A Heuristic Approach to the Flow Shop Scheduling Problem with Time-Varying Electricity Prices

Author:
Eduardo Salvador Rocha
277584

Supervisors:
Prof. Doc. Julia Rieck
M. Sc. Jan Busse

14th January 2020

Thesis submited for
Master of Science in Data Analytics

Wirtschaftsinformatik und Maschinelles Lernen
Stiftung Universität Hildesheim
Universitatsplätz 1, 31141 Hildesheim

Statement as to the sole authorship of the thesis:

A HEURISTIC APPROACH TO THE FLOW SHOP SCHEDULING PROBLEM WITH TIME-VARYING ELECTRICITY PRICES.

I hereby certify that the master's thesis named above was solely written by me and that no assistance was used other than that cited. The passages in this thesis that were taken verbatim or with the same sense as that of other works have been identified in each individual case by the citation of the source or the origin, including the secondary sources used. This also applies for drawings. sketches, illustration as well as internet sources and other collections of electronic texts or data, etc. The submitted thesis has not been previously used for the fulfilment of a degree requirements and has not been published in English or any other language. I am aware of the fact that false declarations will be treated as fraud.

14.01.2020, Hildesheim, Lower Saxony, Germany
Eduardo Salvador Rocha

Abstract

Energy as a major cost factor has become increasingly important for manufacturing companies. Due to current developments, such as smart metering, the internet of things, and varying pricing schemes, companies can increase energy efficiency and reduce energy costs. Based on suitable electricity price forecasts, it is possible to set up an energy cost-oriented schedule for a planning horizon of varying days. This thesis proposes a heuristic scheduling model that minimizes the overall operational cost in an Flow Shop Scheduling Problem under RTP tariffs. The core of the model relies on two elements: specific scheduling rules that build upon sorted time indexes of the dynamic tariff and dispatching rules applied to the jobs. The model also can be easily tailored to minimize other objectives such as the makespan. Furthermore, by encoding the jobs as permutation sequences, the model can be optimized through a genetic programming approach, which also opens the possibility to use the NSGA-II heuristic to perform a multi-objective optimization. The obtained results are compared with the MIP model introduced in [9].

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In any business always exists sets of processes or tasks that need to be completed within a time frame, and finding them a right order can lead to considerable benefits. This ordering problem is also known as scheduling and is especially important for industries that are related to the production of consumer goods, whose demand grows in parallel to the world's population, estimated to increase from 7.7 billion in 2019 to around 8.5 billion in 2030 [32]. In this regard, the industry sector also requires a considerable amount of energy, and as the demand for consumer goods increase, the energy consumption does too. For example, in 2018 the use of primary energy grew at a rate of 2.9%, almost double its 10-year average of 1.5% per year, and the fastest since 2010 [7]. While the majority of the European countries showed a decline in their energetic consumption, just China, the US, and India together accounted for more than two-thirds of the global increase in energy demand, with US expanding at its fastest rate for 30 years [7]. Although there is an increased effort to find profitable and efficient sources of renewable energy generation, these alternatives strongly depend on weather conditions and geographic location, and therefore, they are not reliable. That is the reason for fossil fuels still representing the source of about 85% [3] of the total energy consumed worldwide, which raises concerns regarding fossil fuel depletion and greenhouse gas emissions. To illustrate, carbon emissions from energy use in 2018 grew by 2.0%, (around 0.6 gigatonnes) [7], and specifically, the industry sector generated more than 50% (approximately 15.6% Gt) of the total emissions of CO_2 worldwide [6]. Given a growing societal awareness on climate change along demands for urgent action, an increasing cost of energy, and increasing energy demands from developing countries; a sustainable use of resources is essential.

To meet the gap between demand and supply in electricity generation, most efforts have been made in increasing the production capacity of existing power plants and in reducing the overall energy consumption by creating more electricity-efficient machines. These options require a considerable size of capital investment and creating new energetic facilities usually affects even more the environment. Instead, diverse studies have demonstrated that power use and overall production cost can be reduced through demand-side management (DSM) [3] [12] which includes energy efficiency (EE) and demand response (DR) programs. The intuition behind those policies is to reduce energy consumption by shifting the load during peak hours to take advantage of lower prices during off-peak hours. Energy-aware (EA) manufacturing belongs to EE programs, and its objective is to minimize the total energy consumption at the operational level. Likewise, energy-cost-aware (ECA) manufacturing is part of DR programs, and also aims to minimize the total energy cost but specifically considering time-varying energy prices. Time-of-use (TOU) pricing, critical peak pricing (CPP), and real-time pricing (RTP) are 3 time-based DR programs commonly available to industrial customers. Both are implemented through optimal job scheduling or lot sizing of the production plan [17].

However, the research done in the ECA area has focused more on reducing electricity costs of residential and commercial buildings, than of manufacturing facilities [4]. This is due is easier to schedule "shiftable" appliances like lights, washing machines, etc., than processing jobs that cannot be interrupted randomly. On top of that, as electrical distribution is moving towards a more intelligent, reliable, stable and secure electrical systems, the dynamic interactions between factories and electricity market prices have to be considered when developing new manufacturing schedules. Still, job-shop scheduling is an NP-hard combinatorial problem [8] on which classical scheduling methods scale up poorly. For example, to find an optimal solution, exact scheduling algorithms based on either mixed-integer programming (MIP) or branch and bound methods, face great challenges in terms of computational stability and time. To solve this issue, approximate scheduling algorithms based on priority dispatch rules or meta-heuristics are used to quickly find a feasible solution, but in most cases, the global optimum is not guaranteed (sometimes even exact methods can not find it in a reasonable time). [36]

This thesis proposes to use a heuristic algorithm and different dispatch rules to solve a job flow-shop scheduling in a cost-oriented minimization problem, as an alternative to reduce the time to find a solution using MIP. The heuristic algorithm is then further optimized using a genetic programming

approach on two typical objectives: makespan and cost. In this research, a job is composed of up to 3 sub-steps, each to be scheduled in a different machine, following a preceding ordering rule. The final operational cost is then minimized over periods of 7, 14 and 28 days, considering the price fluctuation of electricity over such time. The efficiency of the heuristic algorithm is compared against the MIP solution in terms of the final cost and the final makespan.

Types of Job Scheduling

The classical problem of job shop scheduling (JSP) deals with assigning a set of jobs $J_k, (k = 1, 2, ..., n)$ to machines $M_l, (l = 1, 2, ..., m)$ in a particular time or order trying to minimize the total completion time required for all jobs (also known as the makespan). The components of a scheduling problem could convey various requirements such as raw materials, machinery, equipment, operators, or other characteristics; they even can be divided into smaller sub-components that may or may not share the same requirements.

In the simpler case the constrains to meet are few; there is no a specific order between two or more jobs, the processing of a job cannot be interrupted, and each job can be performed only on one machine at a time (and vice-verza). Thus, a *schedule* specifies when each task and their sub-components is to be processed. If a schedule meets all the constraints given in the problem, it is called $feasible$ and, as the nature of the problem is combinatorial, multiple feasible solutions might exist. Conversely, a schedule is denominated *optimal* if it is feasible but also optimizes one or (possibly) several objectives [14].

Nevertheless, calculate an optimal schedule is often difficult. As the complexity of the problem increases (e.g. increasing the number of jobs, the number of machines or having more than one objective to optimize) classical solution procedures start to perform poorly and most of the times, cannot be directly applied to complex manufacturing structures. Moreover, in cases where reaching optimality requires high computing time or is just unattainable, they start to rely on heuristic alternatives.

To have a clear picture of the distinct solving algorithms for JSPs, one must first know in which scenarios to apply them. On this matter, JSPs can be classified using diverse criteria, just like the sources of production demand, the number of machine tools, performance index, characteristics of production environments, processing characteristics of operations, plant involved and resource constraints [36]. However, one of the most used principles of

classification is based on the problem space (or structure) they solve:

- A Basic JSP: This is the simplest case where only one machine process all the jobs.
- Multi-machine ($FJSP$): Also known as the $Flow$ shop problem; in this scenario, multiple machines process the jobs. If all of the machines can be chosen it is called a "complete FJSP" otherwise, if only some of them can be chosen it is called a "partial FJSP".
- Hybrid Multi-machine (HFS): Similarly as in an FJSP, this scenario accounts for multiple m machines divided in m_i steps or levels, to process all jobs. Also known as the hybrid flow shop problem.
- Multi-resources ($MrFJSP$): Additional to the inputs and constraints of FJSP, plant layout information and plant resources (e.g. tools, dies, fixtures, operators, vehicles, robots), are also restrictions the model. The scheduling has also to consider transition times between the assignations of jobs and resources to a process.
- Multi-plant ($MpFJSP$): Based on MrFJSPs, this case also takes into account multiple plants and transportation times between them. This kind of model is one of the most complicated and dynamic cases in nowadays scheduling scenarios, especially because centralized scheduling systems have great difficulties calculating optimal solutions. Multiple agents are often used in a multi-plant environment to deal with collaboration scheduling between plants, and therefore, this type of model can be considered as semi-distributed scheduling.
- Smart factory type ($SFFJSP$): In a manufacturing environment, dynamic changes like emergency events, new orders, canceled orders, and machine failures, occur in real-time. Centralized scheduling and semi-distributed scheduling do not handle these scenarios efficiently, and often a reschedule cost more time and causes longer recovery periods with severe disruptions to a manufacturing process. This type of scheduling uses a fully distributed system supported by novel and emerging manufacturing technologies like Big Data, the Internet of Things, Artificial intelligence, etc.

The most frequently objective to optimize in all these cases is either the makespan or the overall production cost. This thesis is focused on solving a cost-oriented flow shop problem, and therefore the following subsections will be directed to topics related to this application area.

FJSP: Solving Methods

Depending on the size and purpose of the scheduling, the methods to solve FJSPs (and JSPs in general) can be divided into mono and multi-objective methods, distributed and centralized methods, or exact and approximate methods. That classification is not mutually exclusive, for example, an approximate method can be used to solve a multi-objective schedule in a distributed environment. For this section, the attention will focus on exact and approximate methods, the most widely known classification.

Exact Methods

Since 1959 JSPs were solved to optimality employing mixed discrete integer and linear programming (MIP) approaches, which take advantage that JSP can be formulated using a linear objective function, a series of linear constraints and decision features modeled as binary integer variables [33]. Such approaches were quickly outperformed by emerging enumerative techniques known as Branch and bound (B&B), which consist of listing all feasible solutions to the combinatorial problem. This is also known as an enumeration tree, and branches of it are defined by specific attributes and represent subsets of solutions that do not intersect. Within a branch, the objective value of the best solution is lower bounded (bounding) via relaxation, and in case the lower bound exceeds the value of the best (smallest) known upper bound, the current branch can be dropped from further consideration. Otherwise, the search is continued by dividing the current attribute into smaller subsets through the definition of additional attributes. At the end of the process, the optimal solution found is characterized by a set of satisfied attributes [10].

Without doubt, *branch and bound* (B&B) is the preferred technique when solving most HJSPs, but the majority of the research done so far, however, has concentrated on simplified versions of the problem. This imposes a paradigm because a nm JSP problem has $(n!)^m$ possible solutions, and although B&B, as well as most exact optimization methods can, in theory, obtain the optimal solution, for large-scale situations is not possible to complete calculations in a good responding time. This further limits its real-world applications.

Heuristic (Approximate) Methods

The advances in computer technology and intelligent algorithms allowed research methods related to JSPs, to gradually shift into approximation methods. They are based on constructive techniques and their main characteristic

is that they can find the JSP solution with dramatically reduced computation and time. The simplest type of heuristics are dispatching rules, also known as scheduling policies or list scheduling algorithms, and include three typical methods: priority dispatch rule, insert algorithm and bottleneck based heuristics.

Priority dispatch rules, as the name suggests, make use of precedence criteria to process the jobs in a sorted fashion. Typical rules include the shortest processing time, the longest remaining total processing time, the earliest delivery time and the selection of the same machine on the first working process. Dispatching rules have proven to be highly effective dealing with complex, dynamic, and unpredictable environments, as they can be easily tailored by just selecting the best fitting priority rule. For example, if reducing the average flow time of all jobs is the most important, the shortest processing time rule can be chosen, or if optimizing the maximum delay is the most important, the earliest delivery time rule should be used instead. It is also common that several priority dispatch rules are either built simultaneously to achieve an optimal solution. But one major observed drawback for highly complex scenarios is that often infeasible solutions might be generated if the number of rules is increased, as they begin to be mutually restrictive, trapped in a loop, and even contradictory at times [36].

In *Insertion methods*, firstly a pre-schedule is obtained through a heuristic algorithm and then additional tasks such as maintenance are scheduled. A more sophisticated version of this algorithm uses a divide-and-conquer strategy in which the original problem is divided into smaller subproblems that are solved one at a time and their solutions are integrated into a whole solution to the original problem. The idea of separating the sequencing and machine assignment has been exploited in many studies [30]. Another particularly effective divide-and-conquer approach for the m-stage problem are the variants of the shifting bottleneck procedure (SBP). The shifting bottleneck heuristic consists of two subroutines, the first one repeatedly solves one machine scheduling problem, while the second one builds a partial enumeration tree starting with the solution found by the first subroutine [10]. Under this scheme, SBP always schedule "bottleneck" machines first.

Meta-heuristic (Approximate) methods

In comparison with the heuristic methods, a metaheuristic approach includes an element of randomness in a neighborhood search. The idea is that its repeated usage may lead to better solutions than the ones generated deterministically. Randomness is integrated into the acceptance criterion allowing the

search to deviate from local optima so as to direct the search to other areas of the search space [30] [14]. Simulated annealing (SA), tabu search (TS), and genetic algorithms (GA) are the three most widely known metaheuristic algorithms used in FJSPs.

Simulated Annealing (SA) is a methodology that simulates the annealing process used in physics to cool solids slowly until they reach a low energy state. It utilizes two ways to improve upon an incumbent solution: interchanging the positions of a pair of jobs in the permutation, or moving one job in a pair to a position right after the other. Both types of job movements are examples of descent algorithms that seek to reduce the makespan value. The first is denominated as a swap and the second as an insertion. Swaps and insertions may be done in a predetermined order that captures all n^2 possible pair combinations, or randomly [22].

Tabu search based algorithms (TS) starts with a carefully selected initial solution. The incumbent solution is transformed into another solution through appropriate moves like swaps, insertions, etc., chosen from a neighborhood. To avoid cycling through the same or similar solutions, a tabu list is created that includes solutions that are prohibited. The tabu list allows TS to exit local minima in hopes of reaching alternative, hopefully, better optima. The algorithm terminates when an iteration or time limit is reached [14].

Genetic algorithms (GA) belong to the larger class of evolutionary algorithms (EA), which generate solutions usually represented by a bit array. In a basic sense, GA is a search heuristic that mimics the process of natural evolution. It initializes the optimization problem by producing a determined set of feasible solutions, from which afterward a set of "parent" solutions is selected to breed (combine) until a new generation (set of solutions) of appropriate size is achieved. New generations typically share many of the characteristics of its "parents" and they are created as long as a "child" does not meet the optimal value of the fitness function or a stopping criterion. Two of the most popular types of genetic operators used to produce a child are crossover and mutation. There are many types of crossover operators, and a simple crossover method can consist in simply choosing randomly a point on two "parent" solutions (designated a 'crossover point') and proceed to swap bits to the right(or left) of that point between the two parents. Similarly, there is a variety of mutation operators. The classic example involves a probability that an arbitrary position in a solution will be changed from its original state. Besides, a small proportion of less fit solutions are included, for reasons of diversity to avoid getting trapped in local optima [11].

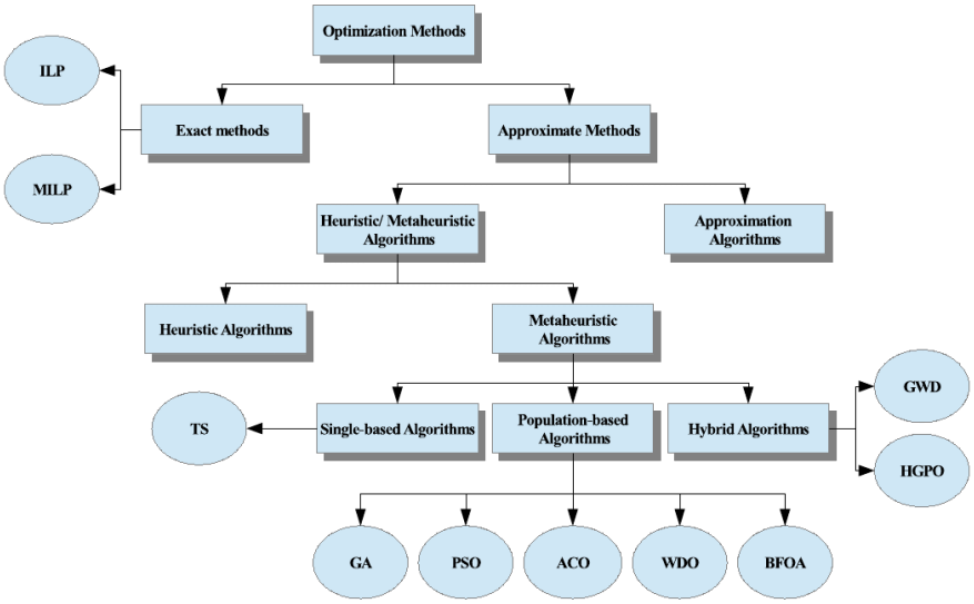

Figure 1.1: Types of schedules. Reproduced from [24]

Other meta-heuristics that have also been used in HFS are ant colony optimization (ACO), artificial immune systems (AIS), neural networks (NN), etc.

Demand Response Programs

From the first DC transmission line installed by Thomas Edison in 1882 to nowadays super large-scale complex AC-DC hybrid power grids, power systems have evolved with the progress of human industrial civilization [26]. At present, the demand for energy is increasing due to the multitude of home and residential appliances that require a considerable amount of power to operate, as well as the huge quantity of energy required in the industry sector. As a consequence, the optimal use of electrical resources plays an important role in nowadays society, considering that old power grids fail to fulfill the user requirements due to the primitive nature of their infrastructure [24].

To overcome such challenge, more efficient, controlled, reliable, and safer systems known as smart grids (SGs) have been replacing old power grids. SGs are incorporated with new technologies, advanced software for data management and intelligent controllers, to enhance the delivery network by reducing power demand during peak periods, minimizing in this way the cost of power

production. SGs can also include renewable energy sources (RESs) to generate power and thus, reduce the ever-present reliance on fossil fuels to generate energy. In such capacity, SGS represents the next generation of energy grids that can improve, manage and fulfill the present energy demand.

Optimizing power consumption plays a lead role in achieving the objectives of SGs and leads to a reduction in electricity bill (EB) of users as the major benefit. This power consumption can be achieved by scheduling appliances in residential areas and shifting the working load in industries from peak to off-peak periods using a Demand response programs (DR). DR has two types of schemes, programs with time-based pricing (dynamic prices) and programs incentive-based.

Incentive-based programs

Incentive-based programs include those that offer fixed or varying incentives to users to reduce their power usage during stressed or peak periods of a power system. The response of the users to these programs is optional, but several programs have heavy penalizations in electricity costs in on-peak periods [24].

A *Direct Load Control (DLC)* program permits power supply companies (PSCs) to remotely turn off the electricity supply for appliances and machines. This is feasible through switches placed within the premises of the facilities under this program, which allows PSCs a direct control. Incentive payments are given in advance to users who participate in DLC, to motivate them to maintain their power consumption within predefined thresholds [24].

Interruptible/Curtailable program (I/C) offers advanced incentives, such as rate discount or bill credit, to motivate users to curtail part of their total power consumption to reduce load during peak periods. However, several penalties are imposed in case I/C terms are not respected. Examples of this program can be found in Iran, wherein 2007 a study presented several scenarios for a power system grid under this scheme [1].

Capacity Market Program (CMP) is offered when a predefined power consumption curve is provided. This helps PSCs to predict the power required to be generated. CMPs are typically offered by wholesale market providers such as Independent System Operators (ISOs) that operate installed capacity markets and are the organized market analog of Interruptible/Curtailable tariffs. Eligibility to participate in this program is based on a demonstration that the reductions are sustainable and achievable. For example, the requirements to receive CMP payments in NYISO are: a minimum load reduction of 100kW,

minimum of four hours reduction, two hours notification, and to be subject to one test or audit per capability period. These requirements are designed to ensure that the reductions can be counted upon when they are called [1].

Time-based programs

Pricing schemes based on time tariffs allow users to choose periods of electricity use without curtailing power consumption. Time-based pricing is also known as dynamic pricing as a different price is provided on different periods. On-peaks periods are characterized by higher prices due to high power demand obliges PSCs to use additional power plants to meet user requirements. On the contrary, off-peaks are characterized by low electricity prices due to low consumer demand. The goal of these schemes is to encourage users to shift their load from peak to off-peak periods to reduce power demand [24].

Time-of-use (TOU) is a dynamic pricing scheme that can provide either two prices scheme (off-peak and peak periods) or three prices (low, mid and high peaks) during a day. The price curve of TOU is previously determined for a quarter of the year [24].

Critical Peak Pricing (CPP) is a scheme similar to TOU as it provides two different prices between off-peak and peak periods to balance power demand. The main difference is that is only used when the demand is extremely high compared with other power periods. For example, CPP is declared only on days that are forecasted to have a very high power demand period, called a critical period (CP) versus the costs provided by TOU. In general, the CPP price curve is announced a day before CPP for 15 days in a year [30]. CPP plays a major role in balancing power demand in a day by generating very high EPs during CP, and users will typically prefer to shift their power demand out of CP [24].

Real-Time Pricing (RTP) is a pricing scheme that provides EPs that are nearest to the real generation price during a certain period. EP provided in this pricing scheme changes dynamically every hour. Two types of RTP used by PSCs are day-ahead pricing and hourly pricing. For the day-ahead pricing scheme, PSCs provide EP to users 24 h beforehand. For hourly pricing, EP is provided every hour. Day-ahead pricing is more effective than hourly pricing because users are given sufficient time to schedule their power consumption. The RTP, particularly the day-ahead pricing scheme, along with the TOU pricing model, are the widest schemes used to scheduling machines and appliances to minimize electricity costs [24].

Chapter 2

Related Work

Energy Efficient (EE) manufacturing aims to reduce energy consumption by adjusting the managerial parameter of the production process. It can take into account traditional Production Planning (PP) objectives, such as the minimization of inventory holding cost, setup cost, or total completion time, but also some less explored energy-related ones, such as the minimization of energy consumption or energy cost. EE machine scheduling can be used to model such PP objectives to foster energy efficiency on existing manufacturing equipment in a mid-term and short-term planning level [5]. In this scenario, the power scheduling problem involves allocating a set of jobs to machines (e.g., home appliances, industrial devices) formulated to be constrained to a time horizon and (sometimes) an electricity curve.

Several optimization algorithms have been adapted to handle this kind of problem in all levels of complexity of the JSPs. Nevertheless, most of the work done in energy-efficient scheduling tackles the cost and energy optimization aspect of the problem as just to reducing the non-processing time on machines. This can be either in the form of minimizing either the maximum flow time or their maximum tardiness of the jobs. But the number of researches that implicitly include a dynamic electricity price tariff in their objective function or solving model, are less common.

Extensive research performed in [18] presents an overview of the type of industrial sectors, applied energy sources and relative usage of objective criteria (monetary and non-monetary) that are typically covered in EE models. The monetary aspect of the researches can be summarized by the criteria of profit, energy costs, and penalty costs (i.e. economical fees imposed by government regulations). The non-monetary criteria contain traditional scheduling objectives such as makespan, total (weighted) tardiness, maximum tardiness, production volume, and demand fulfillment. It is shown

that the top usage in more than 50% of the reviewed literature, is directly related to a monetary need in the form of energy cost due to an electric source.

The survey also summarizes the objective criteria according to multi-objective (MO) and single objective (SO). Multi-objective criteria are present in only up to 35% of the researched articles, while the rest represent EE approaches with a single, cost-oriented, objective function (SO). Concerning solving methods, linear programming approaches (MILP & ILP) are preferred for single machine setups and small number instances due to optimal solutions are achievable in a reasonable time. For multi-machine setups and an increased number of instances, metaheuristic methods (GA, ACO, TS, etc.) are preferred [30].

Energy-Efficient Scheduling solved by Linear Programming Approaches

In [28], the authors solved an FJSP in a two-machine setting. A MILP approach was used for small instances, and directly reduced the total energy cost (TEC) by including a variable to sequence jobs and determine feasible schedules, and a variable that directly accounts for the prices given by the TOU tariff. A 2-stage Heuristic was used for larger instances and it can be compared as lot sizing. In the first stage of the heuristic, the time horizon of the TOU tariff is divided into several segments of varying priority; afterward, jobs distributed in those intervals are scheduled in the boundaries of each segment. The second stage schedule the remaining jobs in the idle times of each cost interval.

To solve an FJSP without job precedence rules, two integer programming models, namely a disjunctive and a time-indexed formulation, are proposed in [25]. The objective was to find a schedule that respected a power limitation during the spanning horizon with a variable electricity cost profile while minimizing the energy cost. The time-indexed formulation outperformed the disjunctive one and also proved to be more generic. Non-constant energy cost profiles and machine power consumption profiles in the disjunctive model demonstrated to be a hard task to consider. For more complex scenarios, a heuristic algorithm using the time-indexed model was developed; it consisted of reducing the number of available time slots to deploy operations. This was done by aggregating time slots located at the end of the time horizon; this provides warm start solutions to the B&B algorithm to accelerate its convergence. Similarly, in [15] a two-machine FJSP was solved using an MO-MILP, where makespan and peak power consumption were considered

simultaneously. A key feature of their formulation were variables and constraints that kept track of jobs running concurrently across machines.

The authors in [17] considered intermittent renewables sources, energy storage, and real-time electricity pricing to minimize the energetic cost in a two-stage SMILP model. In the first stage, to minimize the electricity purchase cost, the production scheduling is performed using the day-ahead energy demand curve and the energy forecast of the renewable source. The second stage is designed to minimize total real-time electricity supply costs and to resolve the mismatch between the forecast and the actual energetic production of the renewable source. Overall, the case study shows different degrees of cost reduction, ranging from 8.1% to 48.6% for solar supply, 11.3% to 68.3% for wind supply, and 9.3% to 59.0% for aggregated wind and solar supply. Another research that made use of MILPS models and renewable energy sources is presented in [35]. The EC reduction was done by taking full advantage of an on-site installed wind energy source and dynamic scheduling under RTP tariffs. The problem was modeled as an FJSP and considered the electricity grid of the wind turbine. Both, the production schedule and energy supply decisions, were updated dynamically as wind speed and electricity price data became available. In [9], a MILP scheduling model was proposed to solve a 3-machines FJSP with the objective to maximize the gross with priority on jobs with a high rate of energy costs versus their sales revenue within low RTP intervals. The main advantage of this model is that it provides a lot tailored options depending on how the priority on jobs is made, allowing fine adjustments, i.e., load profiles and electricity-prices in accordance with the length of a time period l, the possibility to maximize the total or job-specific gross profit, or the assignment of a job's stage to different resources.

Energy-Efficient Scheduling solved by meta-heuristic approaches

In multi-objective optimization (also known as Pareto optimization) as the name suggests, more than one objective function has to be optimized simultaneously. In practice, it is often found that the objective functions conflict to each other, therefore a (possibly infinite) number of Pareto optimal solutions exists. A solution is called nondominated, Pareto optimal, Pareto efficient or non-inferior, if none of the objective functions can be improved in value without degrading some of the other objective values [27].

Multi-objective evolutionary algorithms (MOEAs) based on non-dominated sorting are widely used to solve FJSP problems. As the optimization of en-

ergy cost can be easily included in the formulation of these models, either through the evaluation of the fitness function in population-based algorithms or explicitly in the objective function, Evolutionary Algorithms (EAs) can also be effectively applied to EE problems. One of the most popular EAs for energy reduction is the Non-dominated Sorting Genetic Algorithm II (NSGA-II). Their non-dominated sorting approach has a computational complexity of $O(MN^2)$, and is able, for most problems, to find a much better spread of solutions and better convergence near the true Pareto-optimal front compared to the Pareto-archived evolution strategy and the strength-Pareto evolutionary algorithm, two other elitist MOEAs that pay special attention to creating a diverse Pareto-optimal front [13].

Authors in [38] used the NSGA-II to address makespan and total production EC in an FJSP with no precedence constrains between jobs (i.e. jobs did not visit each machine in a particular order). Reduction in the EC was a direct result of turning machines off in Idle times between jobs, specified as a "setup" operation. Their experiments illustrated that the NSGA-II can simultaneously optimize machine selection and operation sequencing. In the same direction, authors in [29] also used an EA-MOA for solving an HFS problem by minimizing EC and makespan. The reduction in EC was measured with the decrease of the processing duration for all machines. The singularity in this research was the representation of each solution by two vectors (the machine assignment priority vector and the scheduling vector), and the specificity in the decoding rules used to perform the scheduling. The GA model was fine-tuned by carrying out an extensive hyperparameter search: four types of decoding; two efficient crossover operators, namely, Single-point Pareto-based crossover (SPBC) and Two-point Pareto-based crossover (TPBC); and eight neighborhood structures plus an adaptive neighborhood selection method, for offsprings production. A final adjustment of the jobs was performed by a right-shifting procedure. In [2], they also used a MOGA for EC and makespan minimization, but in a simpler scenario of a two-machine FJSP. They use a specific decoding algorithm that leverages in the variable speed of operations and in the priority rules given for the jobs. The results shown that MOGAs hybridized with constructive heuristics outperform both, the regular MOGA and heuristics alone, in terms of quality and cardinality of the Pareto frontier. In a similar fashion, the research done in [39] handles a dynamic FJSP by constructing specific decision rules for job sequencing and a machine assignment. The GA simultaneously evolves both rules to solve weighted tardiness, maximum tardiness and mean flow time.

To solve an EE-HFS problem with limited buffers, a MO-Evolutionary

Algorithm based on decomposition (MOEA/D) is proposed in [21]. The main idea is to divide the MO optimization problem into several single-objective sub-problems by using a scalar function. The objectives minimized are the total weighted tardiness (TWT) and non-processing energy (NPE). The model uses a simple job-permutation vector to represent the scheduling solution and a two-pass procedure composed of a discrete-event system (DES) simulation and a greedy post-shift procedure to decode found solutions. The HFS problem in a distributed (multi-factory) environment was studied in [16]. The EE was achieved through makespan minimization and constraining the idle times to be zero. For this purpose a MO-Collaborative Optimization Algorithm (MO-COA) was proposed, and used two heuristics, on for population initialization to guarantee certain quality and diversity and one for multiple search operators to enhance the exploration adaptively. The algorithm was further enhanced by using a local intensification strategy for the dominated and non-dominated individuals (to improve the exploitation), and by a speed adjusting strategy for the non-critical operations (to improve total EC). In [31], to solve the HFS problem, a multi-phased iterated local search algorithm (ILS) was developed to determine a three-dimensional Pareto front, constructed using three objectives: makespan, total energy costs, and peak load. Due to the multi-criteria objective function, a lexicographic approach was used to illustrate the influence of different priority orders on the optimal schedule. Then, the epsilon method served to identify the optimal Pareto front for small problems, and to compare them with the solutions found by the heuristic model. Tabu lists, a right-shifting procedure, a reference point based fitness function, and several time-energy listing scheduling algorithms were used to obtain high-quality solutions. Their computational study analyzed the interdependencies of objectives and compared the proposed algorithm to the NSGA-II heuristic, from which the ILS has proven to be suitable in purposeful search in the solution space.

The research done in [20] shows the use of a genetic algorithm to minimize the overall EC of an mk-machine scheduling problem (HFS) under a TOU tariff. Three states for each machine (stand-by, processing and shut down) were encoded (and optimized) in the fitness function. For each sub-level k a local GA was used to find a solution, further optimized by shifting the scheduled operations to off-peak periods. The proposed Single Objective Genetic Algorithm (SOGA) model used in [19] minimizes directly the EC of an FJSPs, but their main contribution is that they show how the GA can reduce the EC by alternating between multiple energy sources (compressed air, natural gas, and electricity) and five machine energetic states (off, stand-by, Idle, Set-up, and working) in response of the dynamic pricing scheme of

all energy sources. Authors in [37] used a Pareto Evolutionary Algorithm (SPEA2) and a novel hybrid three-level structure gene encoding to solve an HFS problem under TOU tariff. The first level of the chromosome encoded the processing sequence of each job, the second level specified the selected machine tool, and the third level denoted the starting time of each stage which was arranged from small to large according to the sequence of the job. Assuming that the operation of processing was uninterruptible, once the start time of the process was determined, the corresponding electricity tariff period, the Pareto Frontier of the makespan, and ECs were immediately determined.

Given the characteristics of population-based algorithms, they can also be easily parallelized. Tasks that can undergo parallelization are the calculation of the fitness function for each individual within a generation, and the offspring procedure from selected parents. For instance, in [23] an HFS problem under TOU tariffs was solved using a GA designed with the NVIDIA CUDA software. The proposed model incorporated a predictive reactive complete rescheduling strategy to address the variation of peak power electricity values and the arrival of new jobs.

This thesis proposes a heuristic scheduling model that minimizes the overall operational cost in an FJSP under RTP tariffs. The core of the model relies on two elements: specific scheduling rules that build upon sorted time indexes of the RTP tariff and dispatching rules applied to the jobs. The model can be easily tailored to minimize other objectives such as the total flow time and the makespan. Furthermore, by encoding the jobs as permutation sequences, the model can be optimized through a GA, which also opens the possibility to use the NSGA-II heuristic to perform MO optimization. The final results are compared with the MILP model introduced in [9].

Chapter 3

Methodology

Problem Setting

The initial setting of this problem takes some part of the model proposed in [9]. Let $\mathcal{T}$ be the planning horizon divided in equidistant time points $t = \{1, ..., T\}$, $\mathcal{P}$ the set of electricity prices of the RTP tariff with $p_t = \{p_1, \dots p_T\}$ as the specific price at time step t. Let $\mathcal{J} = \{1, ..., J\}, j \in \mathcal{J}$, be the finite set of jobs to be processed in a manufacturing environment. Each job j has to be processed on a set of stages $S_j = \{1, ..., S_j\}$, $S_j \leqslant 3, s \in S_j$, in an increasing index order (e.g. starts in the stage with index number 1, proceeds to stage indexed as 2, etc.). In this setting, jobs may exist that do not require processing on all stages, i.e. stages can be skipped, but at least one stage must be passed.

The energy demand of job j on stage $s \in S_j$ varies during the whole processing time $tp_{j,s}$ of j in s. As a simplification strategy, the energy demand of job j on stage s is divided in $1 \leqslant M_{j,s} \leqslant 3, m \in M_{j,s}$ micro-steps, corresponding to a starting phase, execution phase and shutdown phase.

In Figure 3.1, an energy load profile of a job j at a specific stage s is visualized. The fluctuating energy demand is displayed with a dotted line, the constant energy demand $ec_{j,s,m}$ in micro steps $m = 1, 2, 3$ is shown with a solid line, and the overall constant energy demand $ec_{j,s}$ is shown with a dashed line. The vertical lines at the beginning and end of each micro-step m, represent the starting ($i_{j,s,m}$) and completion ($c_{j,s,m}$) times of job j at stage s in micro-step m, respectively; note that the completion time of a micro-step is equivalent to the starting time of its consecutive micro-step. The processing of a job j in stage s cannot be interrupted.

Consequently for the processing time $tp_{j,s,m}$ of job j on stage s in micro step $m \in \mathcal{M}_{j,s} = \{1, ..., M_{j,s}\}$ the condition $\sum_{m=1}^{M_{j,s}} tp_{j,s,m} = tp_{j,s}$ is satisfied, where $tp_{j,s}$ represents the total processing time of j on s. The notation used in this thesis is summarized in table 3.1

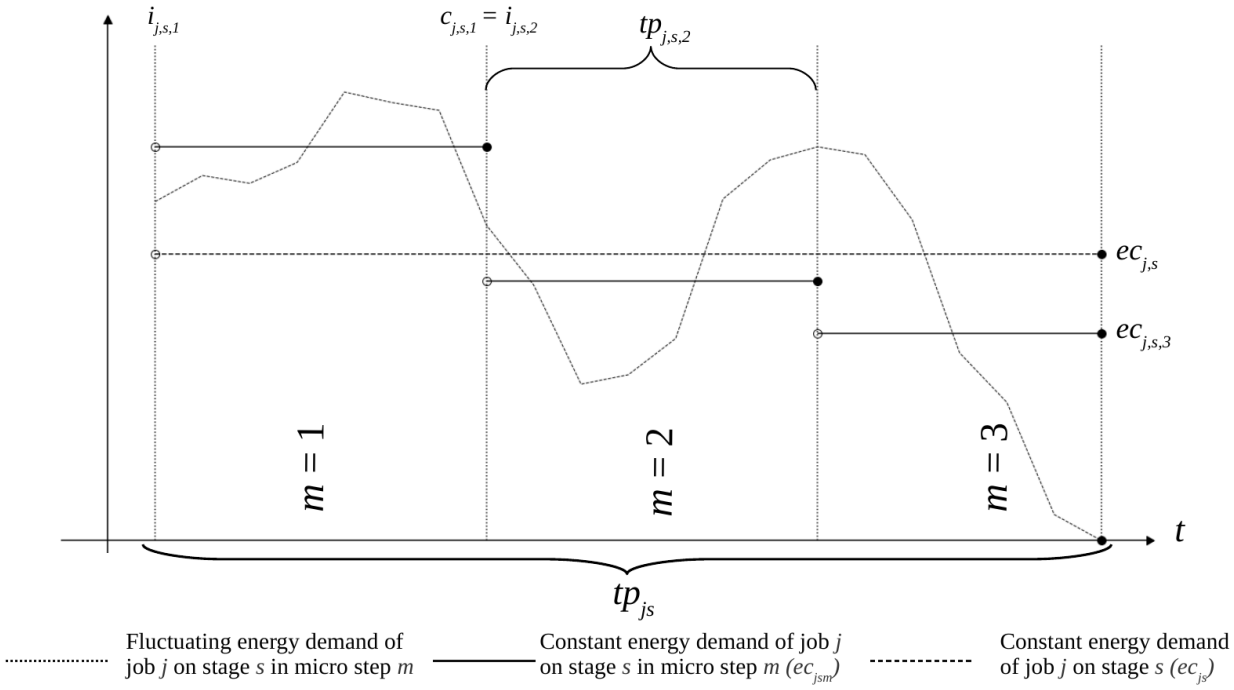

Figure 3.1: Energy Load Profile for a job j in stage s

The total energy demand ed_j for job j during its entire manufacturing process can be calculated as follows:

$$ed_j = \sum_{s=1}^{S_j} \sum_{m=1}^{M_{js}} ec_{jsm} \cdot tp_{jsm} \tag{3.1}$$

The average energy demand of job j is defined by:

$$\bar{ed}_j = \frac{ed_j}{\sum_{s=1}^{S_j} \sum_{m=1}^{M_{js}} ec_{jsm} \cdot tp_{jsm}} \qquad \forall j \in \mathcal{J} \tag{3.2}$$

Table 3.1: Notation

sets	
$j \in \mathcal{J}$	set of jobs, $\mathcal{J} = \{1, \dots J\}$
$s \in \mathcal{S}_j$	stage of job j, $\mathcal{S}_j = \{1, \dots S_j\}$
$m \in \mathcal{M}_{js}$	micro-step of job j, $\mathcal{M}_{js} = \{1, \dots M_{js}\}$
$t \in \mathcal{T}$	set of time points, $\mathcal{T} = \{1, \dots T\}$
$tc_s \in \mathcal{T}$	set of constrained time points in stage s, s.t. $t \geq c_{js_p} \in \mathcal{T}$
$p_t \in \mathcal{P}$	set of electricity prices of the dynamic tariff t, $\mathcal{P} = \{p_1, \dots p_T\}$
parameters	
ec_{js}	average energy demand for job j on stage s
ec_{jsm}	average energy demand for job j on stage s in micro-step m
tp_{js}	processing time of job j in stage s
tp_{jsm}	processing time of job j in stage s in micro-step m
i_{js}	starting processing time job j in stage s
i_{jsm}	starting processing time of micro-step m of job j in stage s
c_{js}	completion processing time job j in stage s
c_{js_p}	preceding completion processing time of job j in stage s'
c_{jsm}	completion processing time of micro-step m of job j in stage s
s_0	initial processing stage s within a job j
s_p	the immediate preceding stage s' of stage s within a job j *
s_f	final processing stage s within a job j

* numerical indexes are not used due a job can skip contounous stages (e.g. a job might require to be processed on stage 1 and 3 but not in 2, etc.)

The total energy cost EC_j for job j during its entire manufacturing process can be calculated as follows:

$$EC_j = \sum_{s=1}^{S_j} \sum_{m=1}^{M_{js}} \sum_{t=i_{jsm}}^{c_{jsm}} ec_{jsm} \cdot p_t \qquad \forall j \in \mathcal{J} \quad \forall\, p_t \in \mathcal{P} \tag{3.3}$$

The total energy cost is then defined by:

$$TEC = \sum_{j=1}^{\mathcal{J}} \sum_{s=1}^{S_j} \sum_{m=1}^{M_{js}} \sum_{t=i_{jsm}}^{c_{jsm}} ec_{jsm} \cdot p_t \qquad \forall\, p_t \in \mathcal{P} \tag{3.4}$$

The maximum completion time is defined as:

$$C_{\max} = \max c_{j,s} \tag{3.5}$$

The individual job flow time (F_j) and the total flow time (F) are defined bellow. Note that s_0 and s_f corresponds to the initial and final processing stage within a job j, respectively; indexes are not used due to the number of required stages within a job j can vary and be up to 3; in case that a job only requires to be processed in a single stage, $s_0 = s_f$.

$$F_j = c_{j,s_f} - i_{j,s_0} \qquad \forall j \in \mathcal{J} \tag{3.6}$$

$$F = \sum_{j=1}^{J} c_{j,s_f} - i_{j,s_0} \tag{3.7}$$

The following assumptions and constraints are imposed:

- All machines and operations are simultaneously available at time zero.
- There are no disruptions during processing times.
- Preemption is not considered, that is, all operations are processed continuously and not interrupted.
- There is infinite buffer between any two consecutive stages.
- Each stage can process at most one operation at a time and each operation is processed on at most one stage at a time.
- The processing time for each operation is a positive integer.
- No two operations overlap, i.e., the start time of the succeeding job must be greater than the completion time of its immediately preceding job.
- The starting time of any operation is greater than or equal to its release time from the previous stage.
- All operations within a job are strictly assigned to one stage per operation.

Heuristic Scheduling

The core of the heuristic algorithm is based on sorting both, the list of job indexes (commonly known as dispatch rules in the operational research literature), and the electricity prices (also referenced as electricity curve) of the dynamic tariff. Normally, a time series is inherently sorted by the natural time index $(1, .., t)$ at which a value is observed, but in the proposed algorithm, in order to minimize the operational electricity cost, the time series is sorted in an ascending order of the electricity values (p_t) in the electricity curve, giving in this way a scheduling priority to the also sorted time indexes. In case that minimizing C_{max} is the objective function, the electricity curve is not sorted and used as is. A job operation in stage s can be allocated either at the left or right of a time index t, depending on the following conditions: at the left if all time slots covered within the period $t - tp_{js}$ are available and the condition $t - tp_{js} \geq 0$ is valid; at the right if all time slots covered within the period $t + tp_{js}$ are available and $t + tp_{js} \leq T$ is valid. If a job operation can be scheduled at the left, the allocation operation stops and the scheduling algorithm continues allocating the following job operation in the subsequent stage, otherwise it will try to allocate the current job operation at the right; if the job can not be scheduled on either side of the time index, a new different time index must be tried. This set of operations are outlined in algorithm 1. It has to be noted that time indexes t at a stage s considered for scheduling, can not have a lower value than the completion processing time index of the preceding job operation at stage s', i.e. $t \geq c_{js_p}$.

As the scheduling algorithm progresses, the probability of an operation in not finding a feasible time slot increases. This is an expected behavior; for example, a job in stage 3 with operations in stage 1 and 2 already scheduled, posses a small search space due the highly constrained positions in the electricity curve produced by stage 1 and 2. The search space is further reduced as the number of scheduled jobs increases. To face this difficulty, the list of time indexes at each stage is kept until all job operations are scheduled, so in case that an operation can not be scheduled in stage s, the previous operation in stage s' is moved to their next time index position available in their list, and the algorithm calculate the new feasible time indexes in stage s s.t. $t \geq c_{js_p}$; also, each machine M_l keeps an individual track of the operations being scheduled on them. The full scheduling algorithm is presented in Algorithm 2.

Algorithm 1 AssOX (Assignation Algorithm)

Require: $\mathcal{T}, t, S_j, c_{js_p}, M_l$
  From S_j get tp_{js}
  if $(t - tp_{js}, t) \in \mathcal{T}$ is available **and** $t - tp_{js} \geq 0$ **and** $t - tp_{js} \geq c_{js_p}$ **then**
    Assign S_j to interval $(t - tp_{js}, t) \in \mathcal{T}$ on machine M_l
  else if $(t, t + tp_{js}) \in \mathcal{T}$ is available **and** $t + tp_{js} \leq T$ **then**
    Assign S_j to interval $(t - tp_{js}, t) \in \mathcal{T}$ on machine M_l
  else
    j_s can not be scheduled in t
  end if

To foster the generation of feasible schedules, a priority change rule was also implemented; it consists of keeping track of the jobs that failed in being scheduled, and moving them to a higher priority order in the sorted job list.

As instance, suppose that we have a list of 5 jobs $J = \{1, \ldots, 5\}$ in the following order: $\{5, 4, 3, 2, 1\}$; let the jobs $\{4, 1\}$ unable to being scheduled, the priority change rule will rearrange the original list to produce the following order $\{4, 1, 5, 3, 2\}$, and proceed to try to schedule them again. In case a job was unable to be scheduled more than once, the algorithm ends and declared the schedule as infeasible.

The full heuristic scheduling procedure is summarized in algorithm 3, and further explained in the following section.

Heuristic Algorithm: Numerical Example

The following example will aim to clarify the working procedure of the heuristic algorithm. Suppose that we want to schedule a job that requires processing in stages 1 and 2 with respective processing times $tp_1 = 2$ and $tp_2 = 1$, in a toy electricity curve that spanns in a horizon of 5 time steps $\mathcal{P} = \{7.7, 6.3, 7.1, 4.4, 3.2\}, \mathcal{T} = \{1, 2, 3, 4, 5\}$, see Figure 3.2. The first step of the algorithm is to arrange the unsorted time indexes (fig. 3.2 top-left) $\mathcal{T}$ in ascending order of the electricity prices $\mathcal{P}$; the resulting series $\mathcal{T}_{new} = \{5, 4, 2, 3, 1\}$ (fig. 3.2 top-right) serves as the list of testing points to schedule the stages of the job.

Algorithm 2 HeuALG (Heuristic Algorithm)

Require: $\mathcal{T}, \mathcal{J}, \mathcal{P}$
  Sort $\mathcal{T}$ given $\mathcal{P}$
  $\ell_F = \emptyset$
  for $j \in \mathcal{J}$ **do**
    Get tc_s for $s_0 \to tc_{s0}$
    for $s \in \mathcal{S}_j$ **do**
      if $s = s_0$ **then**
        for $t \in tc_{s0}$ **do**
          AssOX($\mathcal{T}, t, s_0, c_{js_p}, M_l$)
          if s_0 can not be scheduled in t **then**
            Try a new t in tc_{s0} **Break**
          end if
        end for
      else
        Get tc_s for current $s \to tc_s$
        for $t \in tc_s$ **do**
          AssOX($\mathcal{T}, t, s_p, c_{js_p}, M_l$)
          if s can be scheduled in t **then**
            if There is a further stage **then**
              Get tc_s for the next stage $s+1 \to tc_{s+1}$
              for $t \in tc_{s+1}$ **do**
                AssOX($\mathcal{T}, t, s_{p+1}, c_{js_p}, M_l$)
                if $s+1$ can be scheduled in t **then**
                  Stop iterating in tc_{s+1} **Break**
                end if
                if $s+1$ can not be scheduled **and** $tc_{s+1} = \emptyset$ **then**
                  Try a new t in t_{cs}
                end if
              end for
            else
              There are only 2 stages in total in job j
              Stop iterating in tc_s **Break**
            end if
          end if
          if s can not be scheduled in t **and** $tc_s = \emptyset$ **then**
            Try a new t in tc_{s0}
          end if
        end for
      end if
      if j can not be fully scheduled **and** $tc_{s0} = \emptyset$ **then**
        Append $j \to \ell_F$
      end if
    end for
  end for

Algorithm 3 PipeLine

Require: $\mathcal{J}, \mathcal{P}, \mathcal{T}$ Dispatching Rule
  Sort Jobs $j \in \mathcal{J}$ by Dispatching Rule $\rightarrow \mathcal{J}'$
  HeuALG$(\mathcal{T}, \mathcal{J}', \mathcal{P}) \rightarrow \ell_F$, Job Schedules
  while $\ell_F \neq \emptyset$ **do**
    Move jobs $j \in \ell_F$ to a higher position in the sorted job list $\mathcal{J}' \rightarrow \mathcal{J}''$
    $\mathcal{J}' = \mathcal{J}''$
    HeuALG$(\mathcal{T}, \mathcal{J}', \mathcal{P}) \rightarrow \ell_F$, Job Schedules
    if A job j is more than once in ℓ_F **or** $\ell_F = \emptyset$ **then**
      Break
    end if
  end while
  if $\ell_F \neq \emptyset$ **then**
    Jobs can not be scheduled fully
  else
    All Jobs have been scheduled
  end if
  End Algorithm

For the initial operation $s_0 = 1$, there is no preceding stage so $c_{js_p} = 0$ and all time slots are available. The algorithm analyzes the feasibility of the first test point, the time index 5 (fig. 3.2 bottom-left). The conditions to schedule the operation at the left are meet: period $(t - tp_1, t) \rightarrow (5 - 2, 5) \rightarrow (3, 5)$ is available and the constraint $(t - tp_1 > 0 \rightarrow 5 - 2 > 0 \rightarrow 3 > 0$ is meet; afterwards the time slots (3,4) in stage 1 are blocked for further consideration.

For the following operation $s_p = 2$, $c_{js_p} = 5$, the algorithm calculates the constrained time indexes $tc_s = t \geq c_{js_p} \in \mathcal{T} \rightarrow t \geq 5 \in \{1, 2, 3, 4, 5\} = \{5\}$ and proceed to check the feasibility on both sides. To the left, the inequality $t - tp_2 \geq c_{js_p} \rightarrow 5 \ - 1 \geq 5 \rightarrow 4 \geq 5$ is violated, to the right, the inequality $t + tp_2 \leq T \rightarrow 5 \ + 1 \leq 5 \rightarrow 6 \leq 5$ is also violated. As a consequence the algorithm concludes that the initial test point for the operation s_0 is not feasible to use for scheduling, and proceeds to try the second test point in s_0 which has a time index of 4. As done previously, the algorithm test the feasibility to schedule s_0 to the left of index 4 for which all constraints are meet, and therefore is scheduled in the interval (2,4). For the operation in stage $s_p = 1$, we now have $c_{js_p} = 4$ and $tc_s = \{5, 4\}$.

The algorithm takes the test point in index 5 and as all the constraints are meet, it determines that now is feasible to schedule the operation to the left (fig. 3.2 bottom-right). With no more jobs to schedule the algorithm ends. It should be pointed out that, in case of trying to minimize C_{max}, the only difference in the procedure is that there is no sorting of the electricity price, and the initial time index to be scheduled correspond to the number 1.

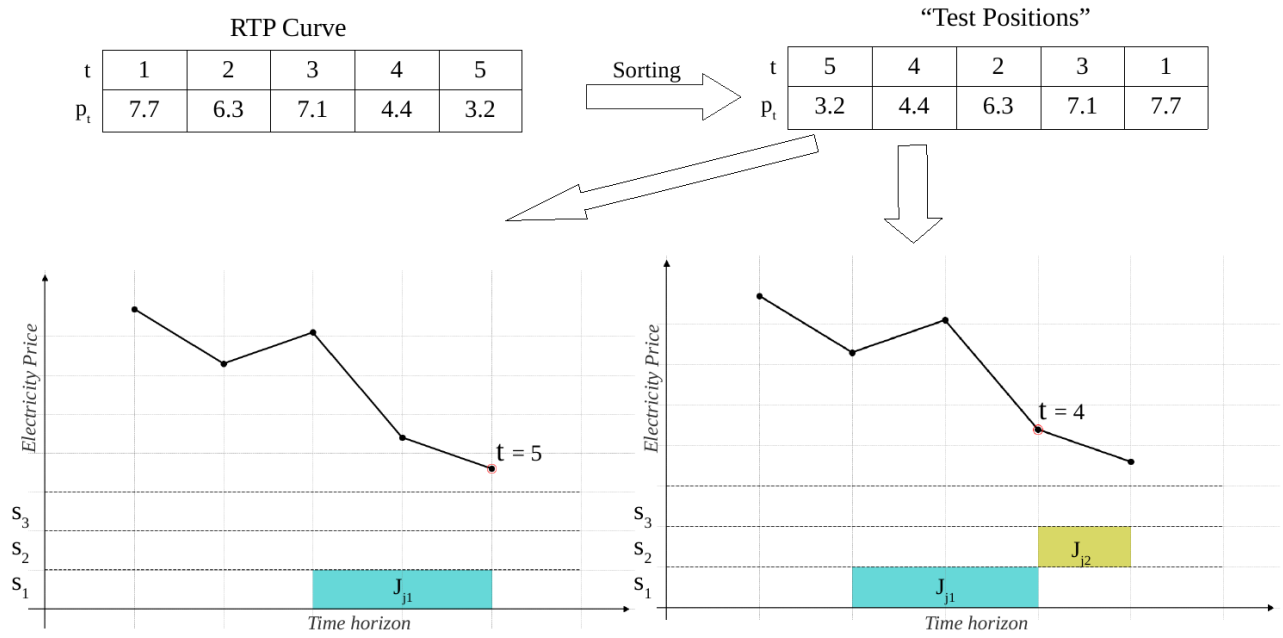

Figure 3.2: Scheduling procedure.

Dispatching Rules

Following the literature, the following dispatching rules were tested over the list of jobs $\mathcal{J}$:

- **EcHigh**: Jobs $j \in \mathcal{J}$ sorted by its overall energy consumption EC_j (Eq. 3.3), decreasing order.
- **EcLow**: Jobs $j \in \mathcal{J}$ sorted by its overall energy consumption EC_j, increasing order.

- **Length**: Jobs $j \in \mathcal{J}$ sorted by its overall processing time $\sum_1^s \sum_1^m tp_{jsm}$, decreasing order.
- **LengthS**: Jobs $j \in \mathcal{J}$ sorted by its overall processing time $\sum_1^s \sum_1^m tp_{jsm}$, increasing order.
- **ByID**: Jobs $j \in \mathcal{J}$ sorted by its code number (1,...J), increasing order. This would be equivalent to processing the jobs by their arrival order to a manufacturing environment.
- **Random**: $j \in \mathcal{J}$ Jobs sorted at random until a feasible one is found. The *while* loop in Algorithm 3 is omitted, and the randomization of the jobs list and scheduling is performed until a feasible schedule is found or a maximum number of iterations (100) is reached.
- **Length + Multi Start**: Jobs $j \in \mathcal{J}$ sorted by Length and then scheduled on 10 different modified price vectors. The multi-start procedure consist in modifying each individual value p_t of the original electricity curve's price vector $\mathcal{P}$, by using the formula $p_{t_{new}} = p_{t_{old}} * (1 \pm x)$, where x is chosen at random from a range of $[0, 0.3]$. The modified price vector $\mathcal{P}'$ is used to perform the scheduling, and after completion, the modified price vector is changed to the original one. This procedure is repeated a total of 10 times, and then the solution with the lowest overall energy cost is chosen.

Genetic Optimization

Usually, a Genetic Algorithm can use a chromosome (a particular solution) encoded by numeric, symbolic, or alphanumeric ways. In this FJSP, the array of ordered jobs can be viewed as an array of unique integers and thus, can be directly related to Genetic Algorithms used for solving the Travel Salesman Problem (TSP). Under this framework, the Heuristic algorithm acts as the gen decoder and allows to calculate the evaluation parameters (fitness function) to perform the Genetic Optimization of the makespan C_{max} (Eq. 3.5) and the total electricity cost TEC (Eq. 3.4).

There are a number of different crossover operators for integer arrays in TSP [34], and the one used for this optimization is the Position-Based Crossover.

This operator rely in two probabilities, a mating probability and a genetic exchange probability. The mating probability establishes how often a set of parents will produce an offspring. The genetic exchange probability specify the number of positions (at random) from the first parent to be inherited to the offspring. For example, if a genetic exchange is set to 50 % for a parent gen of size 9, it means that the values of 5 positions chosen at random will be transferred to a proto-offspring. To complete it, the integers already selected from the first parent are excluded from the second parent, and the remaining integers will fill the missing positions of the proto-offspring in the order that they appear in the second parent (see Figure 3.3).

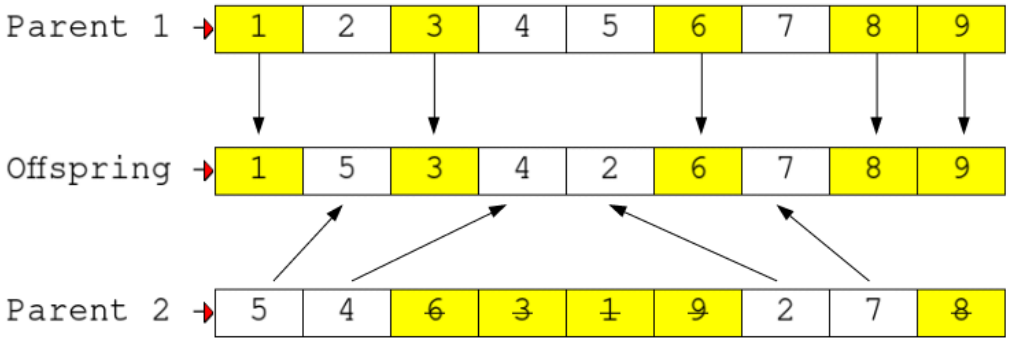

Figure 3.3: Position-Based Crossover

The mutation operator chosen is the Interchanging Mutator. Consists in selecting two random positions of the offspring based in a occurrence probability, and interchanging the values of those positions (see Figure 3.4). For instance, for a mutation probability of 20 %, approximately 1 in 5 offsprings will go through mutation.

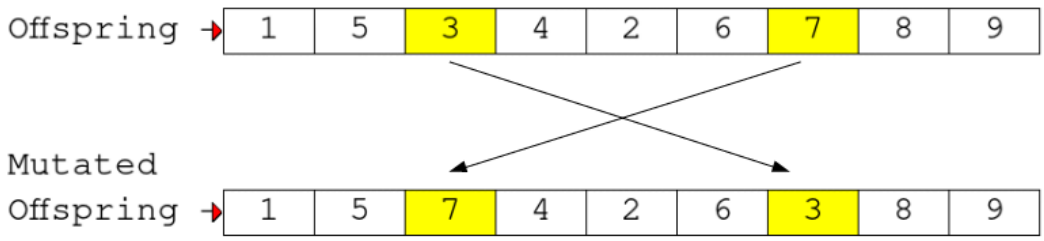

Figure 3.4: Mutation Operation

After defining a population size (i.e. the number of *individuals* to be generated and evaluated within each generation), an initial population is produced by randomizing the the list of jobs $\mathcal{J}$. After the first generation has been evaluated with the fitness functions, a percentage of individuals with the best fitness values are selected to mate (this is also known as elitism) until a new population of the same size of its predecessor is reached; the original parents are always included in the next generation.

Two stopping criteria were used in the genetic optimization process: either evaluate a maximum of 30 Generations or an early stop if the standard deviation of the best values of the fitness function for the last 5 generations is less than $1x10^{-2}$ (meaning that the algorithm has converged to a local optima). The pseudoalgorithm for the Genetic Optimization is presented in Algorithm 4. The Heuristic Algorithm is used to evaluate each individual within each generation, and in case that an gen yielded an infeasible schedule, the values of the fitness function (TEC and C_{max}) for that specific gen were set to infinity (since the best values for the objective functions of makespan C_{max} and energy cost TEC tends to zero, such gens would be ignored when selecting the elite gens). The hyperparameters values used in the genetic algorithm are presented in Table 4.2.

Algorithm 4 GenALG (Genetic Algorithm)

Require: max_num_of_generations, population_size, num_elite_gens, fitness_function
  for Generation ∈ (max_num_of_generations) **do**
    if Generation = First Generation **then**
      Initialize a random population of size → population_size
    else
      if Generation > 15 **then**
        if Std_Dev of past 5 generations < $1x10^{-2}$ **then**
          Convergence Reached **Break**
        end if
      end if
      From evaluated_generation select Elite_Gens → num_elite_gens
      Mate parents from the Elite_Gens using the Crossover and Mutation Operators, until reach the population_size
    end if
    Evaluate Generation with fitness_function → evaluated_generation
  end for
  Return Best Gen (best solution), and best fitness value
  End Algorithm

Chapter 4

Experiments

Experiment Design

To assess the efficiency of the Heuristic algorithm for the FJSP, a total set of 750 different scenarios were created. Three main parameters differentiate all the scenarios: Number of jobs, Difficulty, and range of energy consumption per stage job. The number of jobs varied from a value of 10 to 70, in increments of ten.

The difficulty is related to the processing time of each stage s within each job j. A difficulty type *"a"* was assigned to short duration job stages, a difficulty type *"b"* to long-duration job stages, and a difficulty type *"c"* to job stages with a duration mixed between type *"a"* and *"b"*.

Similarly, three categories of ranges for energy consumption were established; a type *"1"* for those stages with low energy consumption, a type *"2"* for those stages with high energy consumption, and a type *"3"* for those with a mixture of levels *"1"* and *"2"*.

Also, to measure reproducibility each permutation of { Number of Jobs, Difficulty, Energy Consumption} has at least 10 different scenarios. The general design of all the scenarios is summarized in table 4.4, which also includes the time horizon used for the job scheduling; figure 4.1 shows an example of electricity curve with price fluctuations over a period of 168 h. Table 4.1 shows the specific ranges used for each parameter. Table 4.3 shows an example of a two-job scenario, with the detailed information of each stage s within each job, as well as the specific values of the processing time tp_{jsm} and energy consumption ec_{jsm} for each micro-step m_{js}. If a stage has a total processing time $tp_{js} = 0$, means that the job does not require to visit a the machine at such station; in this example, j_1 and j_2 only visit a single machine each (machine 2 and machine 3, respectively).

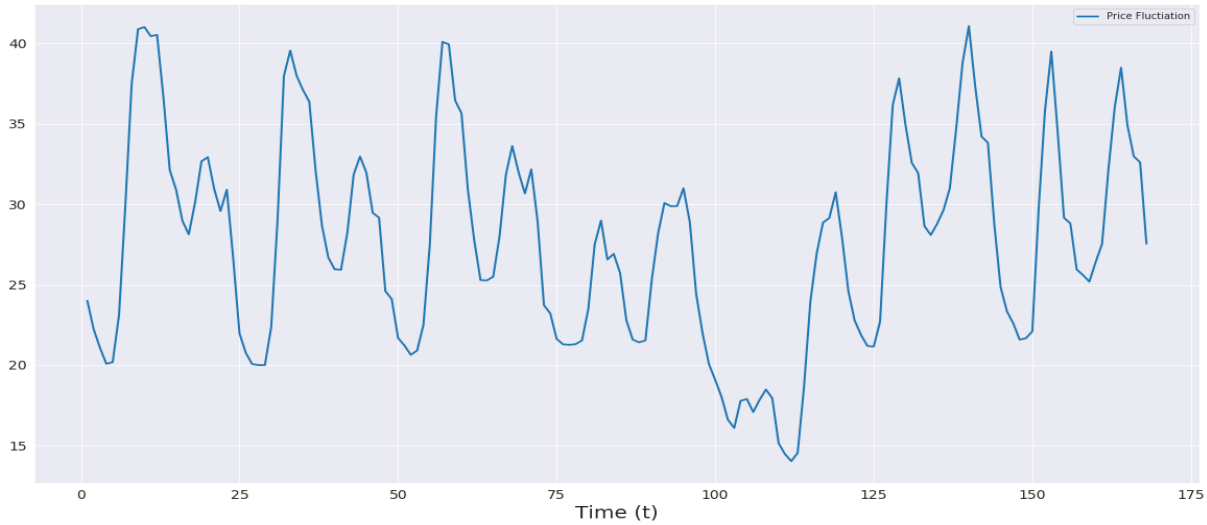

Figure 4.1: Example of an Electricity curve spanning over a period of time of 168 h.

For each of the dispatching rules mentioned in section 3 and for the genetic optimization, the heuristic algorithm tried to find a optimal solution for each of the 570 scenarios. A summary of the results is presented in table 4.5 were three main metrics are considered: the average computing time in seconds; the success rate (in percentage), defined as the number of times a feasible solution was found by the heuristic algorithm for a given scenario; and the average cost gap (in percentage), defined as the gap in terms of the overall electricity cost between the solution found by the MIP solver against the one found (if any) by the heuristic algorithm. More detailed information can be found at tables 6.1, 6.2, and 6.3 in the appendix.

Table 4.1: Parameter ranges used to create each scenario of the FJSP

Energy Consumption	Range
Case 1	0.1 to 3
Case 2	3 to 6
Case 3	0.1 to 6
Processing time at stage s	Range
Case a	1 to 10
Case b	11 to 20
Case c	1 to 20

Table 4.2: Hyperparameters used for the Genetic Optimization

Mating Probability	100%
Genetic Exchange Probability from parent 1	60%
Convergence Tolerance	$1x10^{-2}$
Mutation Probability	10%
Max Number of generations	30
Population Size	50
Number of elite genes per generation	10

Table 4.3: Example of a Two-Job Scenario

Job ID	**Stage** s	**Micro-step** m_{js}	**Processing time** tp_{jsm}	**Energy consumption** ec_{jsm}
J1	1	m1	0	0
		m2	0	0
		m3	0	0
	2	m1	1	0.4116
		m2	4	0.2401
		m3	1	0.0294
	3	m1	1	2.7027
		m2	0	0
		m3	0	0
J2	1	m1	0	0
		m2	0	0
		m3	0	0
	2	m1	0	0
		m2	0	0
		m3	0	0
	3	m1	1	1.7302
		m2	9	0.9495
		m3	2	0.1688

Table 4.4: Experimental Design.

No. of Jobs	Difficulty	Energy Consumption Ranges	Time Horizon (h)	Total No. of Scenarios
10	a	1,2,3	168 h	30
	b	1,2,3	168 h	30
	c	1,2,3	168 h	30
20	a	1,2,3	168 h	30
	a	1,2,3	336 h	30
	b	1,2,3	336 h	30
	c	1,2,3	336 h	30
30	a	1,2,3	168 h	30
	a	1,2,3	336 h	30
	c	1,2,3	336 h	30
40	a	1,2,3	336 h	30
	c	1,2,3	336 h	30
50	a	1,2,3	672 h	30
	b	1,2,3	672 h	30
	c	1,2,3	672 h	30
60	a	1,2,3	672 h	30
	c	1,2,3	672 h	30
70	a	1,2,3	672 h	30
	c	1,2,3	672 h	30
			Total	570

For the Genetic Optimization, the hyperparameters were optimized by random search over 10 % of all the data set (i.e. choosing one scenario at random from each of the permutation {Number of Jobs, Difficulty, Energy Consumption}). The values of the Hyperparameters are shown in table 4.2. The experiments were conducted in a laptop with Intel(R) Core(TM) i3-2328M CPU @2.20GHz and RAM Size 4096 MB, Type DDR3 with speed of 1333 MT/s.

Results

Minimizing Total Cost Electricity

This section aggregate the results obtained by the Heuristic algorithm over the 570 instances for the TEC minimization, for which the Heuristic algorithm sorts the values of the electricity price curve in an increasing manner. They are presented with a focus on the three ranges of difficulty (processing times) used, as this factor was determined to be the most important one to be considered to find a feasible solution. The range of energy consumption and the number of jobs have a direct impact on the final value of the TEC objective, but is ultimately the length of the stages within each job that dominates if the heuristic algorithm is able or not to find a solution.

At an early research stage, only 5 sorting methods were considered: *"ByID, "ECHigh","EcLow","Length","Random"*, which are described in detail in section 3. Analyzing the results of only those sorting methods, the success rate is nearly 100 % in all the scenarios with short processing stages (difficulty *"a*), and decreases up to 30% as the length of the processing stages increases. This can be directly seen in table 4.5, as scenarios with difficulty *"b"* (i.e. the ones with the largest processing times) have the worst results, and the scenarios which difficulty *"c"* (a mixture between leghts found in difficulty *"a"* and *"b"*) have an intermediate success rate. As an effort to improve the success rate in the worst cases (difficulty "b) two additional sorting methods were tested "LenghtS", and "LengthL+MultiS" described in detail as well in section 3.

The results obtained with the sorting method *"LengthS"* shows that providing a greater priority to jobs with shorter processing times have a remarkably negative impact on the success rate of the heuristic algorithm, producing the lowest success rate among all sorting methods tested. The behaviour of this sorting method is attributable to the fact that as the scheduling progresses, jobs at the end of the list (the ones with the longest processing times) have to find a feasible position within a highly constrained time schedule at each machine, resulting in most cases in the impossibility to allocate all of them. Following this line of thought, unsurprisingly the best sorting method obtained was *"Length"* within all the levels of difficulty (it even achieves a 100% of success for the *"a"* scenarios.

The multi-start procedure can be evaluated as a direct improvement to the sorting method *"Length"*, and for the scenarios within difficulty *"b"*, it increases the success rate from 54% to 73%, with the cost gap maintained almost identical (11.16% versus 11.23%).

Table 4.5: Heuristic algorithm Results. Computing Time, Success Rate and Energy Cost Gap compared against different dispatching rules and the Genetic Optimization. Objective function: Total Electricity Cost TEC minimization.

Difficulty	**Sorting Method**	**Avg. Computing Time (s)**	**Success Rate (in %)**	**Avg Cost Gap (in %)**
a	ByID	0.385	97.037	13.326
	EcHigh	0.799	98.889	10.187
	EcLow	0.441	97.037	16.229
	Length	0.425	100.000	12.889
	Random	17.2	96.296	13.274
	Ga_Opt	49.945	98.519	6.750
b	ByID	2.006	50.000	9.524
	EcHigh	1.438	58.889	9.080
	EcLow	1.900	33.333	11.046
	Length	0.713	54.444	11.169
	Random	33.10	45.556	10.089
	Ga_Opt	49.994	41.111	6.570
	LengthS	0.832	31.110	9.780
	LengthL-MultiS	12.32	73.330	11.230
c	ByID	2.503	73.333	12.613
	EcHigh	1.320	85.238	9.245
	EcLow	2.418	61.905	14.867
	Length	2.152	92.381	12.170
	Random	27.885	76.667	12.443
	Ga_Opt	78.698	78.571	6.727

This enhancement can be explained by the fact that a multi-start procedure what it does is to directly increase the searching space of the algorithm; so with a huge new number of positions to test it is more likely to find a feasible (but not necessarily optimal) solution. Putting it in other words, this procedure is an intermediate step between scheduling the same scenario using a totally different electricity curve.

Concerning the minimization of the TEC objective, and considering solely the sorting methods, the one that generated the best cost gap (i.e. the lowest gap between the MIP solution) in all the difficulties is *"EcHigh"*. This is an expected result, as it is logical that by firstly allocating the jobs with high electricity demand on positions with lower electricity price will produce a low-cost schedule.

Consequently, the sorting method that performed the worst was *"EcLow"*, which leaves the jobs with high electricity demand to schedule at the very last, where only the most expensive electricity prices are left.

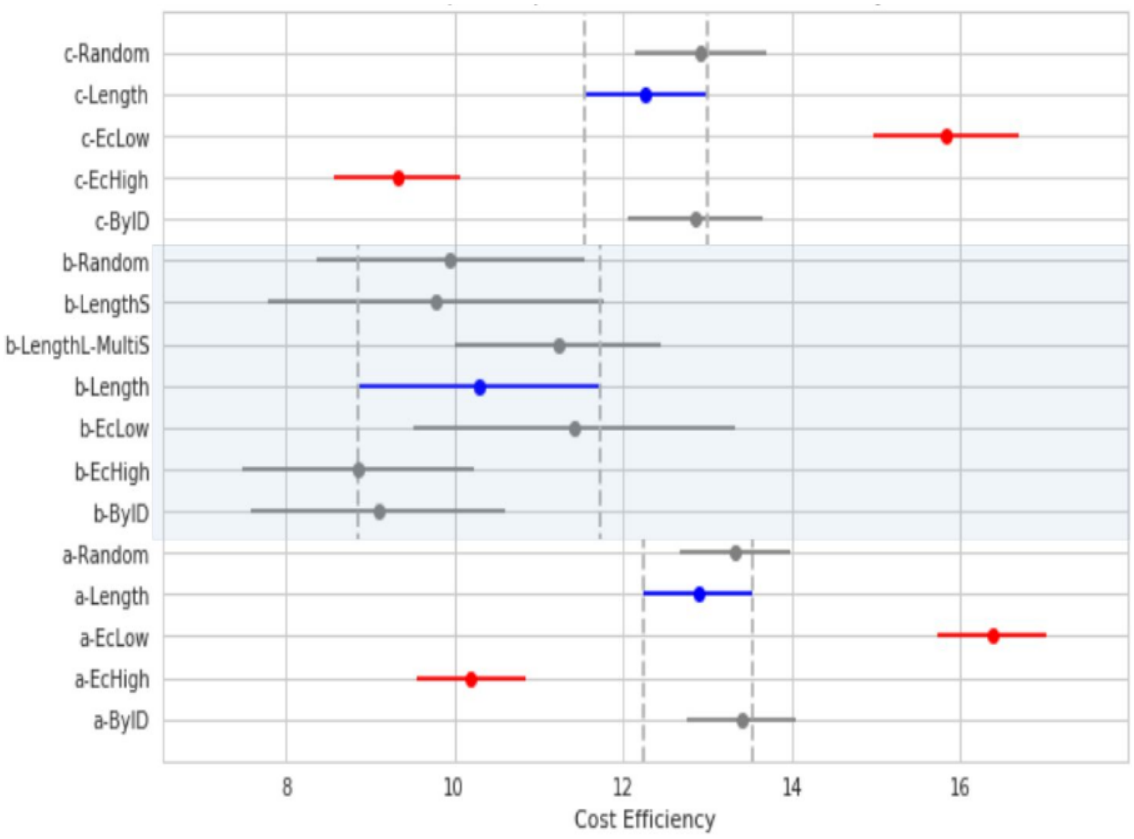

Figure 4.2: Tukey results, mean Cost Gap vs Sorting Method-Difficulty. Significance can be determined by looking for overlap: groups in Red are statistically different from those marked in Blue/Gray. The analysis was subdivided by Difficulty "a","b" (shaded), and "c".

An ANOVA test (alpha =0.05) for the factors "Difficulty", "Sorting Method" and the Interaction "Difficulty – Sorting Method" yields a p-value less than 0.05 in all cases, which tell us that at least there is one mean between these groups that is statistically different from the others. A Tukey was used to identify such differences (see Figure 4.2).

In figure 4.2, by looking at the overlapping intervals for each individual sorting method, it can be inferred that for difficulty *"a"* and *"c"*, the mean results from the sorting method *"EcHigh"* and *"EcLow"* are statistically different from the others, and that they also represents the best and the worst Cost Gap vs the LP-Solver, respectively.

Although in the *"b"* scenario, the average Cost Gap is statistically the same between all the sorting methods, the sorting *"EcHigh"* yields a mean that is directionally lower than the other sorting methods.

With respect to the genetic optimization (*"Ga˙Opt"*), it is the procedure that requires more time than the other simpler dispatching rules, followed by the *"Random"* approach. The *"Ga˙Opt"* method also posses a relatively low success rate for scenarios with *"b* difficulty, and average on the other two. Is in relation to the cost gap where it really shines, as it generates the closest schedules to the MIP solutions (the smaller cost gap).

This algorithm requires a diverse number of hyperparameters to tune for it perform a good minimization of the objective function. Figure 4.3 shows 10 different convergence curves for 10 different scenarios (all composed of 10 jobs); it can be seen that for the majority of the scenarios, the genetic optimization reaches the stopping criteria around 25-30 generations.

Table 4.6 exemplify the procedure followed to choose a suitable Genetic Exchange Probability for a single scenario. In this example, six different exchange probabilities were tested, and for each one of them, the genetic optimization was performed to completion with five repetitions. Statistics of mean, standard deviation, relative standard deviation (%), and distance to the minimum (%) are included. The later is calculated as difference in percentage to the minimum mean value of all the tested probabilities. It can be observed that gen exchange probabilities close to 0.5 produce the lower mean values. Conducting the same procedure over 10% of the total scenarios and averaging, an optimal value of gen exchange of 0.4 (rounded) was obtained. The rest of the hyperparameters were obtained by random search. Figure 4.4 shows a visualization of the solution (schedule) of an scenario composed of 70 jobs, allocated on a time horizon of 672 h.

Table 4.6: Selecting a good gen exchange probability

Gen Exch Prob	0	0.1	0.25	0.5	0.75	1
Rep 1	5,907.39	5,786.41	5,719.99	5,707.34	5,780.84	5,819.41
Rep 2	5,856.16	5,712.25	5,718.16	5,655.79	5,738.86	5,907.39
Rep 3	5,901.87	5,795.30	5,768.44	5,646.99	5,783.88	5,868.47
Rep 4	5,911.07	5,821.76	5,737.35	5,714.10	5,720.73	5,816.81
Rep 5	5,879.67	5,699.81	5,736.41	5,700.11	5,737.96	5,873.77
Mean	5,891.23	5,763.11	5,736.07	5,684.87	5,752.45	5,857.17
Std	20.65	48.19	18.05	27.83	25.28	34.58
Relative Std (%)	0.35	0.84	0.31	0.49	0.44	0.59
Distance to Minimum (%)	3.6	1.4	0.9	0.0	1.2	3.0

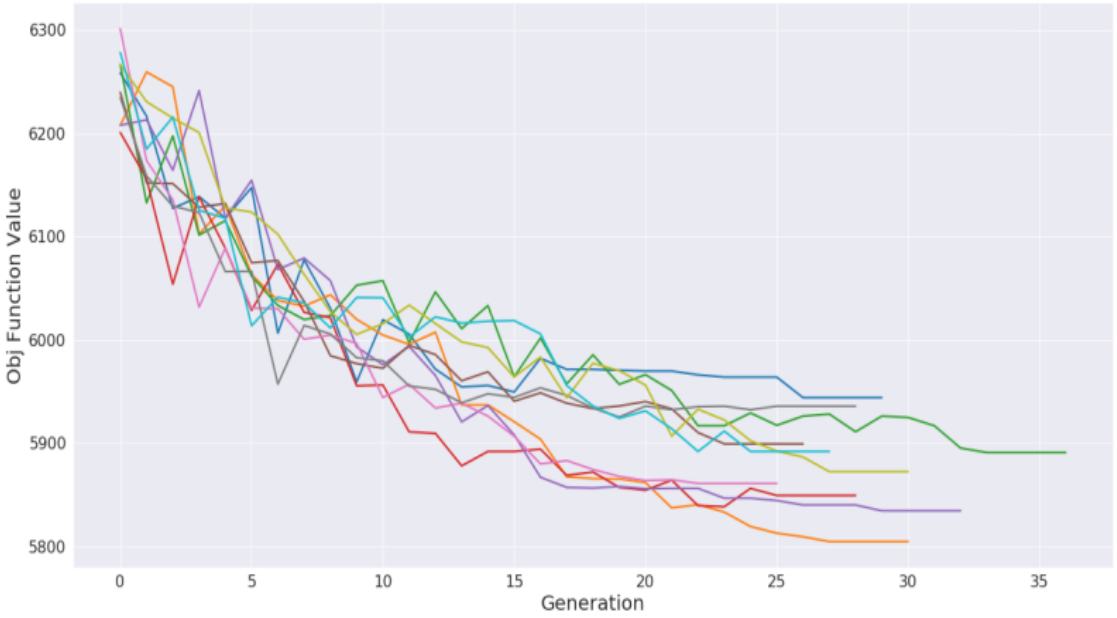

Figure 4.3: Example of the convergence achieved during genetic optimization. Each line represents the evolution of the cost function through each generation for 10 different scenario.

Minimizing Total Cost Electricity C_{max}

This section aggregate the results obtained by the Heuristic algorithm over the 570 instances for the makespan (C_{max}) minimization. For this experiments, the Heuristic algorithm do not sorts the values of the electricity price curve, and it use them as they appear. As in the minimzation of the TEC, table 4.7 presents the results focused on the three ranges of difficulty (processing times) used.

Table 4.7: Heuristic Algorithm. Computing Time, Success Rate and Average Makespan Gap compared against different dispatching rules and the Genetic Optimization. Objective function: Makespan minimization

Difficulty	**Sort Method**	**Avg. Computing Time (s)**	**Success Rate**	**Avg Makesp Gap**
a	ByID	0.385	100.000	6.875
	EcHigh	0.799	100.000	3.632
	EcLow	0.441	100.000	8.486
	Length	0.425	100.000	2.709
	Random	12.32	100.000	6.597
	Ga_Opt	49.945	100.000	-0.470
b	ByID	2.006	100.000	5.125
	EcHigh	1.438	100.000	3.460
	EcLow	1.900	100.000	4.592
	Length	0.713	100.000	2.150
	Random	38.90	100.000	4.680
	Ga_Opt	49.994	100.000	-2.432
c	ByID	2.503	96.111	5.188
	EcHigh	1.320	96.111	2.144
	EcLow	2.418	96.111	7.622
	Length	2.152	96.111	1.440
	Random	23.45	95.556	5.430
	Ga_Opt	78.698	93.889	-1.946

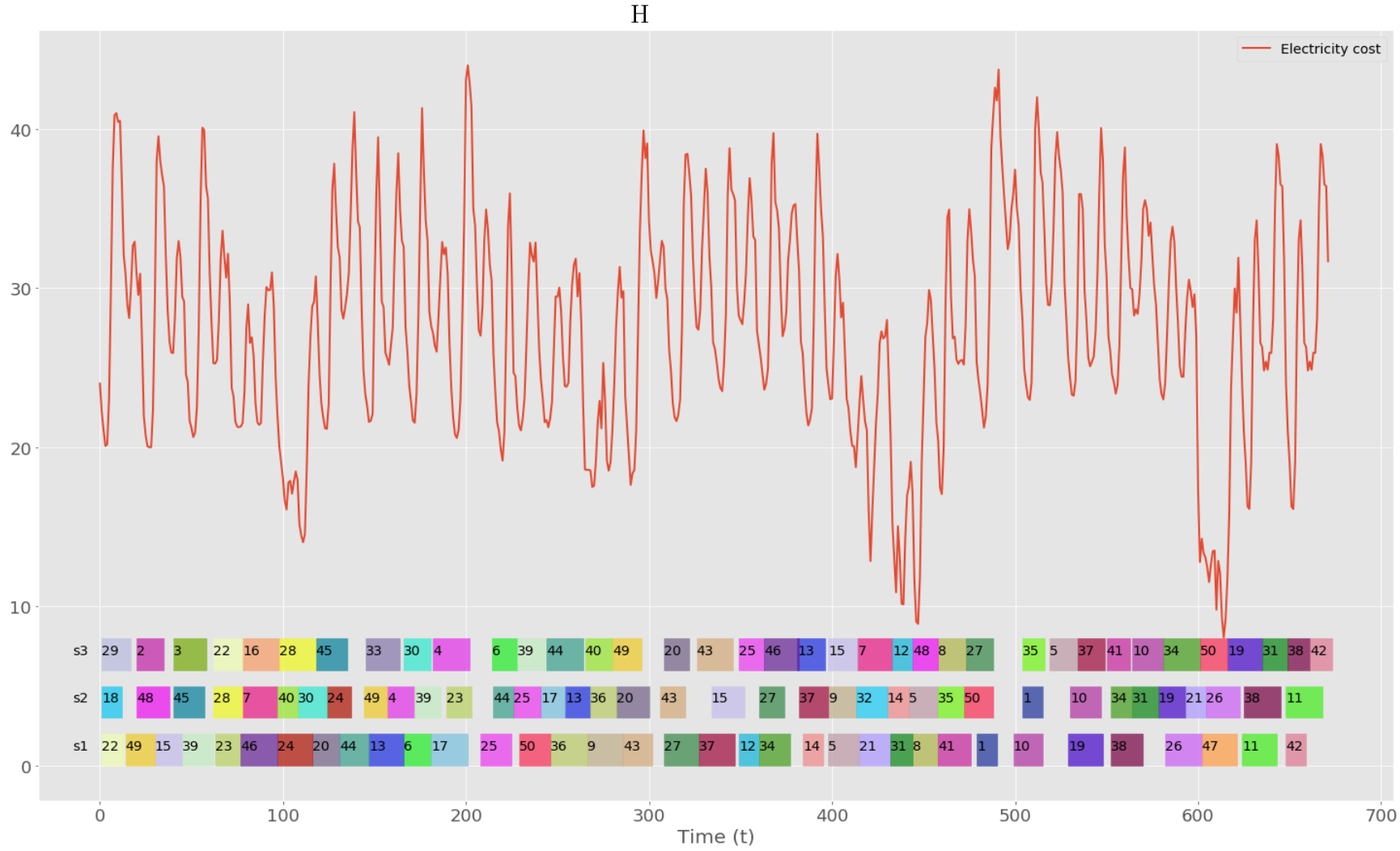

Figure 4.4: Example of an Solution found by the Heuristic Algorithm. Time horizon 672 h , 50 jobs, Difficulty *"b"*, and energy consumption *"2"*

A difference to the TEC minimization, the success rate of the heuristic algorithm is (or nearly is) 100% in the majority of the scenarios and the gap in the C_{max} is way less than it is for the TEC objective. The sorting method that produced the lower gap values was *"Length"*, and the one that produced the higher gap values was *"EcLow"*. It should be noted that due an excellent success rate value, no further sorting methods were tested.

The genetic optimization is again the one that requires the highest computing time among all the tested methods and has the lowest success rate in the difficulty *"c"* scenarios, but is the one that actually could find better schedules (lower C_{max} than the MIP solver.

The reason this happens, is that for scenarios with **"b"**, **"c"**, the MIP solver did not complete the scheduling to optimallity (computing time threshold of 10,000 seconds), whereas the heuristic algorithm in conjunction with the genetic optimization obtained an closer optimal value than the MIP solver.

In this context, one of the most valuable asset of the heuristic scheduler, even in conjunction with the genetic optimizer, is that the computing time is way less than the required from the MIP solver (GAMS). For instance, the MIP solver requires times around 5,000-10,000 seconds to obtain an optimal value for scenarios **"b"** or **"c"**, compared with the 40-100 required for the genetic optimizer.

Chapter 5

Conclusion

A heuristic scheduler was developed and implemented for the FJSP problem under an RTP tariff. The core of the model relies on two elements: a specific scheduling heuristic that builds a scheduled upon sorted time indexes of the dynamic tariff, and dispatching rules applied to the jobs. Furthermore, by encoding the jobs as permutation sequences, the model can be optimized through a genetic programming approach to find the adequate dispatch sequence that optimizes the objective function.

A total of 570 different scenarios with a varying number of jobs, electricity consumption ranges, and processing times, allowed to evaluate the performance of the heuristic versus the MIP solver introduced in [9].

One of the major drawbacks of this heuristic is that, depending on the difficulty of the scenario it was not always possible to find a feasible solution, especially when optimizing the total energy cost. For instance, complexes scenarios had a success rate even below 50%. Commonly known dispatch rules along the heuristic algorithm, achieved electricity costs around 10-13% higher than the ones found by the MIP solver, whereas the Genetic Optimization improved them up to only 6.5% higher.

In the case of the makespan, the dispatching rules alone produced values around 6-10% higher than the ones from the MIP solver, and in contrast, the genetic optimization achieved even better results than the MIP solver (around 2%).

The major asset of the Heuristic algorithm is that it can find a feasible solution in less than 1 minute, compared to the 2-3 hours (over complex scenarios) required for the MIP solver. One of the reasons of this behaviour is that the Heuristic algorithm, in order to assign a job, do not test all the available positions in the time horizon like the MIP does.

Nevertheless, this also could be the the reason that the Genetic Optimization can not close the Electricity efficiency gap any further. In this research, there was always only a single objective function being optimized, but the usage of the genetic algorithm could also allow to optimize multiple objective functions, as well as the creation of Paretto frontiers, by using a NSGA technique. [13].

A further improvement to the genetic optimization could be to avoid using a complete random population initialization, and using instead the sorted job indexes by the best dispatching rules of *"Length"* and *"EcHigh"*; this can serve as a warm start and accelerate the convergence of the algorithm. A multi-ual per generation.

The GA could also be combined with some other heuristic algorithms, often with a local search procedure applied to the offspring generated by crossover and/or mutation. Such a procedure is often denoted as genetic local search (i.e., a population based local search) or memetic search [34].

Chapter 6

Appendix

Expanded Tables

Table 6.1: Average electricity cost gap of the solutions found by the Heuristic algorithm with respect to the MIP solutions, by scenarios.

Cases	ByID	EcHigh	EcLow	Length	Random	Ga_Opt
14d-20j-1a	14.27	9.93	15.01	11.19	12.11	6.4
14d-20j-1b	7.81	9.63	14.08	10.17	12.5	6.04
14d-20j-1c	12.92	8.46	16.04	12.65	12.69	5.6
14d-20j-2a	8.47	7.23	10.1	7.51	8.44	4.92
14d-20j-2b	8.59	7.29	9.92	8.38	6.16	5.46
14d-20j-2c	9.94	7.45	11.21	8.67	9.78	5.27
14d-20j-3a	14.28	9.91	15.79	12.31	13.63	6.06
14d-20j-3b	10.41	9.19	13.79	12.2	12.85	5.65
14d-20j-3c	13.62	9.82	16.06	11.55	12.6	6.32
14d-30j-1a	12.92	8.02	14.98	11	12.06	5.21
14d-30j-1c	12.02	9.56	15.53	12.87	11.28	6.17
14d-30j-2a	9.11	7.68	11.23	8.15	9.85	5.19
14d-30j-2c	9.23	7.04	10	8.2	9.01	4.87
14d-30j-3a	12.15	8.09	15.44	10.75	11.69	5.38
14d-30j-3c	9.3	9.03	13.83	12.86	10.98	6.42
14d-40j-1a	12.51	8.66	16.98	12.53	12.79	6
14d-40j-1c	5.91	8.2	5.84	9.41	9.9	5.18
14d-40j-2a	9.87	7.27	11.3	8.75	10.16	4.9
14d-40j-2c	6	6.83	NaN	5.39	7.42	5.51
14d-40j-3a	14.33	8.79	17.29	12.69	13.39	5.84
14d-40j-3c	9.31	6.46	4.96	11.03	8.03	7.41

Continuation of Table 6.1

Cases	ByID	EcHigh	EcLow	Length	Random	Ga_Opt
28d-50j-1a	18.69	13.47	22.98	19.23	18.22	8.31
28d-50j-1b	11.45	12.12	1.29	15.79	10.93	13.4
28d-50j-1c	18.95	11.7	22.52	16.4	18.51	7.49
28d-50j-2a	12.06	10.95	14.51	12.59	12.52	7.43
28d-50j-2b	6.94	7.73	NaN	11	5.19	5.72
28d-50j-2c	12.67	9.28	14.64	10.65	13.03	6.29
28d-50j-3a	18.96	15.32	24.49	19.95	19.1	9.26
28d-50j-3b	15.4	11.53	18.18	14.9	14.78	8.94
28d-50j-3c	18.37	11.84	22.86	17.56	17.67	7.02
28d-60j-1a	18.23	14	22.77	17.82	18.13	10.35
28d-60j-1c	18.29	10.63	21.15	16.32	17.57	8.14
28d-60j-2a	12.13	10.25	14.26	11.95	12.1	7.18
28d-60j-2c	11.52	8.54	12.26	10.78	11.89	6.4
28d-60j-3a	17.94	12.87	23.27	18.49	18.72	9.29
28d-60j-3c	16.15	10.92	20.34	16.54	16.78	8.44
28d-70j-1a	18.06	12.37	24.35	17.1	17.97	9.61
28d-70j-1c	17.71	12.01	22.58	17.05	13.64	9.64
28d-70j-2a	12.27	10.09	14.09	11.62	12.07	7.87
28d-70j-2c	10.47	8.35	1.49	11.04	10.28	8.12
28d-70j-3a	17.95	12.52	24.48	18.09	18.93	9.38
28d-70j-3c	16.42	10.28	21.06	15.41	16.27	10.55
7d-10j-1a	12.71	8.65	14.53	11.5	10.08	5.41
7d-10j-1b	8.94	8.92	10.69	10.56	10.56	5.16
7d-10j-1c	13.2	11.55	17.18	11.27	11.79	7.28
7d-10j-2a	9.16	8.67	12.41	8.38	11.3	5.89
7d-10j-2b	7.77	6.32	8.65	7.9	7.14	3.98
7d-10j-2c	10.03	7.22	12	9.3	9.26	4.57
7d-10j-3a	13.2	12.01	15.72	13.22	12.92	6.9
7d-10j-3b	8.41	8.99	11.77	9.62	10.69	4.78
7d-10j-3c	12.85	8.98	15.78	10.63	12.92	4.58
7d-20j-1a	14.13	11.11	15.93	11.72	13.61	6.5
7d-20j-2a	10.29	8.88	11.94	9.29	10.57	5.4
7d-20j-3a	12.21	9.9	16.58	12.87	13.07	5.92
7d-30j-1a	13.67	10.52	17	14.01	12.08	6.44
7d-30j-2a	7.37	7.24	8.1	10.53	8.3	4.18
7d-30j-3a	12.85	10.65	12.64	14.75	14.6	7.04

Table 6.2: Success Rate of the Heuristic Algorithm, by scenarios.

Cases	ByID	EcHigh	EcLow	Length	Random	Ga_Opt
14d-20j-1a	100	100	100	100	100	100
14d-20j-1b	60	80	40	80	50	30
14d-20j-1c	100	100	100	100	100	100
14d-20j-2a	100	100	100	100	100	100
14d-20j-2b	80	90	70	80	70	80
14d-20j-2c	100	100	100	100	100	100
14d-20j-3a	100	100	100	100	100	100
14d-20j-3b	80	100	70	100	80	80
14d-20j-3c	100	100	100	100	100	100
14d-30j-1a	100	100	100	100	100	100
14d-30j-1c	60	70	60	90	80	60
14d-30j-2a	100	100	100	100	100	100
14d-30j-2c	90	100	100	100	100	100
14d-30j-3a	100	100	100	100	100	100
14d-30j-3c	100	100	70	100	80	90
14d-40j-1a	100	100	100	100	100	100
14d-40j-1c	60	60	10	80	30	20
14d-40j-2a	100	100	100	100	100	100
14d-40j-2c	20	60	0	70	10	20
14d-40j-3a	100	100	100	100	100	100
14d-40j-3c	10	50	10	60	30	10
28d-50j-1a	100	100	100	100	100	100
28d-50j-1b	30	40	10	10	20	10
28d-50j-1c	100	100	100	100	100	100
28d-50j-2a	100	100	100	100	100	100
28d-50j-2b	40	40	0	10	20	10
28d-50j-2c	100	100	80	100	100	100
28d-50j-3a	100	100	100	100	100	100
28d-50j-3b	20	30	10	20	10	10
28d-50j-3c	70	100	90	100	100	100
28d-60j-1a	100	100	100	100	100	100
28d-60j-1c	70	90	50	100	90	100
28d-60j-2a	100	100	100	100	100	100
28d-60j-2c	90	100	40	100	100	100
28d-60j-3a	100	100	100	100	100	100
28d-60j-3c	70	90	60	100	90	90
28d-70j-1a	100	100	100	100	100	100

Continuation of Table 6.2

Cases	ByID	EcHigh	EcLow	Length	Random	Ga_Opt
28d-70j-1c	40	60	20	70	20	50
28d-70j-2a	100	100	100	100	100	100
28d-70j-2c	20	60	10	70	30	40
28d-70j-3a	100	100	100	100	100	100
28d-70j-3c	50	60	10	100	50	70
7d-10j-1a	100	100	100	100	100	100
7d-10j-1b	60	50	40	70	70	60
7d-10j-1c	90	100	90	100	100	100
7d-10j-2a	100	100	100	100	100	100
7d-10j-2b	50	50	40	70	50	50
7d-10j-2c	100	90	100	100	100	100
7d-10j-3a	100	100	100	100	100	100
7d-10j-3b	30	50	20	50	40	40
7d-10j-3c	100	100	100	100	100	100
7d-20j-1a	100	100	100	100	100	100
7d-20j-2a	100	100	100	100	100	100
7d-20j-3a	100	100	100	100	100	100
7d-30j-1a	100	90	80	100	70	80
7d-30j-2a	70	90	60	100	70	100
7d-30j-3a	50	90	80	100	60	80

Table 6.3: Average Computing Time of the Heuristic Algorithm, by scenarios.

Cases	ByID	EcHigh	EcLow	Length	Random	Ga_Opt
14d-20j-1a	0.23	0.68	0.26	0.27	0.25	20.82
14d-20j-1b	2.55	2.01	2.52	1.07	5.07	48.71
14d-20j-1c	0.25	0.65	0.27	0.26	0.25	26.38
14d-20j-2a	0.24	0.61	0.26	0.26	0.22	19.11
14d-20j-2b	2.57	1.29	4.62	0.27	1.71	27.87
14d-20j-2c	0.23	0.69	0.26	0.27	0.24	22.03
14d-20j-3a	0.24	0.65	0.25	0.25	0.23	22.19
14d-20j-3b	0.85	1.08	0.82	0.3	0.64	47.7
14d-20j-3c	0.26	0.67	0.29	0.26	0.25	24.62
14d-30j-1a	0.35	0.68	0.35	0.35	0.33	32.88
14d-30j-1c	0.76	1.05	1.09	0.73	1.44	52.67

Continuation of Table 6.3

Cases	ByID	EcHigh	EcLow	Length	Random	Ga_Opt
14d-30j-2a	0.3	0.77	0.33	0.33	0.3	35.87
14d-30j-2c	0.33	0.75	0.48	0.36	0.39	44.63
14d-30j-3a	0.31	0.74	0.33	0.35	0.32	32.67
14d-30j-3c	0.51	0.8	0.95	0.36	0.86	69.3
14d-40j-1a	0.38	0.82	0.43	0.45	0.37	45.58
14d-40j-1c	26.83	6.33	15.14	15.41	31.91	100.25
14d-40j-2a	0.42	0.94	0.45	0.44	0.39	44.78
14d-40j-2c	6.09	0.9	7.87	20.38	4.16	92.55
14d-40j-3a	0.38	0.85	0.43	0.44	0.38	45.5
14d-40j-3c	2.51	1.25	2.65	0.5	3.7	12.51
28d-50j-1a	0.52	0.91	0.59	0.56	0.53	78.72
28d-50j-1b	3.39	4.38	4.1	1.91	2.27	18.32
28d-50j-1c	0.53	0.95	0.7	0.6	0.54	91.49
28d-50j-2a	0.5	0.86	0.56	0.57	0.52	80.03
28d-50j-2b	5.1	1.24	2.34	0.6	3.91	141.62
28d-50j-2c	0.56	0.91	0.8	0.57	0.53	90.69
28d-50j-3a	0.51	0.91	0.59	0.58	0.51	80.28
28d-50j-3b	2.82	0.95	1.78	1.53	2.44	124.89
28d-50j-3c	0.54	0.98	0.58	0.6	0.54	97.41
28d-60j-1a	0.56	1	0.64	0.67	0.6	92.85
28d-60j-1c	0.83	1.28	1.52	0.65	0.76	140.12
28d-60j-2a	0.58	0.99	0.64	0.67	0.55	94.04
28d-60j-2c	0.97	1.03	2	0.67	1.12	129.91
28d-60j-3a	0.57	1.06	0.66	0.65	0.59	95.92
28d-60j-3c	1.22	1.12	2.09	0.67	1.13	132.26
28d-70j-1a	0.64	1.02	0.75	0.76	0.68	108.07
28d-70j-1c	2.78	2.45	3.28	0.78	1.99	212.82
28d-70j-2a	0.64	1.21	0.71	0.83	0.65	110.42
28d-70j-2c	4.35	2.44	5.67	0.84	3.85	97.24
28d-70j-3a	0.63	1.12	0.77	0.71	0.72	105.96
28d-70j-3c	2.53	1.88	4.58	0.79	1.66	180.44
7d-10j-1a	0.13	0.55	0.17	0.15	0.15	8.2
7d-10j-1b	0.18	0.64	0.29	0.17	0.29	12.14
7d-10j-1c	0.15	0.55	0.16	0.16	0.15	9.98
7d-10j-2a	0.14	0.63	0.14	0.15	0.14	9.46
7d-10j-2b	0.27	0.59	0.28	0.4	0.27	11.94
7d-10j-2c	0.17	0.49	0.2	0.18	0.14	12.08

Continuation of Table 6.3

Cases	ByID	EcHigh	EcLow	Length	Random	Ga_Opt
7d-10j-3a	0.14	0.49	0.14	0.17	0.14	8.57
7d-10j-3b	0.32	0.76	0.35	0.17	0.41	16.76
7d-10j-3c	0.16	0.54	0.2	0.15	0.14	13.28
7d-20j-1a	0.24	0.58	0.3	0.24	0.25	19.3
7d-20j-2a	0.22	0.63	0.26	0.25	0.23	18.85
7d-20j-3a	0.21	0.67	0.28	0.25	0.22	16.82
7d-30j-1a	0.37	0.65	0.46	0.34	0.33	41.92
7d-30j-2a	0.45	0.74	0.58	0.35	0.4	41.45
7d-30j-3a	0.5	0.8	0.59	0.44	0.59	38.25

Bibliography

[1] H. Aalami, M. P. Moghaddam, and G. Yousefi, *Demand response modeling considering interruptible/curtailable loads and capacity market programs*, Applied Energy, 87 (2010), pp. 243 – 250.

[2] S. Afshin Mansouri and E. Aktas, *Minimizing energy consumption and makespan in a two-machine flowshop scheduling problem*, Journal of the Operational Research Society, 67 (2016), pp. 1382–1394.

[3] C. N. Anulipt Chandan, Vidyasagar Potdar, *Pricing mechanisms for energy management in smart cities*, Computer Communications and Networks, (2018).

[4] S. Bakr and S. Cranefield, *Optimizing shiftable appliance schedules across residential neighbourhoods for lower energy costs and fair billing*, in AIH+CARE@AUS-AI, 2013.

[5] K. Biel, *Multi-stage production planning with special consideration of energy supply and demand*, PhD thesis, Technischen Universität Darmstadt, 2017.

[6] BP p.l.c., *Bp energy outlook edution 2019*, (2019).

[7] ——, *Bp statistical review of world energy 2019*, (2019).

[8] P. Brucker, *Scheduling Algorithms*, Springer Publishing Company, Incorporated, 5th ed., 2010.

[9] J. Busse, C. Rüther, and J. Rieck, *Energy cost-oriented scheduling with job prioritization*, 11 2018, pp. 514–520.

[10] J. Błażewicz, W. Domschke, and E. Pesch, *The job shop scheduling problem: Conventional and new solution techniques*, European Journal of Operational Research, 93 (1996), pp. 1 – 33.

[11] L. Davis, ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[12] R. de Sá Ferreira, L. A. Barroso, P. R. Lino, M. M. Carvalho, and P. Valenzuela, *Time-of-use tariff design under uncertainty in price-elasticities of electricity demand: A stochastic optimization approach*, IEEE Transactions on Smart Grid, 4 (2013), pp. 2285–2295.

[13] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, *A fast and elitist multiobjective genetic algorithm: Nsga-ii*, IEEE Transactions on Evolutionary Computation, 6 (2002), pp. 182–197.

[14] H. Emmons and G. Vairaktarakis, *Introduction*, Springer US, Boston, MA, 2013, pp. 1–19.

[15] K. Fang, N. A. Uhan, F. Zhao, and J. W. Sutherland, *Flow shop scheduling with peak power consumption constraints*, Annals of Operations Research, 206 (2013), pp. 115–145.

[16] J. fang Chen, L. Wang, and Z. ping Peng, *A collaborative optimization algorithm for energy-efficient multi-objective distributed no-idle flow-shop scheduling*, Swarm and Evolutionary Computation, 50 (2019), p. 100557.

[17] A. Fazli Khalaf and Y. Wang, *Energy-cost-aware flow shop scheduling considering intermittent renewables, energy storage, and real-time electricity pricing*, International Journal of Energy Research, 42 (2018), pp. 3928–3942.

[18] C. Gahm, F. Denz, M. Dirr, and A. Tuma, *Energy-efficient scheduling in manufacturing companies: A review and research framework*, European Journal of Operational Research, 248 (2016), pp. 744–757.

[19] B. Grosch, T. Weitzel, N. Panten, and E. Abele, *A metaheuristic for energy adaptive production scheduling with multiple energy carriers and its implementation in a real production system*, Procedia CIRP, 80 (2019), pp. 203 – 208. 26th CIRP Conference on Life Cycle Engineering (LCE) Purdue University, West Lafayette, IN, USA May 7-9, 2019.

[20] Y. S. Huang, R. and P. Chen, *Energy-Saving Scheduling in a Flexible Flow Shop Using a Hybrid Genetic Algorithm.*, Journal of Environmental Protection, 8 (2017), pp. 1037–1056.

[21] S. Jiang and L. Zhang, *Energy-oriented scheduling for hybrid flow shop with limited buffers through efficient multi-objective optimization*, IEEE Access, PP (2019), pp. 1–1.

[22] P. J. M. Laarhoven and E. H. L. Aarts, eds., *Simulated Annealing: Theory and Applications*, Kluwer Academic Publishers, Norwell, MA, USA, 1987.

[23] J. Luo, S. Fujimura, D. E. Baz, and B. Plazolles, *Gpu based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem*, Journal of Parallel and Distributed Computing, 133 (2019), pp. 244 – 257.

[24] S. N. Makhadmeh, A. T. Khader, M. A. Al-Betar, S. Naim, A. K. Abasi, and Z. A. A. Alyasseri, *Optimization methods for power scheduling problems in smart home: Survey*, Renewable and Sustainable Energy Reviews, 115 (2019), p. 109362.

[25] O. Masmoudi, X. Delorme, and P. Gianessi, *Job-shop scheduling problem with energy consideration*, International Journal of Production Economics, 216 (2019), pp. 12 – 22.

[26] S. Mei, X. Zhang, and M. Cao, *Power Grid Growth and Evolution*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 133–160.

[27] K. Miettinen, *Concepts*, Springer US, Boston, MA, 1998, pp. 5–36.

[28] A. Pilerood, M. Heydari, and M. Mazdeh, *A two-stage greedy heuristic for a flowshop scheduling problem under time-of-use electricity tariffs*, South African Journal of Industrial Engineering, 29 (2018), pp. 143 – 154.

[29] J. qing Li, H. yan Sang, Y. yan Han, C. gang Wang, and K. zhou Gao, *Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions*, Journal of Cleaner Production, 181 (2018), pp. 584 – 598.

[30] R. Ruiz and J. A. V. Rodríguez, *The hybrid flow shop scheduling problem*, European Journal of Operational Research, 205 (2010), pp. 1–18.

[31] S. Schulz, J. S. Neufeld, and U. Buscher, *A multi-objective iterated local search algorithm for comprehensive energy-aware hybrid flow shop scheduling*, Journal of Cleaner Production, 224 (2019), pp. 421 – 434.

[32] United Nations, *World population prospects 2019*, (2019).

[33] H. M. Wagner, *An integer linear-programming model for machine scheduling*, Naval Research Logistics Quarterly, 6 (1959), pp. 131–140.

[34] F. Werner, *Genetic algorithms for shop scheduling problems: A survey*, Preprint Series, 11 (2011), pp. 1–66.

[35] Y. Zhai, K. Biel, F. Zhao, and J. W. Sutherland, *Dynamic scheduling of a flow shop with on-site wind generation for energy cost reduction under real time electricity pricing*, CIRP Annals, 66 (2017), pp. 41 – 44.

[36] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, *Review of job shop scheduling research and its new perspectives under industry 4.0*, Journal of Intelligent Manufacturing, (2017).

[37] M. Zhang, J. Yan, Y. Zhang, and S. Yan, *Optimization for energy-efficient flexible flow shop scheduling under time of use electricity tariffs*, Procedia CIRP, 80 (2019), pp. 251 – 256. 26th CIRP Conference on Life Cycle Engineering (LCE) Purdue University, West Lafayette, IN, USA May 7-9, 2019.

[38] Z. Zhang, L. Wu, T. Peng, and S. Jia, *An Improved Scheduling Approach for Minimizing Total Energy Consumption and Makespan in a Flexible Job Shop Environment*, Sustainability, 11 (2018), pp. 1–21.

[39] Y. Zhou and J. jun Yang, *Automatic design of scheduling policies for dynamic flexible job shop scheduling by multi-objective genetic programming based hyper-heuristic*, Procedia CIRP, 79 (2019), pp. 439 – 444. 12th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 18-20 July 2018, Gulf of Naples, Italy.