U.PORTO

MASTER
MODELLING, DATA ANALYSIS AND DECISION SUPPORT SYSTEMS

Heuristics for the Distributed Permutation Flowshop Scheduling Problem with the Weighted Tardiness Objective

Maria Inês Ferraz Cerveira

M

2019

U.PORTO

FEP FACULDADE DE ECONOMIA UNIVERSIDADE DO PORTO

HEURISTICS FOR THE DISTRIBUTED PERMUTATION FLOWSHOP SCHEDULING PROBLEM WITH THE WEIGHTED TARDINESS OBJECTIVE

Maria Inês Ferraz Cerveira

Dissertation

Master in Modelling, Data Analysis and Decision Support Systems

Supervised by
Jorge Valente
Jeffrey Schaller

2019

Abstract

The Distributed Permutation Flowshop Scheduling Problem (DPFSP) is a recent topic on optimization, in which the literature is small (especially when compared to the Permutation Flowshop Scheduling Problem). It was introduced by Naderi and Ruiz, 2010, where the authors assume that there is a total of F identical factories or shops, each one with m machines disposed in series. A set of n available jobs has to be distributed among the F factories and then a processing sequence has to be derived for the jobs assigned to each factory.

In this dissertation, several heuristics are tested to minimize the weighted tardiness on the DPFSP: five general rules (EDD – Earliest due date; WEDD – Weighted earliest due date; MDD – Modified due date; SLK – minimum slack; WSLK – weighted minimum slack); seven rules specific for weighted tardiness (WSPT – weighted shortest processing time; WSLK_SPT – Weighted minimum slack/shortest processing time; two versions of AR – by Alidaee & Ramakrishnan, 1996; two versions of ATC – Apparent Tardiness Cost; WEDD – weighted earliest due date, adapted to this problem) and two new rules are developed for this problem (WSPT_EDD – weighted shortest processing time/earliest due date; WSPT_WEDD – weighted shortest processing time/weighted earliest due date). Three assignment rules are tested: ES (earliest start), ECM (earliest completion makespan) and EUM (earliest updated makespan).

All combinations of heuristics and assignment rules are tested, and the results are compared among themselves in effectiveness (solution quality) and efficiency (computational time. AR_v2 with EUM rule is the most effective combination heuristic/assignment rule. In general, assignment rule EUM is more effective, but requires more computational time.

Resumo

O problema *Distributed Permutation Flowshop Scheduling Problem* (DPFSP) é um tópico recente em otimização, com literatura escassa (principalmente quando comparado com o *Permutation Flowshop Scheduling Problem*). Foi introduzido por Naderi e Ruiz, 2010. Os autores assumem que existe um total de F fábricas ou lojas idênticas, cada uma com m máquinas em série. Um conjunto de n trabalhos disponíveis têm de ser distribuídos pelas F fábricas e depois a sequência de processamento tem de ser estabelecida para os trabalhos assignados a cada fábrica.

Nesta dissertação, várias heurísticas são testadas para minimizar o atraso médio no DPFSP: cinco regras gerais (EDD – *earliest due date*; WEDD – *weighted earliest due date*; MDD – *modified due date*; SLK – *minimum slack*; WSLK – *weighted minimum slack*), sete regras específicas para o atraso médio (WSPT – *weighted shortest processing time*; WSLK_SPT – *weighted minimum slack/shortest processing time*; duas versões de AR – de Alidaee & Ramakrishnan, 1996; duas versões de ATC – *Apparent Tardiness Cost*; WEDD – *weighted earliest due date*, adaptado ao problema) e duas novas regras são desenvolvidas para o problema (WSPT_EDD – *weighted shortest processing time/earliest due date*; WSPT_WEDD – *weighted shortest processing time/weighted earliest due date*). Três regras de atribuição são testadas: ES (*earliest start*), ECM (*earliest completion makespan*) and EUM (*earliest updated makespan*).

Todas as combinações de heurísticas e regras de atribuição são testadas e os resultados comparados entre si para eficácia (qualidade da solução) e eficiência (tempo de processamento). AR_v2 com a regra EUM é a combinação heurística/regra de atribuição mais eficaz. No geral, a regra de atribuição EUM é mais eficaz, mas exige maior tempo de processamento.

Table of Contents

Introduction

The Distributed Permutation Flowshop Scheduling Problem (DPFSP from now on) is a recent topic on optimization, in which the literature is small (especially when compared to the Permutation Flowshop Scheduling Problem). It was first introduced by Naderi and Ruiz in 2010. I see this problem as a challenge, but mostly an opportunity to build upon something so complex and unexplored and try to do something new and valuable to this area of study, namely by trying to find efficient heuristics to measure the effectiveness of the DPFSP, using as optimization criteria the minimization of the total weighted tardiness (sum of weighted tardiness). The weighted tardiness is a standard way of measuring compliance with the due dates. If these dates are not met, the company may face penalty costs and dissatisfaction of customers.

In this work, I do a literature review of what has been studied about DPFSP and similar scheduling problems and adapt existing heuristics for the Permutation Flowshop Scheduling Problem to the DPFSP. I use the C++ programming language, with the Visual Studio software. I evaluate my results comparing the results of all heuristics I develop among themselves, analyzing which ones obtain the best and worst results in effectiveness (solution quality) and efficiency (computational time).

In the next chapters, I first do an extended review of the existing literature detailing the notation, instances, and heuristics used for each iteration of the model since the FSP to the DAPFSP. Then the DPFSP with tardiness minimization will be described and the notation, instances to be used and heuristics presented. Finally, the results are processed, presented and evaluated.

Literature Review

1. FSP and PFSP

a) Definition, assumptions and restrictions

Manufacturing and service systems can be explained using several characteristics like the number of machines or resources and their configuration and characteristics. To better understand this work, it is important to remember the following definitions and notations (Table 1) (Pinedo & Chao, 1999):

Table 1: Basic definitions and notations

Term/Notation	***Definition***
Machine	Resource used in manufacturing or service. Can be a car, a factory's machine, a hotel room, among others.
Job	Entity that has to be processed on a machine. Can be the rental of the car, a product that needs to be made, the use of a hotel room, among others.
m	Number of machines.
n	Number of jobs.
f	Number of factories.
j **and** k	Subscripts related with jobs.
h **and** i	Subscripts related with machines.

The scheduling problem can be defined as allocating the available resources (typically machines) to tasks (commonly referred as jobs) over time to optimize a given objective and it's mostly used in manufacturing industries (Bahman Naderi & Ruiz, 2014). It is also useful in multiple other domains, such as logistics and computer design. Over the last six decades, a great number of papers by multiple authors have been published on the matter (Bargaoui, Belkahla Driss, & Ghédira, 2017).

Empirically, it is fairly known that good schedules contribute greatly to the overall performance of a company (Bahman Naderi & Ruiz, 2014). To determine the type of scheduling problem to solve, a great number of factors must be taken into consideration, such as the layout of the machines on the production floor, the flow of the jobs in the machines and a myriad of constraints and real-life settings.

A dispatching rule prioritizes all the jobs waiting to be processed. Basic dispatching rules can be characterized in several ways, examples of this characterization can be found in Table 2. They can be used in models with a single objective (they are too simple for multi-objective models).

Table 2: Basic dispatching rules characterization

Term	***Definition***
Static rules	Function of job and/or machine data.
Dynamic rules	Time-dependent. Priority of a job may be variable.
Local rules	Uses information from the queue of jobs or machine where the job is queued.
Global rules	May use any information of the model.

Among all the existing processing layouts in practice, the Flowshop Scheduling Problem (FSP from now on) is a very common and active one, as it is typical for manufacturing plants to manufacture a given family of products that have to visit machines in a known order (Bahman Naderi & Ruiz, 2014). A good practical example of this is car manufacturing in which the painting of the car body has to be performed after the welding of said body, but before any kind of assembly operation, thus creating a flowshop structure (Bahman Naderi & Ruiz, 2014).

The basic FSP is formalized as follows: a set of unrelated jobs are to be processed on a set of machines. These machines are disposed in series and each job must visit all of them in the same order. This order can be assumed, without loss of generality. Therefore, each job is composed of tasks. The processing times of the jobs at the machines are known in advance, non-negative and deterministic (B. Naderi & Ruiz, 2010). Like any model, the FSP entails several assumptions:

1. All jobs are independent and available for processing at time 0;
2. Machines are continuously available (no breakdowns);
3. Each machine can only process one job at a time;
4. Each job can be processed only on one machine at a time;
5. Once the processing of a given job has started on a given machine, it cannot be interrupted, and processing continues until completion (no preemption);
6. Setup times are sequence independent and are included in the processing times, or ignored;

7. Infinite in-process storage is allowed (B. Naderi & Ruiz, 2010).

The goal of the FSP is to obtain a production sequence of the jobs on the machines so that a given criterion is optimized (B. Naderi & Ruiz, 2010). Therefore, the problem here is to define in which order each job is processed on each machine (Victor Fernandez-Viagas & Jose M. Framinan, 2015). The most used criteria are the minimization of the maximum completion time or makespan (Bahman Naderi & Ruiz, 2014) and his focus is the fast processing of the products and balanced use of resources. However, some works, such as Raman (1995) already used the minimum tardiness criteria (used for this work) as customer due dates became a critical concern for many manufacturing systems (Raman, 1995).

One of the biggest issues with the FSP is that there are $n!$ possible job permutations for each of the m machines, which means that there are $n!^m$ possible sequences/solutions for the problem (B. Naderi & Ruiz, 2010). The most used solution for this problem is to reduce the search space by not allowing job passing (once a production sequence is determined for the first machine, it is maintained for all other machines). This way, the number of possible solutions is reduced to $n!$. This variant of the FSP called the Permutation Flowshop Scheduling Problem (PFSP) is the most studied (Sara Hatami, Rubén Ruiz, & Carlos Andrés-Romano, 2015). According to Molina-Sánchez et al, 2016, the PFSP is realistic and often appropriate for real-world applications, since the in-process storage of products is very limited in most situations. A good example of this are factories with convoys between machines to transfer materials and assembly lines whose final product are bulky products (Molina-Sánchez & González-Neira, 2016).

However, PFSP is, still, a very difficult combinatorial optimization problem where a set of n jobs has to be processed on a set of m machines in the same order (Cura, 2015). Figure 1 illustrates the organization of this factory.

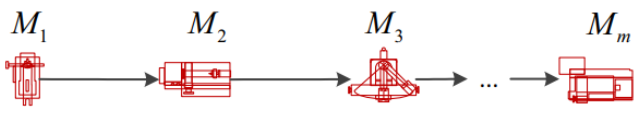

Figure 1: Representation of the Permutation Flowshop Scheduling Problem, (J. Lin & Zhang, 2016), adapted

This is usually a NP-hard problem: it cannot be formulated as a linear program and solved to optimality in a limited computer time with any simple rule or algorithm. If we have a large amount of computer time, we can formulate them as integer or disjunctive programs.

However, in the real world, computer time is usually limited, so one must accept a reasonably good and feasible solution, that, ideally, is not far from the optimal (Pinedo & Chao, 1999).

b) Heuristics and Metaheuristics

Before starting this section, it is highly appropriate to set a definition of both heuristic and metaheuristics as they'll be key aspects of understanding and solving each of the models.

A heuristic is an approach to problem-solving, that doesn't consider part of the information. It doesn't guarantee an optimal or perfect solution, but enough for the immediate goals. They can be of the constructive or improvement type. The first ones construct a schedule, adding one job at a time. A metaheuristic improves an already existent schedule (Pinedo & Chao, 1999).

According to Osman et al, 1996 a metaheuristic guides a subordinate heuristic using concepts derived from artificial intelligence, biological, mathematical, natural and physical sciences to improve their performance (I.H. Osman & Kelly, 1996).

As mentioned before, the literature on PFSP is large. In the next paragraphs, I will focus on some of the works with objectives related with tardiness.

In 2010, Vallada et al published an article about minimizing the total tardiness. They present three genetic algorithms for bio-inspired optimization methods that are commonly used to solve combinatorial problems such as the PFSP, however, despite showing very good performance for the makespan objective its use for the total tardiness is very limited (Vallada & Ruiz, 2010). These methods use advanced techniques like path relinking, a technic used to explore the search space between two individuals of the population by defining one of the select solutions as the *origin* and another as the *destination* and each time a movement is carried out the obtained solutions look more like the *destination* and less like the *origin*. Local search, widely used in genetic algorithms to improve solutions as they´re able to carry out a fine improvement that genetic algorithms don't, by using a method such as job insertion. The procedures include a procedure to control the diversity of the population and then add a speed up the procedure to "reduce the computational effort needed for the local search technique, which results in large CPU time savings" (Vallada & Ruiz, 2010). The authors evaluated their results by comparing them with the best methods that can be found in the literature for the total tardiness objective, and with adaptations of other state-of-the-art methods originally proposed for other objectives (Vallada & Ruiz, 2010).

Fernandez-Viagas et al, 2015 proposed NEH-based heuristics to minimize total tardiness. In order to do that the authors used an algorithm-labeled bounded-search iterated greedy (BSIG), which is a special case and evolution of the more common iterated greedy algorithm (IG) which has already been proven to be one of the best algorithms for solving PFSP problems (Victor Fernandez-Viagas & Jose M. Framinan, 2015) and works by *starting with an initial solution and then iteratively applying four steps. First, several jobs are taken out of the sequence. Then these jobs are again inserted in the sequence, one by one, following a greedy procedure until no more job must be inserted. After that, a local search mechanism is performed in order to improve the solution. Finally, a simulated annealing procedure decides if the actual sequence is kept as iteration sequence* (Victor Fernandez-Viagas & Jose M. Framinan, 2015). They also develop a tie-breaking mechanism to optimize their results. To solve the same problem, Cura, 2015 used an evolutionary algorithm that included a mating procedure (according to Sarker et all,2002 fitter individuals should have higher probabilities of being selected, while unfit ones should only be selected with small probabilities to search for increasingly better individuals). This method is used to recombine two or more parent individuals of the population in the study and is specifically designed for the problem, a local search with two different neighbourhood sizes, and a revision procedure (Cura, 2015). The author used benchmark problems to test the results.

Karabulut, 2016 proposed an iterated greedy algorithm to minimize total tardiness and tested his method on a set of benchmark problems from the literature and compared it to three versions of the traditional iterated greedy algorithm using the same problem instances (Karabulut, 2016).

Molina-Sánchez et al, 2016 studied a GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristic with the objective of minimizing the total weighted tardiness. The authors also proposed two utility functions to implement and modify the GRASP, these functions are adaptions of two well-known dispatching rules and are called Weighted Modified Due Date (WMDD) and Apparent Tardiness Cost (ATC). They assessed the quality of the proposed solutions, comparing them with two different utility functions (Molina-Sánchez & González-Neira, 2016).

c) Instances and distributions

In order to compute, process and test the model's accuracy and efficiency it is needed for the authors of the papers previously mentioned to select the instances, due dates distributions and process times distributions that are going to be used to test their algorithms.

Vallada & Ruiz, 2010 selected a testbed used by them on another paper (Vallada, Ruiz, & Minella, 2008) composed of $\{50,150,250,350\}$ jobs and $\{10,30,50\}$ machines, processing times are uniformly distributed between 1 and 99 $(1,99)$ and due dates follow a uniform distribution between $P(1-T-R/2)$ and $P(1-T+R/2)$, where P is a lower bound of the makespan and T and R are two parameters called the tardiness factor and due date range which take the following values: $T = \{0.2, 0.4, 0.6\}, R = \{0.2, 0.6, 1\}$ (Vallada & Ruiz, 2010). The authors tested each of their algorithms with a set of these instances. This test instance is one of the most used across any flowshop model, including the already referred (Victor Fernandez-Viagas & Jose M. Framinan, 2015) and (Karabulut, 2016) and (Molina-Sánchez & González-Neira, 2016).

For further investigation on the topic PFSP, I recommend consulting the following reviews: (Rubén Ruiz & Maroto, 2005) and (Yenisey & Yagmahan, 2014). To learn more about total tardiness, there is a review of the m-machine flowshop problem (Vallada et al., 2008).

2. DPFSP

a) Definition

One main assumption of the PFSP, and one of its major setbacks, is that there is only one production centre, factory or shop, which means that all jobs are assumed to be processed in the same factory (B. Naderi & Ruiz, 2010). Empirically, it has been shown in economic studies that single factory firms are less and less common, with multi-plant companies and supply chains taking a more important role in the practice as distributed manufacturing enables companies to achieve higher product quality, lower production costs and lower management risks (B. Naderi & Ruiz, 2010).

The first time anyone tried to apply the distributed generalization of the PFSP to multiple factories was in Naderi et al, 2010 when the authors suggest the Distributed Permutation Flowshop Scheduling Problem (DPFSP) (B. Naderi & Ruiz, 2010) that will be the centre of this dissertation.

According to Naderi et al, 2010 when studying scheduling in distributed systems a new problem arises as, unlike in the single factory setting where the problem is to schedule n jobs on a set of m machines, in the distributed systems we face two problems: job assignment to factories and job scheduling at each factory. This is an especially sensible problem since both problems are intimately related and cannot be solved sequentially if high performance is desired (B. Naderi & Ruiz, 2010). Naderi et al, 2010 formalize the problem as follows: a set N of n jobs have to be processed on a set G of f factories, each one of them containing the same set M of m machines, just like in the regular PFSP. All factories are capable of processing all jobs. Once a job $j, j \in N$ is assigned to a factory $f, f \in G$ it cannot be transferred to another factory as it must be completed at the assigned plant. They assumed that processing times do not change from factory to factory. Figure 2 represents the described production environment.

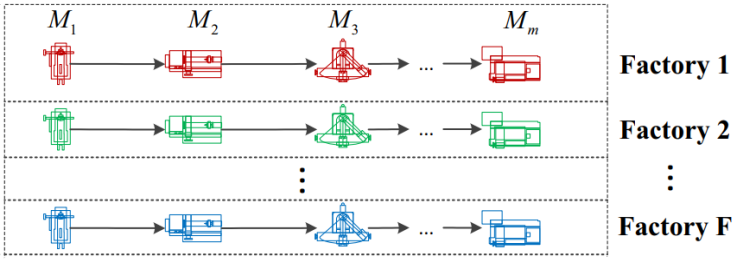

Figure 2: Representation of the Distributed Permutation Flowshop Scheduling Problem, (J. Lin & Zhang, 2016), adapted

The number of possible solutions for the DPFSP is tricky to calculate. If we were only accounting the number of solutions for a given known single partition of jobs into factories it would be $n!$. However, in order to know the total number of solutions in the DFPSP, we need to know the number of ways of partitioning n jobs into F factories and define their sequence. Considering that every factory should have at least one job this number becomes $\binom{n-1}{F-1}n!$, significantly greater than the number of solutions on the PFSP (B. Naderi & Ruiz, 2010).

b) Heuristics and Metaheuristics

Most works done in this area have the objective of minimizing the maximum makespan. Naderi et al, 2010 proposed six different alternative mixed integer linear programming (MILP) models and analyse their performance. The authors also propose fourteen heuristics based on dispatching rules, effective constructive heuristics and variable neighbourhood descent methods (local search method based on the systematic change of the neighbourhood during the search process), with two factory assignment rules:

1. Assign job j to the factory with the lowest current max completion time or makespan (C_{max}) not including job j.
2. Assign job j to the factory which completes it at the earliest time, i.e., the factory with the lowest C_{max}, after including job j (B. Naderi & Ruiz, 2010).

In 2010, Liu & Gao presented a discrete electromagnetism-like mechanism algorithm for solving the Distributed Permutation Flowshop Scheduling Problem with the makespan criterion. In order to do so, they modified the original EM by redefining distance, movement, charge and calculation. The results show that the new, modified EM algorithm surpasses the original one (H. Liu & Gao, 2010).

Gao and Chen, 2011 published two distinct works on this topic. The first one uses a genetic algorithm where the individuals are ranked according to the fitness selected with an efficient local search method to explore neighbouring solutions. This local search improves upon the traditional variable neighbourhood search by applying 3 different rules: Insertion Jobs, Exchange Jobs and Move Jobs. It is important to note that while Insertion Jobs are designed for a single factory, the other ones are used to move jobs between factories (Gao & Chen, 2011a). The efficiency of the approach is measured by comparing with similar genetic algorithms with the existing local search methods and the goal is to minimize the maximum completion time (Gao & Chen, 2011a).

The second work is a modified NEH-algorithm followed by an iterative search to solve this problem. The NEH-algorithm is based on sorting all jobs by decreasing sums of processing times for the jobs on all machines and, for the k^{th} job, finds the best position among k possible ones that minimize the partial makespan, then inserts it into that position. The modification the authors do is, given multiple factories in a DPFSP, the dispatching rule

will insert a group of jobs to the factories at one time instead of inserting one job at one time like the original rules. Intensive benchmark experiments are implemented on the large problem instances to validate the proposed heuristic (Gao & Chen, 2011b).

In 2013, Deng published a paper with the two authors mentioned before, where they apply the tabu search algorithm, followed by local search (Gao, Chen, & Deng, 2013). Tabu search is a metaheuristic that attempts to find a better solution than the current one in its neighbourhood, unlike local search it uses memory structures. A benchmark is used to test the algorithm.

Lin et al, 2013 proposed a modified iterated greedy algorithm (MIG) with a new acceptance criterion and a variable number of removed elements. The greedy algorithm looks for a local optimum on each stage of the process, not considering the whole problem and has the goal of minimizing the maximum completion time among all the factories. The authors used an extended benchmark of Taillard in order to evaluate and validate the performance of the MIG algorithm and found out that, despite being rather simple, the computational results proved that it outperformed all existing algorithms at the time and allowed it to find new best-known solutions in almost half the cases of the DPFSP benchmark suite extended from the Taillard instances, furthermore the differences in the computational results are also statistically significant. This study was based on a practical case. (S.-W. Lin, Ying, & Huang, 2013)

Wang et al, 2013 presented an EDA (estimation of distribution algorithm) approach to the DPFSP with the criterion to minimize the makespan. To generate feasible schedules, the authors assumed the earliest completion factory rule. A probability model is used for describing the probability distribution of the solution space and added to the use of a mechanism it allows to update the probability model with superior individuals. A local search procedure is then conducted (according to the problem characteristics) and used to sort promising individuals. The model was validated through extensive testing results and comparisons and proved the effectiveness of the algorithm in solving the DPFSP which hold especially true for the big instances used on the test where new best-known solutions for 589 of the total 720 were found (S. Y. Wang, Wang, Liu, & Xu, 2013).

In 2014, Naderi and Ruiz published a new work where they apply a scatter search algorithm. The scatter search algorithm had already been used for flowshop problems,

however, the authors here added some new advanced techniques, such as local search, restarts and a set made up of complete and partial solutions. According to the authors, all 720 known best solutions for this problem are improved and, at the time, this algorithm produced better results than any other used to solve the DPFSP (Bahman Naderi & Ruiz, 2014).

Xu et all, 2014 developed a study for solving the distributed permutation flow-shop scheduling problem using an effective hybrid immune algorithm. A decoding method is used to transfer a job permutation sequence to a feasible schedule considering both factory dispatching and job sequencing. (Xu, Wang, Wang, & Liu, 2014)

Deng et al, 2014 present at the 2014 IEEE Internal Conference on Automation Science and Engineering a competitive memetic algorithm for solving the distributed flowshop scheduling problem. In this method multiple sub-populations are employed to perform a search among the solution space with different neighbourhoods, those sub-populations then compete with each other for generating a different number of their own offspring in every generation. Local search is also used to intensify local exploration. (J. Deng, Wang, & Wang, 2014)

Fernandez-Viagas et al, 2015 developed a bounded-search iterated greedy algorithm. On the bounded-search algorithm, one must enumerate all solutions, starting from the root and building a tree. The greedy algorithm searches for the local optimal at each stage/branch and the goal of the algorithm is to minimize the total makespan using the attribution rule from Naderi and Ruiz,2010 for such. The authors performed computational experiences on instances presented on the original paper of DPFSP by Naderi & Ruiz (V. Fernandez-Viagas & J. M. Framinan, 2015).

At the 2016 IEEE Congress on Evolutionary Computation (CEC), three teams of researchers presented papers related with DPFSP, with the objective of minimizing maximum makespan.

Wang et al, 2016 presented a hybrid discrete cuckoo search followed by a variable neighbourhood search. In the hybrid discrete cuckoo search, the permutation based encoding scheme is adopted and the earliest completion factory rule is employed to map the

individuals into feasible schedules (J. Wang, Wang, & Shen, 2016). The results are compared with existing algorithms.

Li et al, 2016 propose a variation of the DPFSP, with changes on a logistic level of each factory: each factory has its own transport timetable and loading capacity and distances to different factories are not equal. They use simulated annealing based local search with multiple different neighbourhoods to solve this problem. The performance test is done with a comprehensive computational and statistical analysis (Z. Li et al., 2016).

The last paper on this topic of CEC 2016 was presented by Duan et al., who considered the flowtime eligibility constraint (factories aren't always available) and proposed a hybrid estimation of distribution algorithm, followed by a variable neighbourhood search. In the end, the authors did numerical simulations to test the heuristic (Duan et al., 2016).

At the 2017 IEEE Congress on Evolutionary Computation, Duan et al considered the machine blocking and job sequence-dependent setup time constraints. The authors proposed an EDA-based (Exploratory Data Analysis) probabilistic memetic framework (framework that controls the learning intensity of each individual and balances the exploration and exploitation), called EDAPrMF, to solve the problem. Experimental results and comparisons with existing algorithms were used to validate the solution (Duan et al., 2016).

Ying et al, 2017 proposed an iterated Greedy Reference algorithm to solve the DPFSP with a no-idle constrain. The authors validate the performance comparing it with a state-of-the-art iterated greedy algorithm and the Mixed Integer Linear Programming model on two benchmark problem sets. Computational results show that the proposed iterated Greedy Reference algorithm outperforms the iterated greedy algorithm (Ying, Lin, Cheng, & He, 2017).

In 2017, Bargaoui et al developed a Chemical Reaction Optimization to minimize the maximum makespan. Chemical Reaction Optimization is a method known for escaping local optima and obtaining excellent results. In this study, an artificial chemical reaction metaheuristic is used on the DPFSP to generate an initial population of molecules and combined with a One-Point crossover and an effective greedy strategy in order to get better quality solutions. The authors evaluate the results making intensive experiments on 720 large

instances. The results of more than two hundred best-known solutions are improved (Bargaoui et al., 2017).

In 2018 another work is published with the flowtime criterion by Pan et all. On this work Pan et all, 2018 use a discrete artificial bee colony to minimize the total flow time on the DPFSP. (J. Q. Pan, Zou, & Duan, 2018)

In 2018 Chen and Wang publish a study on a two-stage memetic algorithm to solve the Distributed No-idle Permutation Flowshop Scheduling Problem using the makespan criteria. A hybrid method is used to generate the initial population and, in order to balance the exploration and exploitation, two search stages are specially designed, and in each stage, multiple problem-specific operators are utilized to adjust factory assignment and job sequences. (Ling-Fang, Ling, & Jing-Jing, 2018)

In 2018, one more article with the total flowtime minimization on the DPFSP was published by Viagas et all. Here the authors use NEH based constructive heuristics to solve the problem and a simple evolutionary algorithm to further improve the solution found (Fernandez-Viagas, Perez-Gonzalez, & Framinan, 2018)

In 2019, Quan-Ke Pan et al published a study on effective heuristics and metaheuristics for the distributed permutation flowshop scheduling problem with the flowtime criterion. The constructive heuristics are adapted from two high-performing heuristics for the flowtime criterion, LR (Liu and Reeves, 2001) and NEH (Nawaz et al., 1983) while the metaheuristics are based on the high performing frameworks of discrete artificial bee colony, scatter search, iterated local search, and iterated greedy search, which has been applied with great success to other scheduling problems. The results show that both heuristics and metaheuristics are effective in solving the DPFSP with the total flowtime criterion (Q. K. Pan, Gao, Wang, Liang, & Li, 2019).

In 2019, Ruiz et al published a work where they present iterated greedy methods for the distributed permutation flowshop scheduling problem. The objective was to minimize the completion makespan across all the factories by using iterated Greedy algorithms which are simply an iterated search method with no memory and few structures, being very simple to replicate and extend to other problems and achieving good performances for different flowshop problems and variants. (R. Ruiz, Pan, & Naderi, 2019)

To the best of my knowledge, there are only three works about DPFSP that do not have as a single objective the minimization of the maximum makespan or flowtime.

Deng et al, 2017 use a competitive memetic algorithm for the multi-objectives of minimizing the maximum makespan and the total tardiness. According to the authors, the memetic algorithm combines evolutionary search with the problem-specific local search to balance the exploration and exploitation reasonably. The effectiveness of the competitive memetic algorithm is measured with extensive computational tests and comparisons (Jin Deng & Wang, 2017).

Cai et al, 2018 presented the first study about multi-objective optimization for the DPFSP with transportation and eligibility constraints. The three objectives considered are the makespan, the maximum lateness and the total costs which include transportation and setup costs. The authors use eight heuristics for generating job permutations: Shortest processing time (SPT), Longest Processing Time (LPT), Johnson, Palmer, earliest releasing time (ERT), earliest due date (EDD) and a new heuristic based on NEH, NEHA and propose an improved Nondominated Sorting Genetic Algorithm II (NSGA-II) to find Pareto optimal solutions (Cai, Yang, & Liu, 2018).

In 2018, Jian et al published a study for the Modified Multi-objective Evolutionary algorithm based on decomposition for a modified version of the DPFSP called distributed permutation flow-shop low-carbon problem with two criteria: minimizing makespan and carbon emission (Jiang, Wang, & Lu, 2018).

c) Instances and distributions

For their pioneer work in studying the DPFSP, (B. Naderi & Ruiz, 2010) use 5 instances for each of the possible 84 combinations, resulting in a total of 420 small instances composed of $\{4,6,8,10,12,14,16\}$ jobs, $\{2,3,4,5\}$ machines and $\{2,3,4\}$ flowshops. The processing times are uniformly distributed between $(1,99)$. This test bed created by (B. Naderi & Ruiz, 2010) has proven itself to be hugely popular, being used by (S.-W. Lin et al., 2013) (S. Y. Wang et al., 2013), (Bahman Naderi & Ruiz, 2014), (Victor Fernandez-Viagas & Jose M. Framinan, 2015), (J. Wang et al., 2016), (Z. Li et al., 2016), (Gao & Chen, 2011a); (Gao et al., 2013); (Pan et all, 2019); (Xu et all,2019); (Deng et all, 2019); (Liu & Gao, 2010); (Ruiz et all,2019); (Pan et all,2018) all use the Taillard benchmark with up to 16 jobs and up to 5 machines in a total of 720 instances with processing times distribution uniformly distributed between $(1,99)$. The same benchmark is also used by (Bargaoui et al., 2017). This test bed by Taillard (1993) was also adapted by Naderi & Ruiz,2010, being composed of 720 large instances based on the 120 instances with 12 sets with the following different combinations of number of jobs n and number of machines m $n \times m = \{(20,50,100) \times (5,10,20), \{200 \times (10,20)\}$ and 500×20. Each of these combinations has 10 replicates and therefore 120 instances in total. All these 120 instances are considered with a different number of factories ($F = \{2; 3; 4; 5; 6; 7\}$) resulting in 720 instances.

(Duan et al., 2016) uses a test bed with 15 small instances and 15 large instances with $\{4,6,8\}$ jobs, $\{2,3,4\}$ machines and $\{2,3\}$ flowshop for small instances. Parameter values for large instances are not specified, neither are distributions for due dates or processing times.

(Ying et al., 2017) uses two different benchmarks: (B. Naderi & Ruiz, 2010) for small instances and (Sara Hatami et al., 2015) for large instances. As for the processing times distribution they use the proposed distribution by (B. Naderi & Ruiz, 2010).

(Jin Deng & Wang, 2017) use a benchmark by (Vallada et al., 2008) with a combination of jobs and machines ($n \times m = \{20,50,100\} \times \{5,10,20\}$ and $200 \times \{10,20\}$, $\{2,3,4,5,6,7\}$ flowshops and 660 (110×6) instances). The processing times are uniformly distributed between $(1,99)$.

Cai et al, 2018 use a test bed composed of 150 small instances were $n = \{4, 6, 8, 10, 12, 14, 16\}$; $m = \{2, 3, 4, 5\}$; $F = \{2, 3, 4\}$ and M is a $0 - 1$ matrix generated randomly which makes sure that the sum of each line is at least one. Thirty random combinations of $n \times m \times F$ are selected and for each combination, 5 different M are generated. They also use 250 large instances where $n = \{20, 50, 100, 200, 500\}$; $m = \{5, 10, 20\}$; $F = \{2, 3, 4, 5, 6, 7\}$ with 50 combinations of n × m × F and for each combination, 5 different M are generated (Cai et all,2018)

Jiang et al,2018 use a test bed of $n = \{20,40,60,80,100\}; m = \{4,8\}; F = \{2,3\}; v = \{1,1.3,1.55,1.75,2.1\}$ where v represents the machines speed which influences carbon emission (Jing et all,2018).

Chen and Weng, 2018 use 420 small instances with $n = \{4, 6, 8, 10, 12, 14, 16\}$, $m = \{2, 3, 4, 5\}$ and $F = \{2, 3, 4\}$. They also use 660 large instances where $n \times m = \{20, 50, 100\} \times 5$, $\{20, 50, 100, 200\} \times 10$, $\{20, 50,100, 200\} \times 20$ and for each combination $F = \{2, 3, 4, 5, 6,7\}$ (Chen and Weng,2018).

Viagas et all, 2018 use both the instances used by Naderi & Ruiz,2010 to compare among constructive heuristics and a test bed of 600 instances where $n = \{20,50,100,200,500\}$, $m = \{5,10,15,20\}$, and $F = \{2,3,4,5,6,7\}$, with five instances for each combination for the experimental parameter calibration (Viagas et all,2018).

3. DAPFSP

a) Definition

The Distributed Assembly Permutation Flowshop Scheduling Problem (DAPFSP) was first introduced by Hatami et al, in 2013. It is a generalisation of the DPFSP that combines the DPFSP and the Assembly Flowshop Scheduling Problem and consists of two stages: production and assembly (Sara Hatami, Ruiz, & Andrés-Romano, 2013). The first stage is similar to DPFSP. The second stage is a singular assembly factory with one assembly machine, M_A. Product assembly can start only when all jobs have been completed in the factories (Sara Hatami et al., 2013). Figure 3 illustrates this problem.

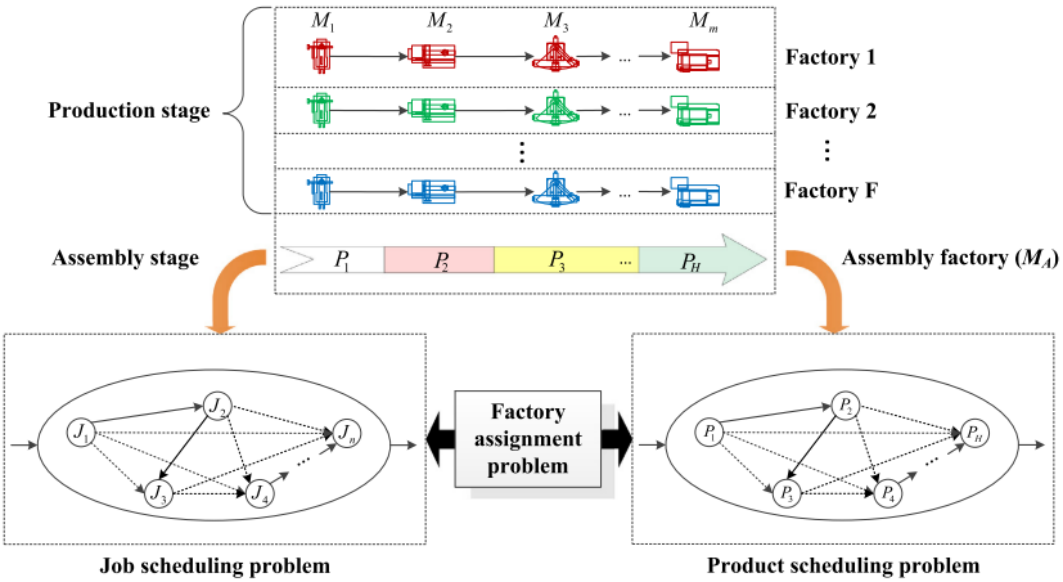

Figure 3: Representation of DAPFSP, (J. Lin & Zhang, 2016)

b) Heuristics and Metaheuristics

One can observe a great similarity between DPFSP and DAPFSP. As in DPFSP, most literature on DAPFSP has the objective of minimizing maximum makespan.

In the first paper about this topic, the authors consider the objective of minimizing the makespan at the assembly factory and present a Mixed Integer Linear Programming model, propose three constructive algorithms and design a variable neighbourhood descent algorithm. The attribution rule used in this paper is the one proposed by (B. Naderi & Ruiz, 2010). The algorithm is tested by a comprehensive ANOVA statistical analysis (Sara Hatami et al., 2013).

In 2014, the same team presented at the 2014 International Conference on Control, Decision and Information Technologies (CoDIT), two simple constructive heuristics for the DAPFSP with sequence-dependent setup times in production and assembly stages and the objective of minimizing overall makespan, (B. Naderi & Ruiz, 2010) attribution rule was also used. The results are tested on small and large instances of two generated sets (S. Hatami, Ruiz, & Andrés-Romano, 2014).

In 2015, Hatami et al published a paper with the same objective and constraints. The authors review the two heuristics mentioned before and two metaheuristics – Variable Descendent Neighbourhood and Iterated Greedy algorithm – to solve it. Once again, similar to previous works, (B. Naderi & Ruiz, 2010) attribution rule was employed. (Sara Hatami et al., 2015). Computational tests are run to evaluate the proposed approaches.

Li et al, 2015 proposed a genetic algorithm for the DAPFSP with the objective of minimizing makespan. An enhanced crossover strategy and three different local searches were adopted. The attribution rule used was the one by (B. Naderi & Ruiz, 2010) and as for assignment rules, two were defined: on the first one, the jobs needed for a product are processed successively in each factory. On the second rule, the order of processing jobs in factories corresponds to the order of making products. That is the jobs needed for the first product to be made are processed first. (X. Li, Zhang, Yin, & Wang, 2015). The authors obtained experimental results of the genetic algorithm and compared them with other previous algorithms.

At the 2016 IEEE Congress on Evolutionary Computation, two works on DAPFSP with the objective of minimizing makespan were presented.

Liu et al presented a possible solution for the mentioned problem: a memetic algorithm based on Variable Neighbourhood Search. The authors applied an efficient initialization based on the NEH heuristic for sequencing the order of products. New intra-product and inter-product neighbourhoods are proposed and incorporated into a Variable Neighbourhood Search (B. Liu, Wang, & Zhang, 2016). The validation of results was made with simulations on benchmarks and comparison with a solution published earlier.

Ji et al proposed a PSOSAHT methodology to solve the DAPFSP with a no-wait constraint, stochastic time for processing and assembly and the objective of minimizing makespan. PSOSAHT is a memetic algorithm that integrates particle swarm optimization (PSO), simulated annealing (SA) and hypothesis test (HT). Both attribution and assignment rules are randomly generated by PSO. Computational results and performance comparisons were used to evaluate the model (Ji, Yang, Duan, Wang, & Liu, 2016).

In 2016, Lin et al published a paper for the DAPFSP with the makespan minimization objective. The team proposed an effective hybrid biogeography-based optimization (HBBO) algorithm that integrates several novel heuristics. A biogeography-based algorithm is inspired by the immigration and emigration process of species among habitats. In this work, the migration operation takes the form of path relinking. The mutation operator is modified by an insertion-based heuristic. The attribution rule used is from (B. Naderi & Ruiz, 2010) and the priority rule is determined using an insertion based heuristic. The effectiveness is measured with simulations on small and large instances (J. Lin & Zhang, 2016).

At the 2016 IEEE Symposium Series on Computational Intelligence, Du et al presented a stochastic DAPFSP with a no-wait constraint in the processing stage. The random variables were processing and assembly times, sequence-dependent setup times (SDST) on processing machines and job release times. The objective was to minimize the makespan. The authors adapt the PSOSAHT method mentioned before to solve this problem and evaluate its efficiency and effectiveness through computational tests conducted on benchmark problems (Du, Ji, Li, & Liu, 2017).

Gonzalez-Neira et al, 2017 proposed a hybrid algorithm that integrates biased randomization and simulation techniques inside a metaheuristic framework to solve the stochastic DAPFSP. The random variables were processing and assembly times. The authors validate the performance of the algorithm in the deterministic version of the DAPFSP and stochastic version of the PFSP before applying it in the stochastic DAPFSP (Gonzalez-Neira, Ferone, Hatami, & Juan, 2017).

At the 2017 IEEE Congress on Evolutionary Computation (CEC 2017), Yang et al presented a work with the objective of minimizing total tardiness with the constraints of no-wait, no-idle and due dates. They use a scatter search based memetic algorithm. A scatter search generates a starting set of solution vectors to guarantee a critical level of diversity and applies heuristic processes designed for the problem considered as an attempt to improve these solutions. The assignment rules used are the shortest processing time (SPT), the earliest due date (EDD), the first-in-first-out (FIFO) and Naderi & Ruiz's assignment rules. The model is tested using experimental studies and comparisons (Yang, Peng, Wang, Liu, & Yongliang, 2017). To the best of my knowledge, this paper is the only one that approaches DAPFSP with a tardiness objective.

At CEC 2017, Li et al also presented a paper about a variation of DAPFSP: the Distributed Multiple Assembly Permutation Flowshop Scheduling Problem (DMAPFSP). As the name indicates, there are multiple assembly factories in this model (the conventional is only one). A no-wait constraint on the processing stage is also considered. The authors proposed a hybrid iterated local search with simulated annealing to solve this model and proved its efficiency through simulations on large-scale benchmark problems (P. Li et al., 2017).

In 2019 Pan et all released a paper on effective constructive heuristics and metaheuristics for the DAPFSP with the objective to minimize the makespan. A total of seven algorithms are used, including a Mixed Integer Linear Programming (MILP) model, three constructive heuristics, two Variant Neighbourhood Descent (VND) algorithms, and an iterated greedy (IG) algorithm for solving the DAPFSP and the results are compared with 16 existing methods and based on 810 benchmark instances (Q. K. Pan, Gao, Xin-Yu, & Framinan, 2019).

In 2019, a study was released by Sang et al, for the total flowtime criterion on the DAPFSP using an effective invasive weed optimization algorithm. A two-level representation that consists of a product permutation and a number of job sequences is put in place, then the effective local search procedures are designed for each one specifically (Sang et al., 2019).

c) Instances and distributions

(Sara Hatami et al., 2013) used a total of 900 small instances and 810 large instances with a total of $\{8,12,16,20,24\}$ jobs for small instances and $\{100,200,500\}$ for large instances, $\{2,3,4,5\}$ machines for all instances, $\{2,3,4\}$ flowshop for small instances and $\{4,6,8\}$ for large instances and $\{2,3,4\}$ products for small instances and $\{30,40,50\}$ for large instances. Processing times were uniformly distributed over the range $(1,99)$. The same benchmark is used by (X. Li et al., 2015), (J. Lin & Zhang, 2016).

(S. Hatami et al., 2014) used the same benchmark for small instances as the one used by (Sara Hatami et al., 2013) as for the large instances, they used a total of 720 large instances with $\{100,200\}$ jobs, $\{5,10,20\}$ machines, $\{4,6,8\}$ flowshop and $\{30,40,50\}$ products. The processing time distribution was also the same used by (Sara Hatami et al., 2013).

(S. Hatami, R. Ruiz, & C. Andrés-Romano, 2015) used only the large instances used by (Sara Hatami et al., 2013), with the same number of jobs, machines, flow shops and products. Processing times are also uniformly distributed over the range $(1,99)$. The same benchmark was used by (Pan et al, 2019) and (Sang et al, 2019).

(B. Liu et al., 2016) used the benchmark by (Pinedo & Chao, 1999) with 10 small instances and 10 large instances, with $\{8,12,16,24\}$ jobs for small instances and 100 for large instances, $\{2,3,4,5\}$ machines for small instances and $\{5,10,20\}$ for large instances, $\{2,3\}$ flowshop for small instances and $\{30,40,50\}$ for large, $\{2,3,4\}$ products for small instances and $\{4,6,8\}$ for large instances.

(Ji et al., 2016) use a simple benchmark with 12 jobs, $\{2,3,4,5\}$ machines, $\{2,3,4\}$ flowshops and $\{2,3,4\}$ products. Processing times were uniformly distributed between $U((1-n)P_{i,j}, (1+n)P_{i,j})$.

(Gonzalez-Neira et al., 2017) use a benchmark with 162 instances, $\{100,200,500\}$ jobs, $\{5,10,20\}$ machines and $\{30,40,50\}$ flowshops.

(Yang et al., 2017) use a benchmark with 10 small instances and 5 large instances, $\{12,24\}$ jobs for small instances, 100 for large instances, 5 machines for both instances, $\{2,3\}$ flowshops for small instances and 4 for large instances, $\{2,4\}$ products for small

instances and 30 for large. The due dates are distributed between $[CP(1 - F - R / 2), CP(1 - F + R / 2)]$.

(P. Li et al., 2017) use a benchmark of 27 large-sized instances with 100 jobs, $\{4,8,12\}$ machines, $\{5,10,20\}$ flowshops and $\{30,40,50\}$ products.

DPFSP with Weighted Tardiness

1. Problem definition and notation

A set $\mathrm{N} = \{1,2,\dots,n\}$ of n independent jobs have to be processed on a set $\mathrm{G} = \{1,2,\dots,\mathrm{F}\}$ of F factories. Each factory has the same set $\mathrm{M} = \{1,2,\dots,m\}$ of m machines, just like in a regular PFS. All jobs follow the same route through the machines, and the processing order of the jobs is the same for all machines (permutation flowshop).

All machines in all factories are continuously available from time zero, and preemptions are not allowed. All factories can process all jobs, and a job must be assigned to a single factory. So, once a job is assigned to a factory, it must be fully processed there, and cannot be transferred to another factory.

The factories are identical, so the processing time of a job on a machine is the same on any factory $f, f \in G$. Job $j, j \in N$, requires a processing time p_{ij} on machine $i, i \in M$. Each job $j, j \in N$ also has a weight w_j and a due date d_j.

Let $C_{f,i,j}$ denote the completion time of job $j, j \in N$ on machine $i, i \in M$, when job j has been assigned to factory $f, f \in G$. Furthermore, let the job sequenced in position j on factory f be denoted by $[j^f]$.

Since all machines are available at time zero, we have $C_{f,1,[0^f]} = 0$, for all $f \in G$. Then, $C_{f,1,[j^f]} = C_{f,1,[j-1^f]} + p_{1[[j^f]]}$ and $C_{f,i,[j^f]} = max\left\{C_{f,i-1,[j^f]}, C_{f,i,[j-1^f]}\right\} + p_{i[j^f]}$, for $i = \{2,3,\dots,m\}$ and for all $f \in G$.

Let j^a denote the factory to which job j was assigned. Finally, for convenience, let the completion time of job j, that is, the time at which job j finishes processing on the last machine, on the factory to which it was assigned, also be denoted by C_j, so $C_j = C_{j^a,m,j}$.

Given a schedule, the tardiness of job j is defined as $T_j = max\{C_j - d_j; 0\}$. The objective is then to find a schedule that minimizes the sum of the weighted tardiness values $\sum_{j=1}^{n} w_j T_j$.

Let U denote the set of jobs that have not yet been scheduled. Let S^f be the current partial schedule on factory $f, f \in G$, that is, the sequence of jobs that are scheduled so far on factory f. The completion time of job $j \in U$, if j is scheduled at the end of sequence S^f, is denoted by $C_j(S^f)$. So, $C_j(S^f)$ is the completion time of job j if it is assigned to factory f and scheduled after the current partial schedule on that factory. Also, let $s_j(S^f)$ be the slack of job $j \in U$ if j is scheduled at the end of S^f, where $s_j(S^f) = d_j - C_j(S^f)$.

The current availability time of machine i, in factory f, under schedule S^f, will be represented by $t_{f,i}(S^f)$. For convenience, the current availability time on the first machine in factory f will also be denoted by t_f so $t_f = t_{f,1}(S^f)$

The current makespan in factory f, under schedule S^f, will be represented by $C_f^{max}(S^f)$. So, we have $C_f^{max}(S^f) = t_{f,m}(S^f)$, since the makespan is equal to the completion time (or availability time) on the last machine.

The makespan in factory f, if job $j \in U$ is scheduled at the end of S^f, is denoted by $C_f^{max}(S^f, j)$. So, this is the updated makespan in factory f, after job j is appended to the current partial sequence.

Let $P_j(S^f) = C_j(S^f) - t_f$ be the total time (total processing time plus any eventual forced idle time) between the start and finish of job $j \in U$ if j is scheduled at the end of sequence S^f.

As previously mentioned, j^a denotes the factory to which job j is assigned. The average, over all jobs $j \in U$, of the $P_j(S^{j^a})$ values will be denoted by $\bar{P}(U)$. So, $\bar{P}(U)$ is the average of the total time between the start and finish of the unscheduled jobs, when they are scheduled on their assigned factory.

Finally, let $T_j(S^f) = max\{C_j(S^f) - d_j; 0\}$ be the tardiness of job $j \in U$ if j is scheduled at the end of sequence S^f.

2. Heuristics

a) General rules

This set of general rules is rather simple but, due to their simplicity, they are widely popular and results have been compared in different problems with multiple production settings.

The EDD (earliest due date) rule was created by Jackson,1955 and has been used on multiple scheduling problems involving due dates. The EDD rule is applied by selecting, in each of the model's iterations, the job with the largest value of the priority index, i.e., the earliest due date available $EDD_j(S^{j^a}) = -d_j$, therefore scheduling said jobs in non-decreasing order of their due dates. (Jackson, 1955)

In 2008, Ruiz and Stützle expanded upon the concept of the EDD rule by taking into consideration the weight of each job within the production total, therefore the EWDD (earliest weighted due date) rule instead of scheduling the jobs according to the due dates it schedules them according to their weighted due dates $EWDD_j(S^{j^a}) = w_j/d_j$ (R. Ruiz & Stützle, 2008).

The EDD rule although simple was modified in 1982 by Baker and Bertrand and in 1987 by Vepsalainen and Morton, creating the MDD (modified due date). This rule consists of adding an extra condidtion to the EDD,as at each iteration we select the job with the minimum value of the modified due date $max\{d_j, C_j(S^{j^a})\} = max\{d_j, t + P_j(S^{j^a})\} = max\{d_j - t, P_j(S^{j^a})\}$. Alternatively, this rule selects, at each iteration, the job with the largest value of the priority index $MDD_j(S)$:

$$MDD_j(S^{j^a}) = \begin{cases} 1/C_j(S^{j^a}) & if\ C_j(S^{j^a}) \geq d_j \\ 1/d_j & otherwise \end{cases}$$. (Baker & Bertrand, 1982)

The SLK rule (minimum slack) is, similarly to the EDD rule, one of the first rules created and rather simple to understand. Unlike the EDD rule, the SLK rules schedules the jobs by selecting at each iteration the jobs with the minimum slack, i.e., the job with the largest value

of the priority index $SLK_j(S^{j^a}) = \begin{cases} C_j(S^{j^a}) - d_j & if\ s_j(S^{j^a}) \leq -1 \\ 1/(s_j(S^{j^a}) + 2) & otherwise \end{cases}$. (Panwalker & Iskander, 1977).

Vepsalainen and Morton 1987 also created the SLK/P rule, which takes into account, not only, the minimum slack for each iteration but also the total required time for each job, therefore, selecting the jobs with the minimum value of the ratio between the slack and the total required time at each iteration $SLK/P_j(S^{j^a}) = -(s_j(S)/P_j(S^{j^a}))$ (Vepsalainen & Morton, 1987). An extended version of the SLK/P heuristic that takes into account the weight of each job will, instead, be used in this paper $WSLK/P_j(S^{j^a}) = \begin{cases} (w_j/P_j(S^{j^a})) * (C_j(S^{j^a}) - d_j) & if\ s_j(S^{j^a}) \leq -1 \\ (w_j/P_j(S^{j^a})) * [1/(s_j(S^{j^a}) + 2)] & otherwise \end{cases}$.

b) Rules for the Weighted Tardiness

The rules to be reviewed in this section are more advanced and are mostly applied to problems with a linear weighted tardiness objective function and most of them were developed for solving single machine environment problems but can be applied to other production settings as well.

The WSPT (weighted shortest processing time) was created in 1956 by Smith and schedules the jobs according to the weighted shortest processing time $WSPT_j(S^{j^a}) = w_j/P_j(S^{j^a})$, i.e., in each iteration of the model the job with the weighted shortest processing time is selected.

The weighted minimum slack/shortest processing time (WSLK_SPT) Osman et al. 2009 combines the minimum slack with the shortest processing time. In each iteration of the model, the job with the minimum ratio $max\{s_j(S^{j^a}), P_j(S^{j^a})\}/w_j$ is selected between the weighted slack or the weighted processing time $WSLK\_SPT_j(S^{j^a}) = \begin{cases} w_j/P_j(S^{j^a}) & if\ s_j(S^{j^a}) \le P_j(S^{j^a}) \\ w_j/s_j(S^{j^a}) & otherwise \end{cases}$ (Ibrahim H. Osman, Belouadah, Fleszar, & Saffa, 2009)

An alternate version of the previously mentioned MDD that takes into account the weight of each job can also be used,

$WMDD = \begin{cases} w_j/P_j(S^{j^a}) & if\ s_j(S^{j^a}) \le 0 \\ w_j/(d_j - t_{j^a}) & otherwise \end{cases}$ (Baker & Bertrand, 1982).

The AR (Alidaee & Ramakrishnan, 1996) and Apparent Tardiness Cost (ATC) (Vepsalainen & Morton, 1987) are known for providing the best performance among all the heuristics created for the single machine linear problem (Alidaee & Ramakrishnan, 1996)

$$AR_j\ (v1)(S^{j^a}) =$$

$$\begin{cases} w_j/P_j(S^{j^a}) & if\ s_j(S^{j^a}) \le 0 \\ (w_j/P_j(S^{j^a})) * [k\bar{P}(S^{j^a})/(k\bar{P}(S^{j^a}) + s_j(S^{j^a}))] & otherwise \end{cases}$$

$$AR_j\ (v2)(S^{j^a}) =$$

$$\begin{cases} w_j/P_j(S^{j^a}) & if\ s_j(S^{j^a}) \le 0 \\ (w_j/P_j(S^{j^a})) * [k(\bar{P}(U)/F)/(k(\bar{P}(U)/F) + s_j(S^{j^a}))] & otherwise \end{cases}.$$

and

$$ATC_j\ (V1)\left(S^{j^a}\right) = \begin{cases} w_j/P_j\left(S^{j^a}\right) & if\ s_j\left(S^{j^a}\right) \le 0 \\ \left(w_j/P_j\left(S^{j^a}\right)\right) * \exp\left(-s_j\left(S^{j^a}\right)/k\bar{P}\left(S^{j^a}\right)\right) & otherwise \end{cases}.$$

$$ATC_j\ (V2)\left(S^{j^a}\right) = \begin{cases} w_j/P_j\left(S^{j^a}\right) & if\ s_j\left(S^{j^a}\right) \le 0 \\ \left(w_j/P_j\left(S^{j^a}\right)\right) * \exp\left(-s_j\left(S^{j^a}\right)/k(\bar{P}(U)/F)\right) & otherwise \end{cases}$$

Both the AR and ATC rules require a lookahead parameter k. This parameter should reflect the number of competing critical jobs (that is, jobs that are not yet tardy, but are about to become so).

To determine the lookahead parameter k, I would propose using a strategy that has been used before. That is, we will consider a job to be critical if its slack is positive, but less than or equal to a value slk_thr, which stands for *slack threshold*. Thus, a job is considered critical if $0 < s_j\left(S^{j^a}\right) \le slk\_thr$.

At each iteration, k is then set equal to the number of critical jobs. If, at a given iteration, no job is critical according to our criterion, k is then set equal to 0.5, since this is the lowest value that has been usually considered for this parameter.

Now for the calculation of the slack threshold. The slack threshold is meant to represent a value such that slacks which are greater are considered large, so the job is not close to becoming tardy, and is therefore not critical. As such, and similarly to what has been done before, I propose calculating the slk_thr parameter as follows.

Let v, with $0 \le v \le 1$,, be a user–defined parameter. Also let $C_{LB}^{max}(U)$ be a lower bound on the makespan of the remaining unscheduled jobs U, if they were the only jobs to be scheduled in the distributed permutation flowshop. That is, $C_{LB}^{max}(U)$ is a lower bound for the makespan of the remaining unscheduled jobs U, if they were the only ones to schedule in our distributed permutation flowshop, that is, if the already scheduled jobs were not considered at all. So, $C_{LB}^{max}(U)$ is essentially a lower bound on the time needed to process the remaining unscheduled jobs in a distributed permutation flowshop. At each iteration, the slack threshold would be set equal to $slk_{thr} = v \times C_{LB}^{max}(U)$.

Now for how to calculate $C_{LB}^{max}(U)$. Let TLB(U) be the Taillard lower bound on the makespan of a permutation flowshop, if the remaining unscheduled jobs U were the only jobs to be scheduled in that flowshop. I would suggest setting $C_{LB}^{max}(U) = TLB(U)/F$. Essentially, we calculate the lower bound on the makespan of the unscheduled jobs in a

permutation flowshop. Then, to convert that lower bound into a distributed permutation flowshop, we divide by the number of factories (similarly to what was done to generate the due dates for the instances). So, we would have $slk_{thr} = v \times (TLB(U)/F)$.

Both for ATC and AR, two versions are tested. These versions differ in whether or not a part of the priority index, namely $k\bar{P}$, is adjusted for the number of factories.

In version v1, there is no adjustment. That is, we use $k\bar{P}$ as is usual, where k is the lookahead parameter, and $\bar{P}$ is the average total processing time among the unscheduled jobs.

In version v2, there is an adjustment, and the priority index is different: $k\bar{P}$ is adjusted by the number of factories. That is, the priority index uses $k\bar{P}/F$ (instead of the usual $k\bar{P}$), where k is the lookahead parameter, $\bar{P}$ is the average total processing time among the unscheduled jobs, and F is the number of factories. Therefore, $k\bar{P}$ is divided by F, in order to take the number of factories into account.

The next two heuristics are, to the best of my knowledge, new.

WSPT_EDD (weighted shortest processing time / earliest due date)

$$\begin{cases} w_j/P_j(S^{j^a}) & if\ s_j(S^{j^a}) \le 0 \\ w_j/P_j(S^{j^a}) - \left[\left(s_j(S^{j^a}) \times \left(d_j + w_j/P_j(S^{j^a})\right)\right)/slk_{large}\right] & if\ 0 < s_j(S^{j^a}) < slk_{large} \\ -d_j & if\ s_j(S^{j^a}) \ge slk_{large} \end{cases}$$

.

WSPT_WEDD (weighted shortest processing time / weighted earliest due date)

$$\begin{cases} w_j/P_j(S^{j^a}) & if\ s_j(S^{j^a}) \le 0 \\ w_j/P_j(S^{j^a}) - \left[\left(s_j(S^{j^a}) \times \left(d_j/w_j + w_j/P_j(S^{j^a})\right)\right)/slk_{large}\right] & if\ 0 < s_j(S^{j^a}) < slk_{large} \\ -d_j/w_j & if\ s_j(S^{j^a}) \ge slk_{large} \end{cases}$$

.

In the single machine problem, WSPT is good when a job is tardy, and EDD is good if the due dates are loose/spread out. So, WSPT is used when a job is tardy, EDD when a job is quite early, and a linear interpolation is performed in between. The second heuristic is similar, but the weight w_j is added.

These two heuristics require calculating the large slack value slk_{large}. I suggest calculating it in the exact same way as the slack threshold slk_thr. So, we would have $slk_{large} = v \times (TLB(U)/F)$.

3. Assignment rules

All the previously mentioned heuristics were tested under three different assignment rules.

The earliest start (ES) rule in which the job is assigned to the factory with the earliest available time, that is, to the factory with the smallest current time on the first machine (ties are broken by selecting the factory with the smallest index). So, the job is assigned to the factory with the smallest t_f. Here, at a given iteration, all jobs are assigned to the same factory.

The earliest completion makespan (ECM) where the job is assigned to the factory with the lowest current makespan, not including the job itself (again, break ties by choosing the factory with the smallest index). So, the job is assigned to the factory with the smallest $C_f^{max}(S^f)$. Again, at a given iteration, all jobs are assigned to the same factory. This priority rule was previously detailed on the literature review and was created by (B. Naderi & Ruiz, 2010).

The earliest updated makespan (EUM) in which the job is assigned to the factory with the lowest makespan after the job is added at the end of the current partial sequence. So, each job is tried on each factory, and it is assigned (and its priority is calculated on) to the factory in which it finishes earliest (again, ties are broken by choosing the factory with the smallest index). Here, different jobs may be assigned to different factories. A job is assigned to the factory with the smallest $C_f^{max}(S^f, j)$. This priority rule was previously detailed in the literature review and was created by (B. Naderi & Ruiz, 2010).

4. Instances and distributions

In order to test this problem two test beds were used: a *test set* of medium and large instances with 10 instances per parameter combination used for parameter adjustment and one *full set* of the same instances but with 100 instances per parameter combination used for heuristic comparisons.

The possible number of factories $F = (2, 3, 4, 5, 6, 7)$, machines $m = (5, 10, 20)$ and jobs $n = (25, 50, 75, 100, 300, \ 500)$ is the same used by some of the most important works on the DPFS literature like (B. Naderi & Ruiz, 2010).

The processing times are uniformly distributed between $[1,100]$ and weights from $[1,10]$. Both these distributions are based on the works of (B. Naderi & Ruiz, 2010) and (Vallada & Ruiz, 2010).

The due dates distributions are a combination between the (Vallada & Ruiz, 2010) proposal for the Permutation Flowshop Problem $[P(1 - T - R/2); P(1 - T + R/2)]$. P is the lower bound of the makespan, T is the tardiness factor and R is the due date range factor but replace P for the Tailard Lower Bound on the makespan for a PFS (TLB) divided by the number of factories. Since we're now dealing with the DPFSP instead of the PFS, therefore the lower bound makespan must be equally divided by all the factories, which gives us $[(TLB/F) * (1 - T - R/2); (TLB/F) * (1 - T + R/2)]$. The possible values for the tardiness factor are $T = (0.2, 0.4, 0.6, 0.8, 1)$ and for the due date range factor are $R = (0.2, 0.4, 0.6, 0.8, 1)$. These ranges are the same ones often used across the literature of the PFSF, like the previously mentioned (Vallada & Ruiz, 2010) .

5. Computational results

In the parameter tests, the heuristics, for each assignment rule, were applied to all instances in the test set (the test set is described in the previous section). For the AR and ATC heuristics, for both v1 and v2 versions, and for the three assignment rules, the best setting was $v = 0.0$. This means that k is always equal to 0.5. For the other two heuristics (WSPT_EDD and WSPT_WEDD), the best setting was $v = 0.05$.

After defining the parameters, all the heuristics were applied to the full set. The results were used to compare the heuristics, considering both effectiveness (solution quality) and efficiency (computational time, measured in seconds).

Whenever a certain set of heuristics were compared, the relative improvement vs worst (ivw) was calculated. To find this measure, the worst result among the heuristics was computed for each instance.

For effectiveness, if the objective function value of a heuristic is equal to the worst (for a certain instance), the ivw for that heuristic, on that instance, is 0. Otherwise, it is equal to $(ofv_{worst} - ofv_{heur})/ofv_{worst} \times 100$. The mean relative improvement is then calculated over several instances.

The results for mean improvement versus the worst result of ofv, for each assignment rule, for instances with $F = 4$ are presented on Table A1 (ES), Table A3 (ECM) and Table A5 (EUM). Results are similar for other values of F. Considering effectiveness, the results are consistent for all assignment rules, with AR_v2 and ATC_v1 (designed specifically for weighted tardiness) obtaining the best results. On the other hand, SLK (simpler heuristic) has the worst results.

The results for mean improvement versus the worst result of ofv, for each assignment rule, for instances with $F = 4, m = 20$ and $n = 300$ are presented on Table A2 (ES), Table A4 (ECM) and Table A6 (EUM). In these tables, the effect of T and R are analysed. Results are similar for other values of n, m and F. Considering effectiveness, the results are consistent for all assignment rules, with AR_v2 obtaining the best results. SLK has the worst results. Analysing the effect of T and R, it can be concluded that ivw decreases with T, but

R does not always have a consistent effect on ivw. So, the difference in performance versus the worst result decreases as the tardiness factor increases.

Nevertheless, higher complexity also usually means more processing time. For efficiency, runtime's average is calculated, in seconds. The results for average of runtime, in seconds, for each assignment rule, for instances with $F = 4$ are presented on Table A7 (ES), Table A8 (ECM) and Table A9 (EUM). Results are similar for other values of F. In general, EDD and WEDD are the fastest heuristics. Then, we have a second group of heuristics with similar runtimes, including: WSPT, MDD, SLK, WSLK/P and WSLK_SPT. ATC and AR are usually the ones that require more time. The WSPT_EDD and WSPT_WEDD usually are in the second or third group (depending on the number of machines).

It is noteworthy that even the slowest heuristics are quite fast: for $F = 4$ the slowest heuristics take about 0.06 seconds to generate a solution and 0.07 seconds for $F = 7$. So, we indeed have fast heuristics, like we intended to develop.

Regarding the assignment rules, for each heuristic EUM provided better solutions, but required more time than ES and ECM. On the other hand, ES and ECM provided similar results among them, with ES being marginally better than ECM. This outcome is similar for every heuristic. This result is natural, since EUM has to consider an assignment to all factories, while ES and ECM only consider an assignment to one factory. The ratio of the runtime of an EUM heuristic to the corresponding ES or ECM heuristic is close to the number of factories, as one would expect. This means that, when $F = 4$, the runtime of an EUM version is somewhat equal to 4 times that of a corresponding ECM or ES versions. It is lower than that, since not all work is done for each factory in EUM, but it still naturally increases with F. In the results for $F = 7$, EUM (average of 0.0738 seconds) takes about 6 times as much runtime as the ES (average of 0.0108 seconds) and ECM (average of 0.0123 seconds) versions.

The results on Table A10 (results for mean improvement versus worst result of ofv and average of runtime, in seconds, for instances with $F = 4$) and Table A11 (results for mean improvement versus worst result of ofv, for instances with $F = 4, m = 20$ and $n = 300$) show the comparison between the heuristics that were described above as more

effective (AR_v2 and ATC_v1) for every assignment rule. It can be concluded that EUM is the most effective, but the least efficient. The most effective combination of assignment rule/heuristic is EUM_AR_v2. For this combination, it can be observed that ivw decreases with T, but R does not always have a consistent effect on ivw. Again, the difference in performance versus the worst result decreases as the tardiness factor increases.

It is noteworthy to mention that the heuristics created for this paper, the WSPT_EDD and WSPT_WEDD fall something in the middle on the trade-off between efficiency and processing time, not dethroning ATC and AR in terms of effectiveness but outperforming them in terms of efficiency, while not being the best on it either.

Conclusion

In this work, we study the DPFSP the criteria of minimizing the weighted tardiness. To the best of my knowledge, this is the first work that approaches this objective on DPFSP.

After reviewing the existing literature on the topic, the problem is defined with its notation. Then the heuristics to test are selected: five general rules, seven rules specific for weighted tardiness and two new rules developed for this problem. These heuristics are tested with three assignment rules.

All combinations of heuristics and assignment rules are tested, and the results are compared among themselves in effectiveness (solution quality) and efficiency (computational time. AR_v2 with EUM rule is the most effective combination heuristic/assignment rule. In general, assignment rule EUM is more effective but requires more computational time.

This work may open a new investigation on the future upon the weighted tardiness minimization objective on the DPFSP with the development of metaheuristics to enhance the effectiveness and/or efficiency. This work can also be developed into a multi-objective view. The weighted tardiness can be applied to the DAPFSP, as well.

References

Alidaee, B., & Ramakrishnan, K. R. (1996). A computational experiment of covert-AU class of rules for single machine tardiness scheduling problem. *Computers and Industrial Engineering, 30*(2), 201-209. doi:10.1016/0360-8352(95)00166-2

Baker, K. R., & Bertrand, J. W. M. (1982). A dynamic priority rule for scheduling against due-dates. *Journal of Operations Management, 3*(1), 37-42. doi:10.1016/0272-6963(82)90020-1

Bargaoui, H., Belkahla Driss, O., & Ghédira, K. (2017). A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion. *Computers & Industrial Engineering, 111*(Supplement C), 239-250. doi:https://doi.org/10.1016/j.cie.2017.07.020

Cai, S., Yang, K., & Liu, K. (2018). Multi-objective Optimization of the Distributed Permutation Flow Shop Scheduling Problem with Transportation and Eligibility Constraints. *Journal of the Operations Research Society of China, 6*(3), 391-416. doi:10.1007/s40305-017-0165-3

Cura, T. (2015). An evolutionary algorithm for the permutation flowshop scheduling problem with total tardiness criterion. *International Journal of Operational Research, 22*(3), 366-384. doi:10.1504/IJOR.2015.068287

Deng, J., & Wang, L. (2017). A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem. *Swarm and Evolutionary Computation, 32*(Supplement C), 121-131. doi:https://doi.org/10.1016/j.swevo.2016.06.002

Deng, J., Wang, L., & Wang, S. (2014). *A competitive memetic algorithm for the distributed flow shop scheduling problem*.

Du, X., Ji, M., Li, Z., & Liu, B. (2017). *Scheduling of stochastic distributed assembly flowshop under complex constraints*. Paper presented at the 2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016.

Duan, W., Li, Z., Ji, M., Yang, Y., Wang, S., & Liu, B. (2016, 24-29 July 2016). *A hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling with flowline eligibility*. Paper presented at the 2016 IEEE Congress on Evolutionary Computation (CEC).

Fernandez-Viagas, V., & Framinan, J. M. (2015). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research, 53*(4), 1111-1123. doi:10.1080/00207543.2014.948578

Fernandez-Viagas, V., & Framinan, J. M. (2015). NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Computers & Operations Research, 60*(Supplement C), 27-36. doi:https://doi.org/10.1016/j.cor.2015.02.002

Fernandez-Viagas, V., Perez-Gonzalez, P., & Framinan, J. M. (2018). The distributed permutation flow shop to minimise the total flowtime. *Computers and Industrial Engineering, 118*, 464-477. doi:10.1016/j.cie.2018.03.014

Gao, J., & Chen, R. (2011a). A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Computational Intelligence Systems, 4*(4), 497-508. doi:10.1080/18756891.2011.9727808

Gao, J., & Chen, R. (2011b). An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems. *Scientific Research and Essays, 6*(14), 3094-3100.

Gao, J., Chen, R., & Deng, W. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research, 51*(3), 641-651. doi:10.1080/00207543.2011.644819

Gonzalez-Neira, E. M., Ferone, D., Hatami, S., & Juan, A. A. (2017). A biased-randomized simheuristic for the distributed assembly permutation flowshop problem with stochastic processing times. *Simulation Modelling Practice and Theory, 79*(Supplement C), 23-36. doi:https://doi.org/10.1016/j.simpat.2017.09.001

Hatami, S., Ruiz, R., & Andrés-Romano, C. (2013). The Distributed Assembly Permutation Flowshop Scheduling Problem. *International Journal of Production Research, 51*(17), 5292-5308. doi:10.1080/00207543.2013.807955

Hatami, S., Ruiz, R., & Andrés-Romano, C. (2014, 3-5 Nov. 2014). *Simple constructive heuristics for the Distributed Assembly Permutation Flowshop Scheduling Problem with sequence dependent setup times.* Paper presented at the 2014 International Conference on Control, Decision and Information Technologies (CoDIT).

Hatami, S., Ruiz, R., & Andrés-Romano, C. (2015). Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. *International Journal of Production Economics, 169*(Supplement C), 76-88. doi:https://doi.org/10.1016/j.ijpe.2015.07.027

Hatami, S., Ruiz, R., & Andrés-Romano, C. (2015, 21-23 Oct. 2015). *Heuristics for a Distributed Parallel Machine Assembly Scheduling Problem with eligibility constraints.* Paper presented at the 2015 International Conference on Industrial Engineering and Systems Management (IESM).

Jackson, J. R. (1955). *Scheduling a production line to minimize maximum tardiness* (Vol. 1).

Ji, M., Yang, Y., Duan, W., Wang, S., & Liu, B. (2016, 24-29 July 2016). *Scheduling of no-wait stochastic distributed assembly flowshop by hybrid PSO.* Paper presented at the 2016 IEEE Congress on Evolutionary Computation (CEC).

Jiang, E., Wang, L., & Lu, J. (2018). *Modified multiobjective evolutionary algorithm based on decomposition for low-carbon scheduling of distributed permutation flow-shop.*

Karabulut, K. (2016). A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops. *Computers and Industrial Engineering, 98*, 300-307. doi:10.1016/j.cie.2016.06.012

Li, P., Yang, Y., Du, X., Qu, X., Wang, K., & Liu, B. (2017, 5-8 June 2017). *Iterated local search for distributed multiple assembly no-wait flowshop scheduling.* Paper presented at the 2017 IEEE Congress on Evolutionary Computation (CEC).

Li, X., Zhang, X., Yin, M., & Wang, J. (2015, 25-28 May 2015). *A genetic algorithm for the distributed assembly permutation flowshop scheduling problem.* Paper presented at the 2015 IEEE Congress on Evolutionary Computation (CEC).

Li, Z., Duan, W., Ji, M., Yang, Y., Wang, S., & Liu, B. (2016, 24-29 July 2016). *The distributed permutation flowshop scheduling problem with different transport timetables and loading capacities.* Paper presented at the 2016 IEEE Congress on Evolutionary Computation (CEC).

Lin, J., & Zhang, S. (2016). An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem. *Computers and Industrial Engineering, 97*, 128-136. doi:10.1016/j.cie.2016.05.005

Lin, S.-W., Ying, K.-C., & Huang, C.-Y. (2013). Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *International Journal of Production Research, 51*(16), 5029-5038. doi:10.1080/00207543.2013.790571

Ling-Fang, C., Ling, W., & Jing-Jing, W. (2018). *A Two-Stage Memetic Algorithm for Distributed No-Idle Permutation Flowshop Scheduling Problem.* Paper presented at the 37th Chinese Control Conference, CCC 2018.

Liu, B., Wang, K., & Zhang, R. (2016, 24-29 July 2016). *Variable neighborhood based memetic algorithm for distributed assembly permutation flowshop.* Paper presented at the 2016 IEEE Congress on Evolutionary Computation (CEC).

Liu, H., & Gao, L. (2010). *A discrete electromagnetism-like mechanism algorithm for solving distributed permutation flowshop scheduling problem*.

Molina-Sánchez, L. P., & González-Neira, E. M. (2016). GRASP to minimize total weighted tardiness in a permutation flow shop environment. *International Journal of Industrial Engineering Computations, 7*(1), 161-176. doi:10.5267/j.ijiec.2015.6.004

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research, 37*(4), 754-768. doi:https://doi.org/10.1016/j.cor.2009.06.019

Naderi, B., & Ruiz, R. (2014). A scatter search algorithm for the distributed permutation flowshop scheduling problem. *European Journal of Operational Research, 239*(2), 323-334. doi:https://doi.org/10.1016/j.ejor.2014.05.024

Osman, I. H., Belouadah, H., Fleszar, K., & Saffa, M. (2009). *Hybrid of the weighted minimum slack and shortest processing time dispatching rules for the total weighted tardiness single machine scheduling problem with availability constraints*. Paper presented at the Mulidisciplinary International Conference on Scheduling Dublin.

Osman, I. H., & Kelly, J. P. (1996). *Meta-Heuristics: An Overview*: Springer, Boston, MA.

Pan, J. Q., Zou, W. Q., & Duan, J. H. (2018). *A discrete artificial bee colony for distributed permutation flowshop scheduling problem with total flow time minimization.* Paper presented at the 37th Chinese Control Conference, CCC 2018.

Pan, Q. K., Gao, L., Wang, L., Liang, J., & Li, X. Y. (2019). Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Systems with Applications, 124*, 309-324. doi:10.1016/j.eswa.2019.01.062

Pan, Q. K., Gao, L., Xin-Yu, L., & Framinan, J. M. (2019). Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem. *Applied Soft Computing Journal, 81*. doi:10.1016/j.asoc.2019.105492

Panwalker, & Iskander. (1977). A SURVEY OF SCHEDULING RULES *Journal of Operations Reserarch, 25*(1), 45-61.

Pinedo, M., & Chao, X. (1999). *Operations Scheduling with applications in manufacturing and services*: McGraw Hill International Editions.

Raman, N. (1995). Minimum tardiness scheduling in flow shops: Construction and evaluation of alternative solution approaches. *Journal of Operations Management, 12*(2), 131-151. doi:https://doi.org/10.1016/0272-6963(94)00010-C

Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research, 165*(2), 479-494. doi:https://doi.org/10.1016/j.ejor.2004.04.017

Ruiz, R., Pan, Q. K., & Naderi, B. (2019). Iterated Greedy methods for the distributed permutation flowshop scheduling problem. *Omega (United Kingdom), 83*, 213-222. doi:10.1016/j.omega.2018.03.004

Ruiz, R., & Stützle, T. (2008). An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research, 187*(3), 1143-1159. doi:10.1016/j.ejor.2006.07.029

Sang, H. Y., Pan, Q. K., Li, J. Q., Wang, P., Han, Y. Y., Gao, K. Z., & Duan, P. (2019). Effective invasive weed optimization algorithms for distributed assembly permutation flowshop problem with total flowtime criterion. *Swarm and Evolutionary Computation, 44*, 64-73. doi:10.1016/j.swevo.2018.12.001

Vallada, E., & Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega, 38*(1), 57-67. doi:https://doi.org/10.1016/j.omega.2009.04.002

Vallada, E., Ruiz, R., & Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research, 35*(4), 1350-1373. doi:https://doi.org/10.1016/j.cor.2006.08.016

Vepsalainen, A. P. J., & Morton, T. E. (1987). PRIORITY RULES FOR JOB SHOPS WITH WEIGHTED TARDINESS COSTS. *Management Science, 33*(8), 1035-1047. doi:10.1287/mnsc.33.8.1035

Wang, J., Wang, L., & Shen, J. (2016, 24-29 July 2016). *A hybrid discrete cuckoo search for distributed permutation flowshop scheduling problem.* Paper presented at the 2016 IEEE Congress on Evolutionary Computation (CEC).

Wang, S. Y., Wang, L., Liu, M., & Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics, 145*(1), 387-396. doi:10.1016/j.ijpe.2013.05.004

Xu, Y., Wang, L., Wang, S., & Liu, M. (2014). An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem. *Engineering Optimization, 46*(9), 1269-1283. doi:10.1080/0305215X.2013.827673

Yang, Y., Peng, L., Wang, S., Liu, B., & Yongliang, L. (2017, 5-8 June 2017). *Scatter search for distributed assembly flowshop scheduling to minimize total tardiness.* Paper presented at the 2017 IEEE Congress on Evolutionary Computation (CEC).

Yenisey, M. M., & Yagmahan, B. (2014). Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega, 45(*Supplement C), 119-135. doi:https://doi.org/10.1016/j.omega.2013.07.004

Ying, K. C., Lin, S. W., Cheng, C. Y., & He, C. D. (2017). Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems. *Computers and Industrial Engineering, 110*, 413-423. doi:10.1016/j.cie.2017.06.025

Table A1: Mean improvement versus the worst result of ofv, for the ES assignment rule, for instances with $F = 4$

m	n	AR_v1	AR_v2	ATC_v1	ATC_v2	EDD	MDD	SLK	WEDD	WMDD	WSLK_SPT	WSLKP	WSPT_EDD	WSPT	WSPT_WEDD
5	25	38.74476007	37.72922984	38.61105416	37.37798351	13.69151753	32.19618758	0.116203626	28.08410955	38.75912	38.19634404	24.90724406	36.42111206	38.22630194	36.445488
	50	45.14418671	44.53196503	45.22028366	43.48143739	12.95515613	33.99995828	0.167582339	32.4904792	44.61306979	41.46132477	27.91356234	41.67455999	41.04864866	42.63313643
	75	47.70523231	47.5599507	48.06343918	46.25588256	12.64097482	33.77886813	0.86096702	33.46624917	46.50791251	41.18531902	29.34677415	43.79528084	39.40370221	45.46238241
	100	49.63188605	49.77276728	50.26092235	48.16526373	13.15207068	33.72837747	2.079248707	34.1576658	47.98804951	41.20625826	31.18531898	45.53233068	37.83607097	47.59474928
	300	55.22480438	55.55471676	56.48417317	53.53347105	19.62963493	34.35617268	10.47433206	35.4509021	53.37239976	45.85047522	38.38244697	52.14539939	29.76830898	54.5795947
	500	56.82204047	57.07802284	58.21140361	55.48672421	22.62270269	34.08621823	14.80452861	36.36215826	55.87870594	49.62677466	41.10339642	54.58124775	26.91613609	57.30420506
10	25	27.22515574	27.16990086	27.22084178	27.15616602	9.387753059	23.52357216	0.046254057	18.94521695	27.22467977	27.23710431	19.7337733	27.07923636	27.23710431	27.07923636
	50	33.54747077	33.165202	33.55926301	33.09712056	8.459423927	24.52365589	0.043417632	23.48078729	33.60364885	32.99641828	23.05026618	32.74706812	33.0026295	32.75194267
	75	37.2761286	37.03822296	37.25379783	36.78164609	7.828849158	23.919329	0.046474719	26.70024624	36.98625177	35.22488465	25.39205189	35.93973481	35.11797633	36.28835418
	100	39.52184551	39.618888	39.63492023	39.265447	7.258388211	22.66163441	0.06894143	28.28572934	38.88456361	35.73154552	26.66260971	37.98598856	35.44548692	38.81504028
	300	47.66528871	49.37136374	48.88692722	47.91197651	10.12333804	21.01393029	4.307798367	31.97380251	44.77130808	36.57040986	32.85449108	45.60148579	31.69370433	48.03242373
	500	50.80064931	52.73364777	52.42574593	50.83500082	13.70352023	20.97218016	8.375646692	32.75000746	47.38064025	38.92948588	36.56989352	48.8297665	29.08562466	51.46479112
20	25	18.41798919	18.41748449	18.41798919	18.41748449	6.616045247	16.66982081	0.052307889	12.92800007	18.41798919	18.41798919	14.53538282	18.41748449	18.41798919	18.41748449
	50	22.9518159	22.92423304	22.95186431	22.92251557	5.895453135	17.37927673	0.018777862	15.42730738	22.9555869	22.95665332	17.18188316	22.90804599	22.95665332	22.90804599
	75	26.21937385	26.16859746	26.21085248	26.16682204	5.361960067	16.90316741	0.017119454	17.6113634	26.23439656	26.09553913	19.25371347	26.11312387	26.09685653	26.11312387
	100	28.83027386	28.70148686	28.82066288	28.68842789	4.799646666	16.07525323	0.025076209	19.82663852	28.80005345	28.26507359	21.17183636	28.55745739	28.27703998	28.56250022
	300	38.91572247	40.30644076	39.46675135	39.85972412	3.525784834	10.71118108	0.110469939	27.86845658	37.33329069	32.23481589	27.39983938	38.06413608	31.57224528	39.6007511
	500	42.96501648	45.38041202	44.10615851	44.36050809	5.590577564	10.28785329	2.573972407	29.77187018	40.10159031	32.55531578	30.73994511	42.07614457	30.08801627	44.39172719

Table A2: Mean improvement versus the worst result of ofv, for the ES assignment rule, for instances with $F = 4, m = 20$ and $n = 300$

T	R	AR_v1	AR_v2	ATC_v1	ATC_v2	EDD	MDD	SLK	WEDD	WMDD	WSLK_SPT	WSLKP	WSPT_EDD	WSPT	WSPT_WEDD
0.2	0.2	63.74892	64.12908	64.52127	55.47228	11.22878	27.27155	0.119261	57.58043	63.4494	61.34625	46.14033	45.32993	60.18065	63.46478562
	0.4	62.39697	64.22926	64.36823	56.95959	10.62961	29.3942	0.464164	47.42338	57.90766	48.63587	33.68519	49.69536	44.7012	60.93384118
	0.6	60.96774	67.82644	66.79174	62.85987	14.7148	36.01207	1.994858	31.90724	49.63221	27.83095	29.87189	56.45588	16.15166	62.02123358
	0.8	68.85911	81.43038	77.10611	79.27077	47.80258	65.42658	35.11778	34.49462	53.14096	24.92074	48.8328	73.12618	0	73.01436915
	1	76.4257	89.61099	84.92323	91.3694	76.94383	89.5677	69.22078	47.0297	61.27548	31.27967	71.00992	86.66213	0	84.96781779
0.4	0.2	55.99238	55.65315	56.5453	51.42159	7.807556	20.37052	0.035433	50.47725	55.70164	53.664	40.60804	44.24993	52.7854	55.29231623
	0.4	58.11516	56.57292	58.30412	52.07267	6.898849	20.6137	0.116689	48.07491	55.58271	48.17974	34.77442	48.38135	45.67931	55.8427619
	0.6	60.32495	60.39955	60.6378	57.25955	8.087245	21.11168	0	46.70278	55.72863	40.82223	34.26451	54.76681	36.02724	57.38261257
	0.8	61.16313	64.3573	63.08323	63.22066	8.596788	23.2709	0.038025	46.06474	54.47647	33.55017	31.4331	60.44118	23.893	61.25406153
	1	59.36353	65.41691	61.7243	65.39661	7.969239	24.68913	0.225295	47.06153	52.96895	30.53596	29.46124	63.96856	15.75643	63.96855628
0.6	0.2	48.05375	47.39589	47.76626	46.02067	5.695581	15.02128	0.005003	43.23548	47.84153	45.05687	37.25346	42.32818	44.91942	46.85448177
	0.4	49.3611	48.83162	49.04451	48.10034	5.339754	14.39718	0.00353	42.5142	48.63734	42.83726	35.30029	46.38481	41.92612	47.56690828
	0.6	48.99668	49.31242	49.06882	49.07002	4.643994	14.04869	0.018496	39.0339	48.20842	40.64678	33.10802	49.01109	38.07912	49.01108666
	0.8	48.50462	48.92131	48.61463	49.01065	4.056082	14.69923	0.037164	32.67534	47.25767	38.84525	31.62518	49.05786	34.90636	49.05786033
	1	46.21563	47.20617	46.58453	47.28008	4.140974	14.07206	0.092114	26.76607	44.82078	35.57074	28.57429	47.10187	30.95861	47.10187287
0.8	0.2	37.78058	37.60256	37.77991	37.56311	4.461144	12.5126	0	30.92301	37.84524	37.05665	32.14869	37.63909	37.06841	37.63909304
	0.4	37.62991	37.5892	37.67336	37.5705	3.495051	11.65402	0.029021	21.98239	37.63226	35.95908	30.1247	37.66385	35.79643	37.66384815
	0.6	36.72309	36.75205	36.7322	36.73561	3.003045	10.53311	0.051062	16.65299	36.80733	34.7515	27.97427	36.8678	34.18581	36.86780188
	0.8	35.77266	35.79554	35.71149	35.83116	2.620594	10.43405	0.013436	15.14599	35.60891	32.95424	25.87244	35.82447	31.73735	35.82447153
	1	35.18692	35.23515	35.15031	35.27666	2.642599	10.00707	0.029901	14.80237	34.80246	31.41741	23.9795	35.16105	29.96413	35.16105494
1	0.2	28.54557	28.52599	28.54339	28.52582	3.362524	9.251307	0	12.34906	28.55373	28.5689	25.97658	28.53575	28.5689	28.53575278
	0.4	28.21988	28.23131	28.18711	28.22615	2.672409	8.323209	0.007094	12.26496	28.29141	28.23544	24.47267	28.22834	28.26294	28.228342
	0.6	28.02397	28.03622	28.03928	28.03975	2.298892	7.915204	0.005841	11.62256	28.01442	27.87223	23.02694	27.99705	27.67262	27.99704528
	0.8	27.88647	27.85165	27.88956	27.85333	2.170974	7.781776	0.006571	11.57033	27.89989	27.32701	21.69176	27.79581	27.08156	27.79581098
	1	27.37379	27.37103	27.38249	27.39258	1.800564	6.969443	0.063442	10.98984	27.19721	26.39528	20.15204	27.36281	26.03995	27.36280704

Table A3: Mean improvement versus the worst result of ofv, for the ECM assignment rule, for instances with $F = 4$

m	n	AR_v1	AR_v2	ATC_v1	ATC_v2	EDD	MDD	SLK	WEDD	WMDD	WSLK_SPT	WSLKP	WSPT_EDD	WSPT	WSPT_WEDD
5	25	37.15019	36.53628	37.13543	36.21688	12.55336	29.74868	0.04771	27.35411	37.08952	36.41243	25.17477	35.30021	36.38606	35.33566
	50	43.60984	43.99847	43.80444	43.11301	12.0205	31.32288	0.068587	32.26094	42.82212	40.03766	30.21653	41.20116	39.57609	42.18265
	75	45.77668	47.29004	46.33821	46.37484	12.35893	31.06362	0.894807	33.43734	44.2169	39.54277	33.01737	44.08334	38.45022	45.55458
	100	47.56026	49.69803	48.27568	48.64522	13.84742	31.16682	2.503763	34.01887	45.28684	39.21648	35.3358	46.37513	37.25798	48.05312
	300	53.62967	55.80415	55.17486	54.10801	20.8825	33.32212	11.44071	35.60849	50.25433	43.38517	42.68651	53.19144	30.47128	55.32224
	500	55.89288	57.10471	57.29797	55.69599	23.78858	33.43541	15.26608	36.38423	53.19485	47.07597	43.54536	55.14843	27.71222	57.67557
10	25	26.20591	26.17894	26.20362	26.16441	8.911141	22.36903	0.012248	18.56044	26.20747	26.21252	19.91302	26.10162	26.21252	26.10162
	50	32.92478	32.62452	32.9185	32.57029	8.085883	23.60563	0.013231	23.38573	32.84868	32.28796	23.79417	32.24711	32.28662	32.24856
	75	36.61074	36.72262	36.67381	36.4733	7.500088	22.98086	0.01568	26.77615	36.27986	34.78035	26.65738	35.65859	34.69837	36.04621
	100	38.72808	39.50171	38.89185	39.13123	7.186997	21.78738	0.015822	28.57695	38.05171	35.61898	28.56804	38.05038	35.46237	38.77332
	300	45.90943	49.1432	47.16121	48.1629	10.99609	20.83363	4.775045	32.27115	42.72355	35.51812	36.74327	46.41602	32.20981	48.61079
	500	49.00246	52.38557	50.88488	51.18322	14.41109	21.33188	8.750866	32.97366	44.99477	37.31474	39.9831	49.62644	29.59975	52.15933
20	25	17.73947	17.73905	17.73947	17.73905	6.315752	15.96348	0.016543	12.73405	17.73947	17.73947	14.58009	17.73905	17.73947	17.73905
	50	22.39299	22.3861	22.39212	22.38581	5.630206	16.83831	0.008195	15.28714	22.38721	22.38777	17.45102	22.37947	22.38777	22.37947
	75	25.84252	25.83419	25.85348	25.82776	5.085901	16.41516	0.008825	17.58261	25.85555	25.70234	19.68407	25.78482	25.70066	25.78482
	100	28.62065	28.52619	28.60175	28.51802	4.647648	15.71561	0.011822	19.89782	28.53917	28.0854	21.77731	28.39667	28.09237	28.40057
	300	37.90589	39.92346	38.40439	39.75421	3.644785	10.58023	0.09498	28.14217	36.43024	32.24161	29.64596	38.29482	31.91957	39.73121
	500	41.44068	44.75752	42.65529	44.216	5.962263	10.43768	2.655644	29.96604	38.62657	32.06352	33.62173	42.42087	30.35481	44.6411

Table A4: Mean improvement versus the worst result of ofv, for the ECM assignment rule, for instances with $F = 4, m = 20$ and $n = 300$

T	R	AR_v1	AR_v2	ATC_v1	ATC_v2	EDD	MDD	SLK	WEDD	WMDD	WSLK_SPT	WSLKP	WSPT_EDD	WSPT	WSPT_WEDD
0.2	0.2	64.25441	65.34216	63.80325	57.70949	12.24131	26.83024	0.105125	57.99059	62.94004	61.43482	49.24267	46.84992	62.01278	63.37731
	0.4	60.66185	66.07791	62.74136	58.04783	13.18769	28.35636	0.189665	47.32081	54.27505	47.34924	41.46973	51.45967	46.01854	61.60139
	0.6	55.14366	68.87614	61.38017	64.14815	17.07812	36.3735	1.17955	31.74854	42.65184	23.73004	42.60323	60.49735	17.0717	64.28738
	0.8	60.84365	79.34489	69.22857	80.6124	53.0981	65.67701	39.8373	34.82827	45.1364	17.19735	64.2443	78.96372	0.057996	78.71229
	1	67.8919	87.24468	77.48361	91.84179	85.71244	90.87781	77.21817	46.60616	52.51902	23.37273	86.05079	93.05796	0	91.94799
0.4	0.2	56.42538	55.95621	56.05206	52.2475	7.835555	19.89533	0.052411	50.95995	55.16037	53.55954	41.93636	44.47833	53.55036	54.97887
	0.4	57.30358	57.5224	57.39655	53.73772	7.708108	20.36957	0.007742	49.02692	54.28638	47.94732	38.71056	49.04659	46.78599	55.85312
	0.6	57.4512	60.75886	58.35856	58.2292	8.167337	20.80796	0.043615	47.25984	52.17154	39.8634	39.47372	55.39192	36.96286	57.64636
	0.8	55.86119	63.24423	57.9153	63.11676	8.241373	22.70628	0.114066	47.64565	49.44498	31.62808	39.40523	61.09749	24.519	61.89622
	1	53.84533	62.6057	56.72116	64.36869	8.417795	24.25017	0.289598	48.35427	47.47464	27.69891	38.36426	64.96719	16.51843	64.96719
0.6	0.2	47.98593	47.09808	47.36829	45.89407	5.752516	14.90181	0	43.56108	47.23397	45.24648	37.70671	42.1398	45.49668	46.52172
	0.4	48.57526	48.48403	48.32981	47.8751	5.386573	14.19233	0.000273	42.8543	47.50092	42.87491	36.7032	46.1751	42.34497	47.2045
	0.6	47.67272	48.57636	47.743	48.54295	4.481084	13.70029	0.027693	39.44913	46.72174	40.27735	35.38961	48.67326	38.62615	48.67326
	0.8	46.72024	48.25276	46.90472	48.38293	4.107932	14.2677	0.019696	32.9105	45.50259	38.47167	33.82605	48.42384	35.49776	48.42384
	1	44.63048	46.37894	45.00139	46.55442	4.24981	14.39336	0.036413	27.30344	43.09218	35.32306	31.55921	46.79559	31.43584	46.79559
0.8	0.2	38.04574	37.85671	38.03828	37.83174	4.811947	12.36288	0	31.02643	37.84995	37.38409	32.24185	37.76325	37.35314	37.76325
	0.4	37.57689	37.58368	37.60911	37.62719	3.794468	11.45455	0	22.04465	37.61782	36.25272	30.70261	37.56592	36.13733	37.56592
	0.6	36.44848	36.60802	36.49645	36.55094	3.061238	10.02806	0.007947	16.77602	36.46768	34.89753	28.66371	36.49662	34.25573	36.49662
	0.8	35.21399	35.27056	35.21792	35.27276	2.782915	9.752903	0.029935	14.65493	35.13875	32.66219	26.61867	35.24025	31.80682	35.24025
	1	34.49574	34.85742	34.55301	34.84298	2.473578	9.72435	0.081705	14.88454	34.20777	31.35074	25.48692	34.76249	30.01216	34.76249
1	0.2	28.89698	28.88513	28.91191	28.89659	3.376316	9.14981	0	12.55172	28.94111	28.89199	26.10221	28.88304	28.89199	28.88304
	0.4	28.47131	28.43337	28.46458	28.43235	2.609832	8.106152	0.032195	12.29188	28.45495	28.492	24.83537	28.43732	28.49961	28.43732
	0.6	28.22312	28.23803	28.18231	28.24683	2.257336	7.813285	0.015577	11.62465	28.25958	27.95696	23.76494	28.1967	27.85935	28.1967
	0.8	27.86286	27.93697	27.95502	27.9231	2.220144	7.846587	0.014563	11.78088	27.88209	27.47559	22.51869	27.85207	27.26778	27.85207
	1	27.23376	27.14664	27.17391	27.13913	1.848679	7.002484	0.072901	11.32362	27.15733	26.61428	20.96103	27.185	26.26223	27.185

Table A5: Mean improvement versus the worst result of ofv, for the EUM assignment rule, for instances with $F = 4$

m	n	AR_v1	AR_v2	ATC_v1	ATC_v2	EDD	MDD	SLK	WEDD	WMDD	WSLK_SPT	WSLKP	WSPT_EDD	WSPT	WSPT_WEDD
5	25	37.15019	36.53628	37.13543	36.21688	12.55336	29.74868	0.04771	27.35411	37.08952	36.41243	25.17477	35.30021	36.38606	35.33566
	50	43.60984	43.99847	43.80444	43.11301	12.0205	31.32288	0.068587	32.26094	42.82212	40.03766	30.21653	41.20116	39.57609	42.18265
	75	45.77668	47.29004	46.33821	46.37484	12.35893	31.06362	0.894807	33.43734	44.2169	39.54277	33.01737	44.08334	38.45022	45.55458
	100	47.56026	49.69803	48.27568	48.64522	13.84742	31.16682	2.503763	34.01887	45.28684	39.21648	35.3358	46.37513	37.25798	48.05312
	300	53.62967	55.80415	55.17486	54.10801	20.8825	33.32212	11.44071	35.60849	50.25433	43.38517	42.68651	53.19144	30.47128	55.32224
	500	55.89288	57.10471	57.29797	55.69599	23.78858	33.43541	15.26608	36.38423	53.19485	47.07597	43.54536	55.14843	27.71222	57.67557
10	25	26.20591	26.17894	26.20362	26.16441	8.911141	22.36903	0.012248	18.56044	26.20747	26.21252	19.91302	26.10162	26.21252	26.10162
	50	32.92478	32.62452	32.9185	32.57029	8.085883	23.60563	0.013231	23.38573	32.84868	32.28796	23.79417	32.24711	32.28662	32.24856
	75	36.61074	36.72262	36.67381	36.4733	7.500088	22.98086	0.01568	26.77615	36.27986	34.78035	26.65738	35.65859	34.69837	36.04621
	100	38.72808	39.50171	38.89185	39.13123	7.186997	21.78738	0.015822	28.57695	38.05171	35.61898	28.56804	38.05038	35.46237	38.77332
	300	45.90943	49.1432	47.16121	48.1629	10.99609	20.83363	4.775045	32.27115	42.72355	35.51812	36.74327	46.41602	32.20981	48.61079
	500	49.00246	52.38557	50.88488	51.18322	14.41109	21.33188	8.750866	32.97366	44.99477	37.31474	39.9831	49.62644	29.59975	52.15933
20	25	17.73947	17.73905	17.73947	17.73905	6.315752	15.96348	0.016543	12.73405	17.73947	17.73947	14.58009	17.73905	17.73947	17.73905
	50	22.39299	22.3861	22.39212	22.38581	5.630206	16.83831	0.008195	15.28714	22.38721	22.38777	17.45102	22.37947	22.38777	22.37947
	75	25.84252	25.83419	25.85348	25.82776	5.085901	16.41516	0.008825	17.58261	25.85555	25.70234	19.68407	25.78482	25.70066	25.78482
	100	28.62065	28.52619	28.60175	28.51802	4.647648	15.71561	0.011822	19.89782	28.53917	28.0854	21.77731	28.39667	28.09237	28.40057
	300	37.90589	39.92346	38.40439	39.75421	3.644785	10.58023	0.09498	28.14217	36.43024	32.24161	29.64596	38.29482	31.91957	39.73121
	500	41.44068	44.75752	42.65529	44.216	5.962263	10.43768	2.655644	29.96604	38.62657	32.06352	33.62173	42.42087	30.35481	44.6411

Table A6: Mean improvement versus the worst result of ofv, for the EUM assignment rule, for instances with $F = 4, m = 20$ and $n = 300$

T	R	AR_v1	AR_v2	ATC_v1	ATC_v2	EDD	MDD	SLK	WEDD	WMDD	WSLK_SPT	WSLKP	WSPT_EDD	WSPT	WSPT_WEDD
0.2	0.2	64.25441	65.34216	63.80325	57.70949	12.24131	26.83024	0.105125	57.99059	62.94004	61.43482	49.24267	46.84992	62.01278	63.37731
	0.4	60.66185	66.07791	62.74136	58.04783	13.18769	28.35636	0.189665	47.32081	54.27505	47.34924	41.46973	51.45967	46.01854	61.60139
	0.6	55.14366	68.87614	61.38017	64.14815	17.07812	36.3735	1.17955	31.74854	42.65184	23.73004	42.60323	60.49735	17.0717	64.28738
	0.8	60.84365	79.34489	69.22857	80.6124	53.0981	65.67701	39.8373	34.82827	45.1364	17.19735	64.2443	78.96372	0.057996	78.71229
	1	67.8919	87.24468	77.48361	91.84179	85.71244	90.87781	77.21817	46.60616	52.51902	23.37273	86.05079	93.05796	0	91.94799
0.4	0.2	56.42538	55.95621	56.05206	52.2475	7.835555	19.89533	0.052411	50.95995	55.16037	53.55954	41.93636	44.47833	53.55036	54.97887
	0.4	57.30358	57.5224	57.39655	53.73772	7.708108	20.36957	0.007742	49.02692	54.28638	47.94732	38.71056	49.04659	46.78599	55.85312
	0.6	57.4512	60.75886	58.35856	58.2292	8.167337	20.80796	0.043615	47.25984	52.17154	39.8634	39.47372	55.39192	36.96286	57.64636
	0.8	55.86119	63.24423	57.9153	63.11676	8.241373	22.70628	0.114066	47.64565	49.44498	31.62808	39.40523	61.09749	24.519	61.89622
	1	53.84533	62.6057	56.72116	64.36869	8.417795	24.25017	0.289598	48.35427	47.47464	27.69891	38.36426	64.96719	16.51843	64.96719
0.6	0.2	47.98593	47.09808	47.36829	45.89407	5.752516	14.90181	0	43.56108	47.23397	45.24648	37.70671	42.1398	45.49668	46.52172
	0.4	48.57526	48.48403	48.32981	47.8751	5.386573	14.19233	0.000273	42.8543	47.50092	42.87491	36.7032	46.1751	42.34497	47.2045
	0.6	47.67272	48.57636	47.743	48.54295	4.481084	13.70029	0.027693	39.44913	46.72174	40.27735	35.38961	48.67326	38.62615	48.67326
	0.8	46.72024	48.25276	46.90472	48.38293	4.107932	14.2677	0.019696	32.9105	45.50259	38.47167	33.82605	48.42384	35.49776	48.42384
	1	44.63048	46.37894	45.00139	46.55442	4.24981	14.39336	0.036413	27.30344	43.09218	35.32306	31.55921	46.79559	31.43584	46.79559
0.8	0.2	38.04574	37.85671	38.03828	37.83174	4.811947	12.36288	0	31.02643	37.84995	37.38409	32.24185	37.76325	37.35314	37.76325
	0.4	37.57689	37.58368	37.60911	37.62719	3.794468	11.45455	0	22.04465	37.61782	36.25272	30.70261	37.56592	36.13733	37.56592
	0.6	36.44848	36.60802	36.49645	36.55094	3.061238	10.02806	0.007947	16.77602	36.46768	34.89753	28.66371	36.49662	34.25573	36.49662
	0.8	35.21399	35.27056	35.21792	35.27276	2.782915	9.752903	0.029935	14.65493	35.13875	32.66219	26.61867	35.24025	31.80682	35.24025
	1	34.49574	34.85742	34.55301	34.84298	2.473578	9.72435	0.081705	14.88454	34.20777	31.35074	25.48692	34.76249	30.01216	34.76249
1	0.2	28.89698	28.88513	28.91191	28.89659	3.376316	9.14981	0	12.55172	28.94111	28.89199	26.10221	28.88304	28.89199	28.88304
	0.4	28.47131	28.43337	28.46458	28.43235	2.609832	8.106152	0.032195	12.29188	28.45495	28.492	24.83537	28.43732	28.49961	28.43732
	0.6	28.22312	28.23803	28.18231	28.24683	2.257336	7.813285	0.015577	11.62465	28.25958	27.95696	23.76494	28.1967	27.85935	28.1967
	0.8	27.86286	27.93697	27.95502	27.9231	2.220144	7.846587	0.014563	11.78088	27.88209	27.47559	22.51869	27.85207	27.26778	27.85207
	1	27.23376	27.14664	27.17391	27.13913	1.848679	7.002484	0.072901	11.32362	27.15733	26.61428	20.96103	27.185	26.26223	27.185

Table A7: Average result of runtime, in seconds, for the ES assignment rule, for instances with $F = 4$

m	n	AR_v1	AR_v2	ATC_v1	ATC_v2	EDD	MDD	SLK	WEDD	WMDD	WSLK_SPT	WSLKP	WSPT_EDD	WSPT	WSPT_WEDD
5	25	0,000135	0,000155	0,000138	0,000155	1,79E-05	7,72E-05	8,74E-05	1,78E-05	7,77E-05	7,32E-05	8,44E-05	9,2E-05	8,38E-05	0,000104
	50	0,00056	0,000546	0,000599	0,000553	3,32E-05	0,000282	0,000291	3,43E-05	0,000288	0,000288	0,000287	0,000345	0,000285	0,000347
	75	0,001327	0,001299	0,001421	0,001318	5,26E-05	0,00069	0,000688	5,37E-05	0,000639	0,000672	0,000679	0,000782	0,000656	0,000739
	100	0,002328	0,002316	0,002398	0,002354	6,87E-05	0,001242	0,001197	7,01E-05	0,001159	0,001111	0,001193	0,001294	0,001165	0,001303
	300	0,024215	0,023574	0,024385	0,02427	0,000228	0,012432	0,012282	0,00023	0,011954	0,011802	0,012147	0,013249	0,012201	0,013231
	500	0,075143	0,07577	0,075536	0,077444	0,000412	0,037196	0,037145	0,000418	0,037079	0,038196	0,037813	0,042106	0,037204	0,042391
10	25	0,000166	0,000168	0,00017	0,000168	1,79E-05	9,46E-05	9,37E-05	1,95E-05	8,84E-05	8,84E-05	9,39E-05	0,000131	9,13E-05	0,00013
	50	0,00063	0,000643	0,00065	0,000644	3,44E-05	0,00035	0,000359	3,74E-05	0,000331	0,000325	0,000336	0,000465	0,000337	0,000474
	75	0,00147	0,001443	0,001529	0,001452	5,52E-05	0,000798	0,000802	5,62E-05	0,000724	0,000721	0,000757	0,001052	0,000739	0,001048
	100	0,002522	0,002503	0,002611	0,002517	6,97E-05	0,001448	0,00135	7,29E-05	0,001284	0,001307	0,001312	0,001846	0,001279	0,001837
	300	0,026735	0,02652	0,026531	0,02671	0,000235	0,013231	0,013403	0,000237	0,013117	0,013206	0,013292	0,018048	0,013044	0,018145
	500	0,083873	0,083074	0,08396	0,084147	0,000424	0,041306	0,041611	0,000429	0,041385	0,0411	0,041579	0,055571	0,041176	0,05555
20	25	0,000212	0,000216	0,000216	0,000212	1,89E-05	0,000119	0,000117	2,03E-05	0,000113	0,000112	0,000115	0,000248	0,000115	0,000251
	50	0,000803	0,000805	0,000859	0,000804	3,65E-05	0,000448	0,000435	3,95E-05	0,000419	0,000412	0,000424	0,000899	0,000417	0,000904
	75	0,001873	0,001865	0,001912	0,001867	5,7E-05	0,001008	0,000993	5,9E-05	0,000955	0,000952	0,000963	0,002003	0,000949	0,002009
	100	0,004105	0,003284	0,003324	0,003852	7,51E-05	0,001712	0,001706	7,72E-05	0,001677	0,001665	0,001686	0,003505	0,001664	0,003508
	300	0,033207	0,03312	0,033112	0,033433	0,00025	0,016648	0,016609	0,000254	0,016719	0,016617	0,016827	0,033773	0,016623	0,033895
	500	0,103595	0,102996	0,103398	0,103623	0,000501	0,051223	0,051173	0,000459	0,051547	0,051104	0,051848	0,100409	0,051112	0,100605

Table A8: Average result of runtime, in seconds, for the ECM assignment rule, for instances with $F = 4$

m	n	AR_v1	AR_v2	ATC_v1	ATC_v2	EDD	MDD	SLK	WEDD	WMDD	WSLK_SPT	WSLKP	WSPT_EDD	WSPT	WSPT_WEDD
5	25	0,000156	0,000156	0,000157	0,000156	1,78E-05	8,37E-05	7,79E-05	1,94E-05	7,43E-05	7,5E-05	8,68E-05	9,03E-05	8,52E-05	9,96E-05
	50	0,000547	0,000563	0,000604	0,000558	3,35E-05	0,000301	0,000285	3,63E-05	0,000291	0,000296	0,000295	0,000339	0,00029	0,000334
	75	0,001333	0,001327	0,00142	0,001332	5,36E-05	0,0007	0,000709	5,63E-05	0,000674	0,000718	0,000701	0,000728	0,000659	0,000735
	100	0,002323	0,002388	0,002451	0,002393	6,89E-05	0,001255	0,001254	7,28E-05	0,001188	0,001172	0,001152	0,001286	0,001182	0,001358
	300	0,0246	0,024088	0,024206	0,024272	0,000229	0,012458	0,012323	0,000251	0,012514	0,011928	0,012459	0,013097	0,011953	0,013404
	500	0,076257	0,077642	0,077779	0,078164	0,000413	0,03723	0,0379	0,000423	0,03766	0,038168	0,038171	0,041957	0,037142	0,04218
10	25	0,000161	0,000165	0,000171	0,000166	1,94E-05	9,33E-05	9,4E-05	2,18E-05	8,66E-05	9E-05	9,32E-05	0,000131	9,03E-05	0,000131
	50	0,000643	0,000655	0,000693	0,000651	3,66E-05	0,000349	0,000352	4,12E-05	0,000341	0,000353	0,000347	0,000468	0,000335	0,000473
	75	0,001414	0,001433	0,001526	0,001433	5,66E-05	0,000811	0,000807	6,27E-05	0,000736	0,000782	0,000771	0,001041	0,000748	0,00104
	100	0,002587	0,002594	0,002607	0,002592	7,16E-05	0,001376	0,001372	7,91E-05	0,001285	0,001338	0,001385	0,001826	0,00129	0,001826
	300	0,02697	0,026563	0,027142	0,02679	0,000236	0,013436	0,013978	0,000246	0,013315	0,013366	0,013852	0,017899	0,01315	0,017895
	500	0,08402	0,083525	0,084485	0,084113	0,000426	0,041218	0,041335	0,000444	0,04148	0,041632	0,040741	0,055187	0,041062	0,055144
20	25	0,000209	0,000212	0,000217	0,000212	2,02E-05	0,000117	0,000119	2,26E-05	0,000114	0,000118	0,000116	0,000247	0,000114	0,000249
	50	0,000803	0,000808	0,000863	0,000805	3,83E-05	0,00044	0,000445	4,43E-05	0,000416	0,000444	0,000438	0,000899	0,000419	0,0009
	75	0,001878	0,001872	0,001917	0,001868	5,86E-05	0,001012	0,001006	6,53E-05	0,000955	0,001008	0,000997	0,002	0,000949	0,002
	100	0,004398	0,003644	0,003326	0,004405	7,58E-05	0,001707	0,001711	8,6E-05	0,001676	0,001723	0,001706	0,0035	0,001663	0,003505
	300	0,033396	0,033398	0,033266	0,033514	0,000253	0,016769	0,016779	0,000266	0,016776	0,016666	0,016649	0,033883	0,016781	0,033821
	500	0,103894	0,104056	0,104327	0,10396	0,000454	0,051366	0,051408	0,000463	0,051836	0,05149	0,051332	0,100223	0,051381	0,100041

Table A9: Average result of runtime, in seconds, for the EUM assignment rule, for instances with $F = 4$

m	n	AR_v1	AR_v2	ATC_v1	ATC_v2	EDD	MDD	SLK	WEDD	WMDD	WSLK_SPT	WSLKP	WSPT_EDD	WSPT	WSPT_WEDD
5	25	0,000439	0,00044	0,000382	0,000456	0,000265	0,000384	0,000379	0,000309	0,000374	0,000322	0,000375	0,000328	0,000369	0,000384
	50	0,00162	0,001618	0,001607	0,001728	0,001102	0,001375	0,001418	0,001154	0,00135	0,001355	0,001412	0,001402	0,001362	0,001399
	75	0,0039	0,003901	0,003866	0,003966	0,002466	0,003171	0,003372	0,002535	0,003275	0,003238	0,003333	0,003182	0,003269	0,003394
	100	0,006776	0,006574	0,006931	0,007013	0,004711	0,005494	0,005624	0,004684	0,005831	0,005799	0,005907	0,006003	0,005441	0,005769
	300	0,071301	0,070743	0,071491	0,072093	0,048853	0,058768	0,06036	0,048863	0,060057	0,059242	0,060577	0,063048	0,058843	0,062087
	500	0,228695	0,228309	0,226871	0,22456	0,156521	0,191262	0,191144	0,153538	0,186811	0,188618	0,191386	0,196357	0,190246	0,203284
10	25	0,000475	0,000471	0,000472	0,000467	0,000324	0,000408	0,000424	0,000335	0,000401	0,0004	0,000409	0,000433	0,000399	0,000434
	50	0,001897	0,001892	0,001841	0,00199	0,001261	0,001666	0,002313	0,00157	0,001552	0,001563	0,001654	0,001684	0,001564	0,001698
	75	0,004306	0,004301	0,004256	0,004264	0,002951	0,003664	0,003756	0,002998	0,003663	0,003586	0,003625	0,003844	0,003624	0,003871
	100	0,007622	0,007423	0,007569	0,007556	0,005189	0,006231	0,006546	0,005222	0,006373	0,00634	0,006498	0,006832	0,006248	0,006858
	300	0,079665	0,079414	0,080029	0,079841	0,053737	0,066367	0,067731	0,05364	0,066384	0,067105	0,068127	0,07212	0,066735	0,072591
	500	0,251105	0,250332	0,25079	0,250282	0,168004	0,210606	0,209898	0,167105	0,210026	0,209264	0,210754	0,221143	0,209663	0,22473
20	25	0,000609	0,000606	0,000609	0,000641	0,00042	0,000538	0,000538	0,000437	0,000518	0,000518	0,000534	0,000642	0,000516	0,000648
	50	0,002349	0,002344	0,002363	0,002429	0,001615	0,002055	0,002077	0,00167	0,001995	0,002009	0,002055	0,002469	0,001986	0,002465
	75	0,005837	0,005529	0,005535	0,005535	0,003774	0,004723	0,004709	0,003781	0,004665	0,004689	0,004702	0,005688	0,005045	0,005703
	100	0,010315	0,009805	0,009841	0,009853	0,006549	0,008324	0,008236	0,006516	0,008294	0,008327	0,008316	0,01006	0,008309	0,010123
	300	0,100574	0,100229	0,100512	0,100716	0,067438	0,084285	0,083537	0,0667	0,084049	0,084646	0,084705	0,103506	0,084201	0,102522
	500	0,313659	0,311917	0,312328	0,31237	0,209471	0,262756	0,261753	0,207563	0,26182	0,26202	0,262849	0,312325	0,261585	0,313836

Table A10: Mean improvement versus worst the result of ofv and average of runtime, in seconds, for the most effective heuristics, for instances with $F = 4$

		ofv						runtime					
m	n	ES_AR_v2	ES_ATC_v1	ECM_AR_v2	ECM_ATC_v1	EUM_AR_v2	EUM_ATC_v1	ES_AR_v2	ES_ATC_v1	ECM_AR_v2	ECM_ATC_v1	EUM_AR_v2	EUM_ATC_v1
5	25	1,432636	2,978084	3,184823	4,169654	6,235235	7,119671	0,000155	0,000138	0,000156	0,000157	0,00044	0,000382
	50	2,703614	4,089977	5,216714	4,661745	10,97327	9,858839	0,000546	0,000599	0,000563	0,000604	0,001618	0,001607
	75	3,917298	4,919666	6,615855	4,250334	14,22069	11,63131	0,001299	0,001421	0,001327	0,00142	0,003901	0,003866
	100	4,991334	5,991287	7,781574	3,885786	16,28256	12,71957	0,002316	0,002398	0,002388	0,002451	0,006574	0,006931
	300	7,385606	11,55918	11,34691	6,212129	19,52107	16,52377	0,023574	0,024385	0,024088	0,024206	0,070743	0,071491
	500	6,685977	14,73987	11,79465	10,31422	18,71756	18,61617	0,07577	0,075536	0,077642	0,077779	0,228309	0,226871
10	25	0,64535	0,725786	1,387965	1,424917	3,271126	3,315851	0,000168	0,00017	0,000165	0,000171	0,000471	0,000472
	50	1,065322	1,718125	2,034356	2,510716	4,5745	4,906541	0,000643	0,00065	0,000655	0,000693	0,001892	0,001841
	75	1,616123	1,987707	2,709574	2,572376	6,160131	5,687255	0,001443	0,001529	0,001433	0,001526	0,004301	0,004256
	100	2,420456	2,383973	3,506711	2,203229	7,558001	5,959227	0,002503	0,002611	0,002594	0,002607	0,007423	0,007569
	300	7,57548	5,791868	7,504827	1,685169	11,5781	6,427832	0,02652	0,026531	0,026563	0,027142	0,079414	0,080029
	500	10,20545	7,845445	8,619859	2,23039	11,13987	5,930242	0,083074	0,08396	0,083525	0,084485	0,250332	0,25079
20	25	0,308112	0,308769	0,88975	0,890289	2,131447	2,132241	0,000216	0,000216	0,000212	0,000217	0,000606	0,000609
	50	0,490971	0,52779	0,89834	0,905754	2,381375	2,386529	0,000805	0,000859	0,000808	0,000863	0,002344	0,002363
	75	0,661448	0,724231	1,13051	1,158389	2,641496	2,673653	0,001865	0,001912	0,001872	0,001917	0,005529	0,005535
	100	0,776005	0,963587	1,324914	1,441377	2,994792	2,991766	0,003284	0,003324	0,003644	0,003326	0,009805	0,009841
	300	4,460022	2,692605	4,062475	0,94165	6,170418	2,916427	0,03312	0,033112	0,033398	0,033266	0,100229	0,100512
	500	7,215595	3,986961	6,01203	0,969316	7,725953	2,539403	0,102996	0,103398	0,104056	0,104327	0,311917	0,312328

Table A11: Mean improvement versus the worst result of ofv, for the most effective heuristics, for instances with $F = 4, m = 20$ and $n = 300$

		ofv					
T	R	ES_AR_v2	ES_ATC_v1	ECM_AR_v2	ECM_ATC_v1	EUM_AR_v2	EUM_ATC_v1
0.2	0.2	4.85797	5.802809	9.280554	5.207045	14.2694	12.30289
	0.4	7.30118	7.360928	13.85216	5.212373	20.92769	14.06692
	0.6	15.67327	12.92947	20.87475	2.071064	30.05198	13.76741
	0.8	37.77614	22.9561	32.59198	0.31022	43.90677	16.35381
	1	52.60437	31.0857	43.58966	0.556	51.70171	15.91714
0.4	0.2	2.384725	4.319887	4.003945	4.214042	6.677497	8.938871
	0.4	2.802485	6.557663	5.077512	4.717025	10.72114	10.80532
	0.6	6.206701	6.671049	7.681845	2.085521	15.2235	8.773435
	0.8	14.88966	11.76079	13.02153	0.422492	21.34542	9.236732
	1	19.75828	11.19591	13.94918	0.384768	23.8255	7.935846
0.6	0.2	2.00385	2.673928	2.002608	2.480972	4.641782	5.129012
	0.4	2.561848	2.951303	2.381897	2.085118	6.015589	5.450714
	0.6	3.359809	2.876476	2.354018	0.750685	6.121623	4.602563
	0.8	3.770365	3.206815	2.981806	0.44389	6.883188	4.362907
	1	4.369046	3.241832	2.881786	0.389549	7.406807	3.719957
0.8	0.2	1.022649	1.2988	1.504147	1.796095	2.373384	2.474744
	0.4	1.126466	1.249613	1.278596	1.329601	2.397932	2.443758
	0.6	1.19815	1.170499	1.366704	1.203687	2.618078	2.672525
	0.8	1.429873	1.3024	1.125799	1.0428	2.686615	2.457737
	1	1.403344	1.262904	1.252951	0.789312	2.637661	2.29416
1	0.2	0.261301	0.285976	0.884808	0.923921	1.258344	1.225275
	0.4	0.531229	0.468093	1.077632	1.121488	1.335803	1.455052
	0.6	0.580468	0.585338	1.043316	0.96859	1.455113	1.506487
	0.8	0.719395	0.772354	0.967921	0.989975	1.576013	1.503603
	1	0.794415	0.810063	0.593579	0.632997	1.394048	1.298944

FACULDADE DE **ECONOMIA**