	0.0
200_0.5_0.125_0.5_4	3565	3892	9.2	2	3849	8.0	8	5310	48.9
200_0.5_0.5_0.125_0	3709	3824	3.1	2	3922	5.7	9	5031	35.6
200_0.5_0.5_0.125_1	3863	4156	7.6	1	4013	3.9	10	5343	38.3
200_0.5_0.5_0.125_2	3660	3683	0.0	1	3835	0.0	8	5128	0.0
200_0.5_0.5_0.125_3	3659	3666	0.2	2	3772	3.1	9	4889	33.6
200_0.5_0.5_0.125_4	3320	4080	22.9	4	3775	13.7	7	5102	53.7
200_0.5_0.5_0.25_0	4150	4150	0.0	1	4276	3.0	10	5265	26.9
200_0.5_0.5_0.25_1	3632	3746	3.1	1	3880	6.8	8	5108	40.6
200_0.5_0.5_0.25_2	3530	3587	1.6	1	3752	6.3	10	4910	39.1

200_0.5_0.5_0.25_3	4100	4107	0.2	1	4275	4.3	6	5391	31.5
200_0.5_0.5_0.25_4	3029	3544	17.0	2	3374	11.4	9	4728	56.1
200_0.5_0.5_0.5_0	3661	3865	5.6	0	3847	5.1	8	5035	37.5
200_0.5_0.5_0.5_1	3802	3818	0.4	1	3922	3.2	6	5041	32.6
200_0.5_0.5_0.5_2	3965	3965	0.0	0	4034	1.7	10	5262	32.7
200_0.5_0.5_0.5_3	3551	3939	10.9	2	3911	10.1	8	4863	36.9
200_0.5_0.5_0.5_4	3896	3908	0.3	1	4042	3.7	9	5200	33.5
200_1.0_0.125_0.125_0	2299	2931	27.5	11	2780	20.9	10	3684	60.2
200_1.0_0.125_0.125_1	2298	2600	13.1	2	2626	14.3	6	3497	52.2
200_1.0_0.125_0.125_2	2836	2840	0.1	9	2897	2.2	8	3940	38.9
200_1.0_0.125_0.125_3	2125	2627	23.6	10	2665	25.4	4	3521	65.7
200_1.0_0.125_0.125_4	2373	2462	3.8	1	2785	17.4	5	4067	71.4
200_1.0_0.125_0.25_0	2104	2724	0.0	8	2572	0.0	8	3514	0.0
200_1.0_0.125_0.25_1	3369	3369	0.0	1	4150	23.2	10	4490	33.3
200_1.0_0.125_0.25_2	2185	2294	5.0	1	2330	6.6	4	3573	63.5
200_1.0_0.125_0.25_3	2318	2436	5.1	2	2648	14.2	7	3804	64.1
200_1.0_0.125_0.25_4	2038	2202	8.0	2	2219	8.9	5	3354	64.6
200_1.0_0.125_0.5_0	1871	2390	27.7	1	2317	23.8	7	3826	104.5
200_1.0_0.125_0.5_1	2486	2740	10.2	2	2723	9.5	5	3601	44.9
200_1.0_0.125_0.5_2	2327	2470	6.1	1	2865	23.1	5	3882	66.8
200_1.0_0.125_0.5_3	2375	2580	8.6	2	2683	13.0	10	3724	56.8
200_1.0_0.125_0.5_4	2700	2773	2.7	1	3049	12.9	8	4205	55.7
200_1.0_0.5_0.125_0	2444	2457	0.5	3	2531	3.6	8	3578	46.4
200_1.0_0.5_0.125_1	2304	2487	7.9	3	2583	12.1	5	3728	61.8
200_1.0_0.5_0.125_2	2400	2446	1.9	2	2550	6.3	8	3919	63.3
200_1.0_0.5_0.125_3	2402	3101	29.1	10	2661	10.8	6	3803	58.3
200_1.0_0.5_0.125_4	2643	2719	2.9	1	2775	5.0	7	4127	56.1
200_1.0_0.5_0.25_0	2280	2759	21.0	3	2831	24.2	4	3628	59.1
200_1.0_0.5_0.25_1	2195	2334	6.3	5	2513	14.5	4	3232	47.2
200_1.0_0.5_0.25_2	2543	3011	18.4	11	2776	9.2	9	3823	50.3
200_1.0_0.5_0.25_3	2407	2590	7.6	10	2651	10.1	6	3343	38.9
200_1.0_0.5_0.25_4	2456	2684	9.3	5	2765	12.6	5	3707	50.9
200_1.0_0.5_0.5_0	2180	2882	32.2	1	2613	19.9	6	3509	61.0
200_1.0_0.5_0.5_1	2079	2575	23.9	2	2542	22.3	7	3396	63.3
200_1.0_0.5_0.5_2	2427	2494	2.8	1	2632	8.4	3	3472	43.1
200_1.0_0.5_0.5_3	2141	2500	16.8	10	2597	21.3	8	3283	53.3
200_1.0_0.5_0.5_4	2466	2537	2.9	1	2674	8.4	5	3855	56.3
300_0.5_0.125_0.125_0	5332	5577	4.6	17	5752	7.9	49	7827	46.8
300_0.5_0.125_0.125_1	5353	5416	1.2	25	5634	5.2	60	6854	28.0
300_0.5_0.125_0.125_2	4616	5419	17.4	24	5022	8.8	60	7048	52.7
300_0.5_0.125_0.125_3	4799	5228	8.9	18	5166	7.6	58	7343	53.0
300_0.5_0.125_0.125_4	4515	4924	9.1	13	5050	11.8	60	6966	54.3
300_0.5_0.125_0.25_0	5078	5546	9.2	20	5649	11.2	60	7105	39.9
300_0.5_0.125_0.25_1	5674	5752	1.4	9	5997	5.7	60	7575	33.5
300_0.5_0.125_0.25_2	5473	5541	1.2	29	5718	4.5	58	7601	38.9
300_0.5_0.125_0.25_3	5293	5427	0.0	13	5551	0.0	39	7599	0.0
300_0.5_0.125_0.25_4	4801	5804	20.9	63	5364	11.7	38	7325	52.6
300_0.5_0.125_0.5_0	5428	5766	6.2	15	5973	10.0	61	7819	44.0
300_0.5_0.125_0.5_1	4995	5277	5.6	13	5644	13.0	60	7281	45.8
300_0.5_0.125_0.5_2	5471	5847	6.9	25	5934	8.5	60	7700	40.7
300_0.5_0.125_0.5_3	5128	5164	0.0	13	5598	0.0	60	7205	0.0
300_0.5_0.125_0.5_4	5067	5478	8.1	25	5489	8.3	57	7430	46.6
300_0.5_0.5_0.125_0	5101	5309	4.1	46	5446	6.8	60	7043	38.1
300_0.5_0.5_0.125_1	4577	5961	30.2	64	5398	17.9	60	7338	60.3
300_0.5_0.5_0.125_2	4900	5644	0.0	31	5408	0.0	60	7047	0.0
300_0.5_0.5_0.125_3	5352	5425	1.4	10	5601	4.7	47	7713	44.1
300_0.5_0.5_0.125_4	4979	5302	6.5	12	5278	6.0	60	7436	49.3
300_0.5_0.5_0.25_0	5109	5496	7.6	47	5505	7.8	40	7421	45.3

300_0.5_0.5_0.25_1	5351	5465	2.1	7	5603	4.7	50	8134	52.0
300_0.5_0.5_0.25_2	4937	5639	14.2	10	5387	9.1	44	7603	54.0
300_0.5_0.5_0.25_3	5674	5687	0.2	10	6167	8.7	60	7637	34.6
300_0.5_0.5_0.25_4	5036	5623	11.7	10	5604	11.3	60	7501	48.9
300_0.5_0.5_0.5_0	6558	6558	0.0	9	6808	3.8	60	8016	22.2
300_0.5_0.5_0.5_1	4907	5325	8.5	14	5720	16.6	60	7310	49.0
300_0.5_0.5_0.5_2	5482	5568	1.6	13	5747	4.8	60	7305	33.3
300_0.5_0.5_0.5_3	5366	5855	9.1	13	5580	4.0	54	7185	33.9
300_0.5_0.5_0.5_4	5189	5497	5.9	16	5544	6.8	58	7426	43.1
300_1.0_0.125_0.125_0	3222	3420	6.1	29	3462	7.4	42	5167	60.4
300_1.0_0.125_0.125_1	3127	4150	32.7	57	3644	16.5	61	5071	62.2
300_1.0_0.125_0.125_2	2809	3774	34.4	66	3598	28.1	45	5166	83.9
300_1.0_0.125_0.125_3	3147	3625	15.2	51	3695	17.4	30	5379	70.9
300_1.0_0.125_0.125_4	3957	3997	1.0	13	4147	4.8	60	5806	46.7
300_1.0_0.125_0.25_0	3029	4204	0.0	66	3712	0.0	55	5650	0.0
300_1.0_0.125_0.25_1	4158	4229	1.7	61	4356	4.8	46	5729	37.8
300_1.0_0.125_0.25_2	3681	3693	0.3	62	4016	9.1	60	5640	53.2
300_1.0_0.125_0.25_3	3362	3784	12.6	67	3868	15.1	55	5176	54.0
300_1.0_0.125_0.25_4	3384	3940	16.4	38	3788	11.9	45	5439	60.7
300_1.0_0.125_0.5_0	3796	3807	0.3	18	4158	9.5	39	5251	38.3
300_1.0_0.125_0.5_1	3215	3854	19.9	35	3812	18.6	44	5107	58.8
300_1.0_0.125_0.5_2	3611	3819	5.8	22	3938	9.1	60	5360	48.4
300_1.0_0.125_0.5_3	3618	3768	4.1	17	4117	13.8	52	5813	60.7
300_1.0_0.125_0.5_4	4455	4455	0.0	12	4601	3.3	42	6301	41.4
300_1.0_0.5_0.125_0	3227	3627	12.4	63	3766	16.7	50	4939	53.1
300_1.0_0.5_0.125_1	3239	3511	8.4	63	3557	9.8	40	4932	52.3
300_1.0_0.5_0.125_2	3329	3800	14.1	56	3985	19.7	33	5317	59.7
300_1.0_0.5_0.125_3	3841	3851	0.3	32	4085	6.4	59	5658	47.3
300_1.0_0.5_0.125_4	3185	3716	16.7	61	3960	24.3	46	5158	61.9
300_1.0_0.5_0.25_0	3293	3762	14.2	16	3959	20.2	36	5568	69.1
300_1.0_0.5_0.25_1	3371	3778	12.1	34	3934	16.7	50	5271	56.4
300_1.0_0.5_0.25_2	2919	3991	36.7	65	3472	18.9	53	5355	83.5
300_1.0_0.5_0.25_3	3312	3858	16.5	55	3840	15.9	43	5568	68.1
300_1.0_0.5_0.25_4	3042	3573	17.5	50	3543	16.5	60	5079	67.0
300_1.0_0.5_0.5_0	3119	3698	18.6	53	3736	19.8	47	5405	73.3
300_1.0_0.5_0.5_1	3958	4161	5.1	68	4229	6.8	59	5409	36.7
300_1.0_0.5_0.5_2	3797	3953	4.1	12	4136	8.9	42	5895	55.3
300_1.0_0.5_0.5_3	3489	3736	7.1	30	3925	12.5	38	5379	54.2
300_1.0_0.5_0.5_4	3378	3553	5.2	48	3635	7.6	48	5309	57.2
400_0.5_0.125_0.125_0	7396	7415	0.3	60	11000	48.7	63	10087	36.4
400_0.5_0.125_0.125_1	6391	6967	9.0	37	10681	67.1	62	9356	46.4
400_0.5_0.125_0.125_2	6558	6819	4.0	63	10467	59.6	63	9658	47.3
400_0.5_0.125_0.125_3	6424	6807	6.0	64	11055	72.1	62	9521	48.2
400_0.5_0.125_0.125_4	6998	7045	0.7	53	10149	45.0	61	9801	40.1
400_0.5_0.125_0.25_0	6381	6855	7.4	64	10011	56.9	62	9786	53.4
400_0.5_0.125_0.25_1	6387	7360	15.2	72	10106	58.2	60	9471	48.3
400_0.5_0.125_0.25_2	6296	6727	6.8	66	10070	59.9	63	9063	43.9
400_0.5_0.125_0.25_3	7135	7135	0.0	62	11231	57.4	63	9557	33.9
400_0.5_0.125_0.25_4	6457	7226	11.9	64	10332	60.0	62	9799	51.8
400_0.5_0.125_0.5_0	7299	7958	9.0	33	11526	57.9	60	10772	47.6
400_0.5_0.125_0.5_1	6447	7135	10.7	54	10599	64.4	62	9630	49.4
400_0.5_0.125_0.5_2	6795	7407	9.0	67	11060	62.8	61	9383	38.1
400_0.5_0.125_0.5_3	6538	7233	10.6	66	9873	51.0	62	9221	41.0
400_0.5_0.125_0.5_4	6638	6990	5.3	51	9877	48.8	62	9995	50.6
400_0.5_0.5_0.125_0	6568	6952	5.8	65	10776	64.1	62	9576	45.8
400_0.5_0.5_0.125_1	6981	7750	11.0	68	11050	58.3	63	9762	39.8
400_0.5_0.5_0.125_2	7139	7363	3.1	64	11027	54.5	63	10049	40.8
400_0.5_0.5_0.125_3	6253	7275	16.3	66	10428	66.8	62	9427	50.8

400_0.5_0.5_0.125_4	6919	7155	3.4	65	10965	58.5	63	9916	43.3
400_0.5_0.5_0.25_0	6461	6941	7.4	68	10621	64.4	63	9588	48.4
400_0.5_0.5_0.25_1	5911	6631	12.2	68	10039	69.8	60	8980	51.9
400_0.5_0.5_0.25_2	6425	7201	12.1	59	10461	62.8	63	9704	51.0
400_0.5_0.5_0.25_3	6517	7305	12.1	67	11161	71.3	63	9570	46.8
400_0.5_0.5_0.25_4	6360	6751	6.1	42	9616	51.2	60	9534	49.9
400_0.5_0.5_0.5_0	6379	7446	16.7	65	10982	72.2	61	9537	49.5
400_0.5_0.5_0.5_1	6652	7103	6.8	49	10238	53.9	62	9569	43.9
400_0.5_0.5_0.5_2	7357	7410	0.7	61	10589	43.9	62	10061	36.8
400_0.5_0.5_0.5_3	6824	7069	3.6	64	9935	45.6	61	9657	41.5
400_0.5_0.5_0.5_4	6771	6959	2.8	60	10622	56.9	62	9669	42.8
400_1.0_0.125_0.125_0	4477	4667	4.2	75	7636	70.6	62	6777	51.4
400_1.0_0.125_0.125_1	4595	4827	5.0	55	7524	63.7	62	7474	62.7
400_1.0_0.125_0.125_2	4496	6302	40.2	82	9008	100.4	64	7436	65.4
400_1.0_0.125_0.125_3	4223	4791	13.5	69	7691	82.1	60	6619	56.7
400_1.0_0.125_0.125_4	3822	4293	12.3	65	7422	94.2	64	6399	67.4
400_1.0_0.125_0.25_0	4505	5030	11.7	71	7806	73.3	63	7051	56.5
400_1.0_0.125_0.25_1	5017	5063	0.9	73	7648	52.4	60	7371	46.9
400_1.0_0.125_0.25_2	4051	5458	34.7	75	7527	85.8	62	7165	76.9
400_1.0_0.125_0.25_3	4403	5185	17.8	68	8376	90.2	62	7407	68.2
400_1.0_0.125_0.25_4	4930	4979	1.0	69	7415	50.4	63	7461	51.3
400_1.0_0.125_0.5_0	4377	4585	4.8	71	7870	79.8	61	7152	63.4
400_1.0_0.125_0.5_1	4399	5147	17.0	70	7598	72.7	62	6946	57.9
400_1.0_0.125_0.5_2	4451	4697	5.5	73	7838	76.1	61	6685	50.2
400_1.0_0.125_0.5_3	4234	4979	17.6	63	8434	99.2	61	7271	71.7
400_1.0_0.125_0.5_4	3952	4407	11.5	61	7823	98.0	60	7022	77.7
400_1.0_0.5_0.125_0	4509	4695	4.1	67	8425	86.8	61	7458	65.4
400_1.0_0.5_0.125_1	4158	4356	4.8	68	7829	88.3	62	6875	65.3
400_1.0_0.5_0.125_2	4347	4846	11.5	77	7094	63.2	61	6735	54.9
400_1.0_0.5_0.125_3	4010	4601	14.7	66	7758	93.5	63	6999	74.5
400_1.0_0.5_0.125_4	4333	4573	5.5	71	7519	73.5	60	6731	55.3
400_1.0_0.5_0.25_0	4903	4903	0.0	71	8169	66.6	62	7582	54.6
400_1.0_0.5_0.25_1	4530	4706	3.9	72	7939	75.3	62	7452	64.5
400_1.0_0.5_0.25_2	4076	4721	15.8	70	7155	75.5	62	6615	62.3
400_1.0_0.5_0.25_3	4927	5123	4.0	67	8612	74.8	60	7711	56.5
400_1.0_0.5_0.25_4	4307	4760	10.5	64	7814	81.4	61	7167	66.4
400_1.0_0.5_0.5_0	3971	4601	15.9	71	7618	91.8	63	6461	62.7
400_1.0_0.5_0.5_1	4083	4634	13.5	69	7718	89.0	63	6528	59.9
400_1.0_0.5_0.5_2	4608	4894	6.2	70	8288	79.9	61	7130	54.7
400_1.0_0.5_0.5_3	4278	6029	40.9	61	7578	77.1	60	7092	65.8
400_1.0_0.5_0.5_4	4411	5038	14.2	72	7690	74.3	62	7080	60.5
500_0.5_0.125_0.125_0	7867	8467	7.6	64	13987	77.8	67	12265	55.9
500_0.5_0.125_0.125_1	7694	8411	9.3	69	13660	77.5	66	11465	49.0
500_0.5_0.125_0.125_2	8076	9439	16.9	70	13942	72.6	62	11930	47.7
500_0.5_0.125_0.125_3	8083	8904	10.2	69	14891	84.2	64	12003	48.5
500_0.5_0.125_0.125_4	7917	9315	17.7	68	14044	77.4	64	11445	44.6
500_0.5_0.125_0.25_0	8413	9331	10.9	67	14968	77.9	64	12300	46.2
500_0.5_0.125_0.25_1	8572	8946	4.4	69	14585	70.1	62	12928	50.8
500_0.5_0.125_0.25_2	8859	9921	12.0	68	15048	69.9	63	12453	40.6
500_0.5_0.125_0.25_3	7655	9308	21.6	71	14614	90.9	62	11482	50.0
500_0.5_0.125_0.25_4	8454	8875	5.0	67	14514	71.7	66	11995	41.9
500_0.5_0.125_0.5_0	8378	8872	5.9	69	13996	67.1	64	12317	47.0
500_0.5_0.125_0.5_1	8359	9152	9.5	70	14474	73.2	62	11791	41.1
500_0.5_0.125_0.5_2	9038	9471	4.8	68	15435	70.8	65	13094	44.9
500_0.5_0.125_0.5_3	8992	9328	3.7	67	16107	79.1	60	13078	45.4
500_0.5_0.125_0.5_4	9043	9175	1.5	67	15645	73.0	63	13090	44.8
500_0.5_0.5_0.125_0	8130	10150	24.8	72	14591	79.5	64	12505	53.8
500_0.5_0.5_0.125_1	8085	9018	11.5	71	13914	72.1	61	12176	50.6

500_0.5_0.5_0.125_2	8479	8715	2.8	69	14570	71.8	62	12374	45.9
500_0.5_0.5_0.125_3	7791	9382	20.4	69	13285	70.5	62	11603	48.9
500_0.5_0.5_0.125_4	7993	9272	16.0	70	14385	80.0	62	12066	51.0
500_0.5_0.5_0.25_0	8266	9039	9.4	69	13831	67.3	61	11991	45.1
500_0.5_0.5_0.25_1	8090	8458	4.5	66	13612	68.3	66	12087	49.4
500_0.5_0.5_0.25_2	9015	9040	0.3	65	15256	69.2	60	12708	41.0
500_0.5_0.5_0.25_3	9161	9199	0.4	66	14705	60.5	64	12645	38.0
500_0.5_0.5_0.25_4	8080	9116	12.8	68	14220	76.0	65	12110	49.9
500_0.5_0.5_0.5_0	8020	8475	5.7	72	13754	71.5	67	11650	45.3
500_0.5_0.5_0.5_1	7537	9487	25.9	76	14370	90.7	67	11849	57.2
500_0.5_0.5_0.5_2	8404	10105	20.2	66	14273	69.8	65	12534	49.1
500_0.5_0.5_0.5_3	7984	10375	29.9	71	13643	70.9	66	12195	52.7
500_0.5_0.5_0.5_4	8242	9452	14.7	67	14037	70.3	60	12029	45.9
500_1.0_0.125_0.125_0	5196	5951	14.5	65	10583	103.7	60	9220	77.4
500_1.0_0.125_0.125_1	4600	6176	34.3	75	10413	126.4	63	7645	66.2
500_1.0_0.125_0.125_2	4374	5165	18.1	65	9527	117.8	61	7882	80.2
500_1.0_0.125_0.125_3	5012	5674	13.2	67	10264	104.8	67	8407	67.7
500_1.0_0.125_0.125_4	4463	5022	12.5	62	9643	116.1	65	8444	89.2
500_1.0_0.125_0.25_0	6097	6859	12.5	64	10716	75.8	63	9719	59.4
500_1.0_0.125_0.25_1	5724	5816	1.6	60	10898	90.4	61	8889	55.3
500_1.0_0.125_0.25_2	6124	6188	1.0	63	11917	94.6	65	9516	55.4
500_1.0_0.125_0.25_3	5021	5572	11.0	64	9383	86.9	63	7519	49.8
500_1.0_0.125_0.25_4	6414	6463	0.8	60	10529	64.2	65	9388	46.4
500_1.0_0.125_0.5_0	5304	6279	18.4	69	10302	94.2	60	8192	54.4
500_1.0_0.125_0.5_1	5246	5535	5.5	59	10262	95.6	65	9134	74.1
500_1.0_0.125_0.5_2	5138	5470	6.5	66	11557	124.9	61	8706	69.4
500_1.0_0.125_0.5_3	5607	7658	36.6	63	11326	102.0	62	9521	69.8
500_1.0_0.125_0.5_4	5273	5963	13.1	70	10482	98.8	68	8816	67.2
500_1.0_0.5_0.125_0	5766	5767	0.0	59	10845	88.1	64	8341	44.7
500_1.0_0.5_0.125_1	4703	6593	40.2	70	9328	98.3	65	8546	81.7
500_1.0_0.5_0.125_2	5186	6339	22.2	62	10708	106.5	64	8668	67.1
500_1.0_0.5_0.125_3	4938	5588	13.2	66	10095	104.4	62	8908	80.4
500_1.0_0.5_0.125_4	4917	5558	13.0	67	10163	106.7	60	8120	65.1
500_1.0_0.5_0.25_0	5267	5974	13.4	65	11515	118.6	68	8920	69.4
500_1.0_0.5_0.25_1	6670	6670	0.0	58	11029	65.4	65	9595	43.9
500_1.0_0.5_0.25_2	6333	6337	0.1	60	11521	81.9	66	9482	49.7
500_1.0_0.5_0.25_3	5429	6303	16.1	69	10400	91.6	65	8341	53.6
500_1.0_0.5_0.25_4	5659	5997	6.0	63	10713	89.3	61	9393	66.0
500_1.0_0.5_0.5_0	5326	5840	9.7	68	10508	97.3	67	8931	67.7
500_1.0_0.5_0.5_1	5445	5603	2.9	69	10438	91.7	62	8556	57.1
500_1.0_0.5_0.5_2	5873	6160	4.9	61	10374	76.6	64	9604	63.5
500_1.0_0.5_0.5_3	5435	5935	9.2	61	10537	93.9	60	8555	57.4
500_1.0_0.5_0.5_4	5265	5695	8.2	61	11755	123.3	68	9590	82.1

Bibliography

Almeder, C. and L. Mönch. «Metaheuristics for scheduling jobs with incompatible families on parallel batching machines». In: *Journal of the Operational Research Society* (2011).

Bachmann, P. «Analytische Zahlentheorie». In: *Leipzig: Teubner.* (1894).

Blum, C. and A. Roli. «Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison». In: *ACM Computing Surveys* (Sept. 2013).

Boysen, N. and K. Stephan. «A survey on single crane scheduling in automated storage/retrieval systems». In: *European Journal of Operational Research* (2016).

Bozer, Y.A. and J.A. White. «Travel-Time Models for Automated Storage/Retrieval Systems». In: *IIE Transactions* (1984).

Brucker, P. et al. «Scheduling a batching machine». In: *Journal of Scheduling* 1.1 (1998). DOI: 10.1002/(SICI)1099-1425(199806)1:1<31::AID-JOS4>3.0.CO;2-R.

Chang, D.T. and U.P. Wen. «The impact on rack configuration on the speed profile of the storage and retrieval machine». In: *IIE Transactions* (1997).

Della Croce, Federico, Andrea Grosso, and Fabio Salassa. «A matheuristic approach for the total completion time two-machines permutation flow shop problem». In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. Springer. 2011, pp. 38–47.

Emde, S. «Lectures from "Heuristische Planung", Sommersemester 2019». In: *Heuristische Planung. TU Darmstadt* (2019).

Emde, S., L. Polten, and M. Gendreau. «Logic-based benders decomposition for scheduling a batching machine». In: *Elsevier* (2019).

Gagliardi, J.-P., J. Renaud, and A. Ruiz. «Models for automated storage and retrieval systems: a literature review». In: *International Journal of Production Research* (2012).

Gagliardi, J.P., J. Renaud, and A. Ruiz. «On storage assignment policies for unit-load automated storage and retrieval systems. Working paper». In: (2010).

Geiger, C.D. and R. Uszoy. «Learning effective dispatching rules for batch processor scheduling». In: *International Journal of Production Research, 46 (6), 1431–1454* (2006).

Geiger, C.D., R. Uszoy, and H. Aytug. «Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach». In: *Journal of Scheduling, 9 (1), 7–34* (2006).

Goldberg, D.E. and J.H. Holland. «Genetic algorithms and machine learning». In: *Machine learning* 3.2 (1988), pp. 95–99.

Graham, R.L. et al. «Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey». In: *Elsevier. pp. (5) 287–326* (1979).

Graves, S.C., W.H. Hausman, and L.B. Schwarz. «Storage–retrieval interleaving in automatic warehousing systems». In: *Management Science, 23 (9), 935–945* (1977).

Han, M-H. et al. «On Sequencing Retrievals In An Automated Storage/Retrieval System». In: *IIE Transactions, 19:1, 56-66* (1987).

Hansen, Pierre and Nenad Mladenović. «Variable Neighborhood Search». In: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Ed. by Edmund K. Burke and Graham Kendall. Boston, MA: Springer US, 2005, pp. 211–238. ISBN: 978-0-387-28356-2. DOI: 10.1007/0-387-28356-0_8. URL: https://doi.org/10.1007/0-387-28356-0_8.

Harding, J.A., M. Shahbaz, and A. Srivinas & Kusiak. «Data mining in manufacturing: a review». In: *Transactions of the ASME on Journal of Manufacturing Science and Engineering. 128 (4): 969-976* (2006).

Hausman, W.H., L.B. Schwarz, and S.C. Graves. «Optimal storage assignment in automatic warehousing systems». In: *Management Science, 22 (6), 629–638* (1976).

Honghua, Tan. *Knowledge Discovery and Data mining*. Springer-Verlag Berlin Heidelberg, 2012.

Ismail, R., Z. Othman, and A. Abu Bakar. «Data mining in production planning and scheduling: A review». In: *2009 2nd Conference on Data Mining and Optimization, DMO 2009, 10.1109/DMO.2009.5341895* (2009).

Johnson, S.M. «Optimal two- and three-stage production schedules with setup times included». In: (1954).

Landau, E. «Handbuch der Lehre von der Verteilung der Primzahlen». In: *Leipzig: B. G. Teubner.* (1909).

Lawler, E. «Optimal sequencing of a single machine subject to precedence constraints». In: *Management Science* (1973).

Lee, M.-K. and S.Y. Kim. «Scheduling of storage/retrieval orders under a just-in-time environment». In: *International Journal of Production Research* (1995).

Li, X. and S. Olafsson. «Discovering Dispatching Rules Using Data Mining». In: *Journal of Scheduling* (2005).

Linn, R.J. and X. Xie. «A simulation analysis of sequencing rules for ASRS in a pull-based assembly facility». In: *International Journal of Production Research, 31 (10), 2355–2367* (1993).

Maniezzo, V., T. Stutzle, and S. Voss. «Annals of Information Systems: Vol. 10. Matheuristics: hybridizing metaheuristics and mathematical programming». In: (2009).

Mann, Zoltan. «The Top Eight Misconceptions about NP-Hardness». In: *Computer* 50 (May 2017), pp. 72–79. DOI: 10.1109/MC.2017.146.

Material Handling Institute, Inc. «Considerations for Planning and Installing an Automated Storage/Retrieval System». In: *ASIRS Document-100 7M* (1977).

Metan, G., I. Sabuncuoglu, and H. Pierreval. «Real time selection of scheduling rules and knowledge extraction via dynamically controlled data mining». In: *International Journal of Production Research* (2010).

MIT. *Computational complexity*. 2013. URL: www.youtube.com/watch?v=moPtwq_cVH8.

Pinedo, M. *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing, 2016.

Possani, E. «Lot Streaming and Batch Scheduling: splitting and grouping jobs to improve production efficiency». In: *University of Southampton* (2001).

Randhawa, S.U. and R. Shroff. «Simulation—based design evaluation of unit load automated storage/retrieval systems». In: *Computers & Industrial Engineering* (1995).

Roodbergen, K.J. and I.F.A. Vis. «A survey of literature on automated storage and retrieval systems». In: *European Journal of Operational Research* (2009).

Rosenblatt, M.J., Y. Roll, and V. Zyser. «A combined optimization and simulation approach for designing automated storage/retrieval systems». In: *IIE Transactions* (1993).

Schwarz, L.B., S.C. Graves, and W.H. Hausman. «Scheduling policies for automatic warehousing systems: Simulation results». In: *AIIE Transactions, 10 (3), 260–270* (1978).

Sexton, R.S. et al. «Global optimization for artificial neural networks: A tabu search application». In: *European Journal of Operational Research* (1998).

Sha, D. and CH. Liu. «Using Data Mining for Due Date Assignment in a Dynamic Job Shop Environment». In: *The International Journal of Advanced Manufacturing Technology* (2005).

Suthaharan, S. *Machine Learning Models and Algorithms for Big Data Classification*. Springer Science + Business Media New York, 2016.

Witten, I.H. et al. «Weka: Practical Machine Learning Tools and Techniques with Java implementations». In: *Department of Computer Science. The University of Waikato* (1999).

Zäpfel, G. and R. Braune. *Metaheuristic search concepts: A tutorial with applications to production and logistics*. Springer-Verlag Berlin Heidelberg, 2010.