POLITECNICO DI MILANO

DEPARTMENT OF MECHANICS

Master of Science in Mechanical Engineering

AA 2019-2020

A Restarted Iterated Pareto Greedy algorithm for the multi-objective hybrid flow shop scheduling problem

Candidate:
Michele Tota
Mat. 859389

Supervisor:
Dott. Chunlong Yu

Tutor:
Prof. Quirico Semeraro

To my parents,
that always gave me strength and support.

Table of contents:

.

List of Figures:

List of Tables:

Abstract

The scheduling of flow shops with multiple parallel machines per stage, which is usually referred to as the hybrid flow shop (HFS), is a complex combinatorial optimization problem encountered in many real-world applications. The problem is to determine the allocation of jobs to the parallel machines as well as the sequence of the jobs assigned to each machine, so as to create a Gantt chart to guide the production activities. The basic HFS scheduling problem has been thoroughly studied in recent decades, both from single objective as well as from multi-objective perspectives, but to the best of our knowledge, little has been done to the multi-objective scheduling problem considering the reduction of total machine setup time as one of the objectives. Given that the machine setups act as non-value-added activities which should be avoided or mitigated from the manufacturing practices, it is important to make a proper schedule which results in short total setup time together with other performance indicators, such as productivity and on-time product delivery. For this reason, this thesis focuses on the HFS scheduling problem with the total tardiness and total setup time objectives. In this work, a simple, yet powerful algorithm for the sequence dependent setup time hybrid flow shop problem is proposed. The presented method, known as Restarted Iterated Pareto Greedy or RIPG, is compared to the NGSA-II, which is a well-known algorithm for multi-objective optimization in literature. Computational and statistical analyses demonstrate that the proposed RIPG method outperforms the NGSA-II in the test instances. We conclude that the proposed method is a candidate to be the state-of-art method for this important and practical scheduling problem.

Chapter 1

[1] Introduction

Production scheduling is one of the most complex activities in the management of production systems. It is closely connected with the firm's performance in terms of speed, reliability, flexibility, quality, and cost. The theory of production scheduling, that aims to provide guidelines and methods, for efficient use of resources, has been the subject of countless papers, over the past five decades. Although several features of scheduling problems are still underexplored due to the variety of production environments, the available resources, restrictions may be imposed and there are multiple objectives to be achieved. Moreover, production scheduling is one of the activities of the Planning, Programming and Production Control. It is responsible for deciding the allocation of resources (machines) over time to perform individual items (jobs and/or batch of jobs), in order to better meet a predefined set of criteria. One can understand the production scheduling as a set of functions of decision-making, involving:

- how to allocate jobs on machines over time, called schedule;
- decisions about how to order the jobs on a given machine called sequence,

The scheduling of flow shops with multiple parallel machines per stage, usually referred to as the Hybrid Flow Shop (HFS), is a complex combinatorial problem encountered in many real world applications. Given its importance and complexity, the HFS problem has been intensively studied [1]. Hybrid flow shops are common manufacturing environments in which a set of *n* jobs are to be processed in a series of *m* stages optimizing a given objective function. There are a number of variants, all of which have most of the following characteristics in common:

- The number of processing stages *m* is at least 2.
- Each stage *k* has machines in parallel and at least IN one of the stages.

- All jobs are processed following the same production flow: stage 1, stage 2, …, stage *m*. A job might skip any number of stages provided it is processed in at least one of them.
- Each job *j* requires a processing time *p* in stage *k*. We shall refer to the processing of job *j* in stage *k* as "operation".

In the "standard" form of the HFS problem all jobs and machines are available at time zero, machines at a given stage are identical, any machine can process only one operation at a time and any job can be processed by only one machine at a time; preemption is not allowed, the capacity of buffers between stages is unlimited and problem data is deterministic and known in advance. Although most of the problems described in this review do not fully comply with these assumptions, they mostly differ in two or three aspects only; the standard problem will serve as a "template" to which assumptions and constraints will be added or removed to describe different HFS variants. In particular, in specific case, to better represent the reality of many real industrial cases, a sequence dependent setup time will be considered with also unrelated parallel machines and a constraint of machine eligibility. The first limitation of unrelated machines indicates that the parallel machines in a stage are not identical but there could be differences in terms of processing speed or manufacturing technologies applied. These two limitations are very common in many industrial cases and are also less considered in the previous literature HFS scheduling problem. In terms of the objective, most researches focus on minimizing the makespan of a schedule. However, in most of the cases, makespan is not the most important criterion to considered. Like in the make-to-order environment, job tardiness should be given higher priority than makespan. Also, the setup times/costs [2] are considered as another important indicator to evaluate the schedule quality, but seldom considered in the literature. These motivate us to consider the total tardiness and total setup time as objective functions to be minimized.

The scheduling problem can be denoted using a triplet α|β|γ notation where, α defines the shop configuration, β describes the constraints and assumptions and γ indicates the objective function. Consequently, the described scheduling problem is denoted as:

$$FHm, ((RM^k)_{k=1}^m) \left| M_j, S_{sd} \right| \sum T_j, TST$$

Here, FHm indicates a HFS with m stages; $((RM^k)_{k=1}^m)$ represents that each stage consists of multiple unrelated machines; Mj represents machine eligibility; $\sum T_j$ indicates the total tardiness objective and TST for the total setup time.

Hybrid flow shop scheduling problem is a discrete optimization problem. When multi-objectives are to be optimized, typical multi-objective optimization methods like NSGA-II [3], SPEA2 [4] can be applied. Yet, other algorithms exist, and in particular one potential competitor is the Iterated-greedy search [5]. It has been applied to the single-objective flow shop problem [6], multi-objective flow shop problem [7] and obtained state-of-art results. The HFS scheduling problem, in most cases, are NP-hard. For instance, HFS restricted to two processing stages, even in the case when one stage contains two machines and the other one a single machine, is NP-hard, after the results of [8]. Similarly, the HFS when preemption is allowed results also in strongly NP-hard problems, according to [9]. Moreover, the special case where there is a single machine per stage, known as the flow shop, and the case where there is a single stage with several parallel machines, known as the parallel machines environment, are also NP-hard, [10]. However, with some special properties and precedence relationships, the problem might be solvable in polynomial time[11].

The HFS scheduling problem has attracted a lot of attentions given its complexity and practical relevance. HFS is found in all kinds of real world scenarios including the electronics [12], [13], [14], [15], paper [16] and textile,

[17], industries. Examples are also found in the production of concrete, [18], the manufacturing of photographic film, [19], [20], and others, [21], [22], [23], [24], [25], [26], [27]. We also find examples in non-manufacturing areas like civil engineering [28], internet service architectures [29] and container handling systems [30] [31]. The results, of this research, may be useful for future research, towards the development of new solution methods, and/or for the application of methods investigated in the context of real companies, with this kind of scheduling problem.

The thesis is organized as follows. Chapter 2 conducts a literature review on exact, heuristic and metaheuristic methods that have been proposed over the last decades, also provides a discussion on different methodologies used to solve the problem and their basic features and components. It explains the terminology used to refer to the different assumptions, constraints and objective functions where reviewed papers are briefly commented and only the most important facts are highlighted. Then, the aim is to focus on applying the optimization method called Restarted Iterated Pareto Greedy (RIPG), which is described and presented in Chapter 3. Chapter 4 discusses the obtained results of the experimental campaign. A comparison between the proposed method and the conventional method is described in Chapter 5. The results may be useful for future research, towards the development of new solution methods, and/or for the application of methods investigated in the context of real companies, with this kind of scheduling problem. These aspects with relative research opportunities in HFS scheduling problem concludes the thesis.

Chapter 2

[2] Problem definition

This chapter defines the research problem. Furthermore, this chapter presents some considerations about the limitations emerged from the state-of-the-art analysis. The literature review can be organized in three parts. First, we review different methods solving the HFS scheduling problem, which can be categorized by exact, heuristic and metaheuristic methods, considering the most common assumptions and objectives. Then we review papers and organize them according to different criteria. In particular the criteria considered for the classification were, the encoding and decoding procedure used in the algorithms, the machine selection rule adopted in each stage, the technique used, the adopted constraints on the production chain and the considered objectives.

2.1 The Hybrid Flow Shop: problem description and notation

A Hybrid Flow Shop (HFS) consists of series of production stages, each of which has several machines operating in parallel. Some stages may have only one machine, but at least one stage must have multiple machines. The flow of jobs through the shop is unidirectional. Each job is processed by one machine in each stage and it must go through one or more stage [49]. Depending on the adopted assumptions, machines in each stage can be considered identical, uniform or unrelated. When $p_{ij} = p_j/s_i$ where p_j is the processing time of job j and s_i is the speed of machine i, then the machines are called uniform. If the p_{ij}s are arbitrary then the machines are called unrelated. And both of the uniform and unrelated cases belong to non-identical parallel-machine schedules. In our case, as we already mentioned, there are some differences between them which makes the hypothesis of **unrelated parallel machines** an important assumption close to the real industrial environment. In fact, HFS is often found in the electronic manufacturing environment such as IC packaging and PCB fabrication, where this assumption is often verified. The **setup times** are not included in the processing time and they are **sequence**

dependent. The SDST/HFS problem can be described as follows. A set of n jobs J = {1, 2, ..., n} have to be processed through m production stages {1, 2, ..., m} following the same production route, i.e., first at stage 1, then at stage 2, and so on until last stage m. Each stage k, k = 1, 2, ..., m, has a set of almost identical parallel machines, M_k ($|M_k| \geq 2$ for at least one stage, where $|\bullet|$ denotes the cardinality of a set). Each job $j \in J$ can be processed on one of the $|M_k|$ depending on a machine eligibility criterion. We denote the processing time of job $j \in J$ at stage k as $p_{k,j}$. We have a SDST, denoted as $s_{k,j',j}$, when job $j \in J$ is processed immediately after job $j' \in J$ ($j' \neq j$) on the same machine at stage k. If job $j \in J$ is the first job processed on a machine at stage k, its setup time is denoted as $s_{k,j,j}$. At any time, no job can be processed on more than one machine, and no machine can process more than one job. All jobs are independent and available for processing at time 0. The objective is then to find a schedule so that the Total Setup Time and the Total Tardiness are minimized.

The SDST assumption comes from the need to have a more realistic model of the HFS. Another important mark to be made, is the constraint of machine eligibility which gives a criteria on which machine is eligible to process an operation of a given jobs. The two objectives are to minimize are the total job tardiness and the total setup time.

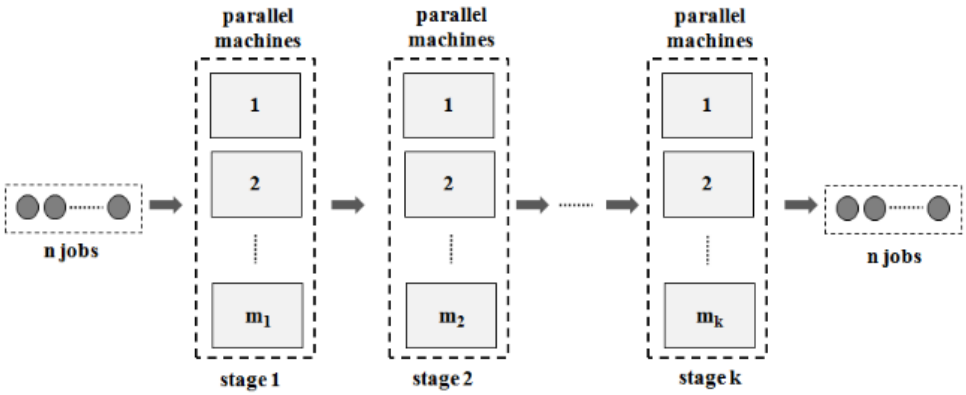


Figura 1- The Hybrid Flowshop

The resume of what stated plus its notation is presented as follows:

- In stage *i*, there are m_i unrelated parallel machines with different processing abilities, where $m_i \geqslant 1$.

- Between the stages *i* and *i* + 1, the buffer capacity is assumed infinite.

- Each job consists of a sequence of operations $O_{i,j}$ where $O_{i,j}$ denotes the i_{th} operation of job *j*, which should be carried out on a selected machine in stage *i*;

- When a job arrives at a stage *i*, it can select exactly one machine from m_i available unrelated parallel machines. The selection is made according to an eligibility index;

- After a job is completed at stage *i*, it may be processed as follows:

    1) the job will be immediately delivered to the subsequent stage when one of the machines at stage *i* + 1 is available;
    2) in cases in which there is no available machine at stage *i* + 1, the job will be stored in the following buffer given the infinite buffer space;

- Each machine in the same stage can process only one job at a time, and each job can be executed by only one machine at a time;

- All jobs and machines are available at time zero;

- Preemption is not allowed; that is, a job cannot be interrupted before the completion of its current operation;

- Setup times are sequence dependent and all the problem data are deterministic and known in advance;

- Machines are reliable and no machine failures can happen.

The following table resumes the adopted notation:

Notation	Description
n	number of jobs ($j = 1,\dots,n$)
m	Number of stages ($i = 1,\dots,m$)
k	Index for machines inside a stage
pi,j,k	Processing time of job j at stage i on machine k
$O_{i,j,k}$	Operation of job j at stage i on machine k
E_i	Set of eligible machines at stage i
$C_{i,j}$	Completion time of $O_{i,j,k}$
d_j	Due date for job j
si,j	Sequence-dependent setup time from job j at stage i.
T_j	Tardiness of job j. $T_j = \max(0, C_j - d_j)$
T	Total Tardiness: $T = \sum_{j=1}^{n} T_j$

Tabella 1- Notation

The scheduling problem can be denoted using a triplet $\alpha|\beta|\gamma$ notation. In this notation, α defines the shop configuration, β describes the constraints and assumptions and γ indicates the objective functions. Consequently, the described scheduling problem is denoted as:

$$FH_{m\,,}\left(\left(RM^{(k)}\right)_{i=1}^{m}\right)|M_j, S_{sd}|\sum T_j\,, TST$$

Here, $FH_{m\,,}$ indicates a HFS with *m* stages; represents that each stage consists of multiple unrelated machines; M_j represents machine eligibility; $\sum T_j\,, TST$ indicates the total tardiness and the total setup time objective.

Chapter 3

[3] Literature review

3.1 Methods for HFS scheduling problem

In literature, methods for HFS scheduling problem can be categorized as exact and heuristic. **Exact methods**, including mathematical programming and **branch & bound**, solve the problem to find an optimal solution. Although the branch and bound was first suggested by [67], the first complete algorithm introduced as a multi-objective branch and bound that we identified was proposed by [68]. Based on the "divide to conquer" idea, it consists in an implicit enumeration principle, viewed as a tree search. The feasible set of the problem to optimize is iteratively partitioned to form sub-problems of the original one. Each sub-problem is evaluated to obtain a lower bound on the sub-problem objective value. The lower bounds on sub-problem objective values are used to construct a proof of optimality without exhaustive search. Uninteresting and infeasible sub-problems are pruned, promising sub-problems are selected and instantiated [46]. However, due to the lack of efficient lower bounds, branch & bound approach is limited to simple shop configurations; also, exact methods require long time for solving large instances. Both facts limit the industrial application of exact methods. A practical idea is to search for quasi-optimal solution in a reasonable time. For this reason, the trend of solving HFS scheduling problems with **heuristic**, especially **metaheuristic**, is increasing [41]. In the past decade, genetic algorithm (GA) has gained the widest applications. **Genetic Algorithms** were introduced by Holland [32] and they have been used in many scheduling problems (see for instance [33], [34]). The GA starts with an initial population of possible solutions called chromosomes to the problem. The relative quality of these chromosomes is determined using a fitness function. This quality is used to determine whether the chromosomes will be used in producing the next generation of chromosomes. The next generation is generally formed via the processes of crossover and mutation. Crossover is the process of combining elements of two chromosomes, whereas mutation means randomly altering elements of a chromosome [35]. Among the metaheuristic methods we can underline different ones such as **population learning**

algorithm [37], **taboo search** [38] and **ant colony system** [39]. In particular, the tabu search algorithm TSNS is commonly considered as the most effective solution method for the considered HFS scheduling problem with related parallel machines. Its high efficiency is obtained due to so called *reduced neighborhood* based on the block properties and the *accelerator* designed for C_{max} computation for all neighbors of the base solution [42]. A detailed description of all TSNS components can be found in [43]. All these focus on minimizing the makespan but it is not the only scheduling problem tackled; [40] focused on minimizing the weighted completion time in proportional flow shops. Also **Hybrid heuristic** are used. **Memetic algorithms** for example are hybrid evolutionary algorithms that combine global and local search by using an evolutionary algorithm to perform exploration while the local search method performs exploitation [36]. There exist hybrid heuristic algorithms that combine **particle swarm optimization** (PSO) [44] [45] with **simulated annealing** (SA) and PSO with **tabu search** (TS), for example. Particle swarm optimization is similar to the genetic algorithm technique for optimization in that rather than concentrating on a single individual implementation, a population of individuals (a "swarm") is considered instead. The algorithm then, rather than moving a single individual around, will move the population around looking for a potential solution. Each individual in the swarm has a position and velocity defined, the algorithm looks at each case to establish the best outcome using the current swarm, and then the whole swarm moves to the new relative location [47]. Instead, **Simulated Annealing (SA)**, introduced in [51], was conceived for combinatorial problems, but can easily be used for continuous problems as well. SA starts with a random solution x^c and creates a new solution x^n by adding a small perturbation to x^c. If the new solution is better than the current one, it is accepted and replaces x^c. In case x^n is worse, SA does not reject it right away, but applies a stochastic acceptance criterion, thus there is still a chance that the new solution will be accepted, albeit only with a certain probability. This probability is a decreasing function of both the order of magnitude of the deterioration and the time the algorithm has

already run. This time factor is controlled by the temperature parameter *T* which is reduced over time; hence, impairments in the objective function become less likely to be accepted and, eventually, SA turns into standard local search. The algorithm stops after a predefined number of iterations R_{max}[48]. Experimental results reveal that these memetic techniques can effectively produce improved solutions over conventional methods with faster convergence [36]. Besides these, in recent years, other less used metaheuristics were proposed for HFS. For example **immune evolutionary algorithm** [52] and **artificial bee colony** [53],[54]. Lastly there have been presented apply new metaheuristics like **water flow-like algorithm** [55], **firefly algorithm** [57], **cuckoo search algorithm** [56] to solve HFS problems. Indeed, different metaheuristics represent different search patterns in the solution space. However, due to the existence of the No-Free Lunch (NFL) Theorem [58], it is more important on how to make use of problem structure information to improve the search procedure than just applying new general purpose optimization methods to HFS problems [41]. Although there exist many different approaches, in last years an important contribution has been given by the Iterated-greedy algorithm. A resume table is presented here.

Year	Authors	Tipology
1960	H. Land, A & G. Doig	Branch & Bound
1975	H. Holland	Genetic Algorithm
1983	G. Kiziltan, E. Yucaoglu	Branch & Bound
1995	J. Kennedy, R. Eberhart	Particle swarm optimization
1998	Y. Shi, R.C. Eberhart	Simulated annealing
2003	J. Jdrzêjowicz, P. Jdrzêjowicz	Population learning algorithm
2004	C. Oðuz, Y. Zinder et al.	Taboo search
2005	C. Oguz, M.Ercan	Genetic Algorithm
2006	K.C. Ying, S.W. Lin	Ant colony system
2006	M. Zandieh, S.M.T.F. Ghomi, S.M.M. Husseini	Immune evolutionary algorithm
2011	F. Choong, S. Phon-Amnuaisuk, M.Y. Alias	Memetic algorithms
2012	F. Pargar, M. Zandieh	Water flow-like algorithm
2013	M.Ciavotta, Minella, Ruiz	Iterated greedy algorithm
2014	M.K. Marichelvam, T. Prabaharan, X.S. Yang	Firefly algorithm
2014	M.K. Marichelvam, T. Prabaharan, X.S. Yang	Cuckoo search algorithm
2015	Z. Cui, X. Gu	Artificial bee colony
2016	] J. Li, Q. Pan, P. Duan	Artificial bee colony

Tabella 2 - methods for HFS

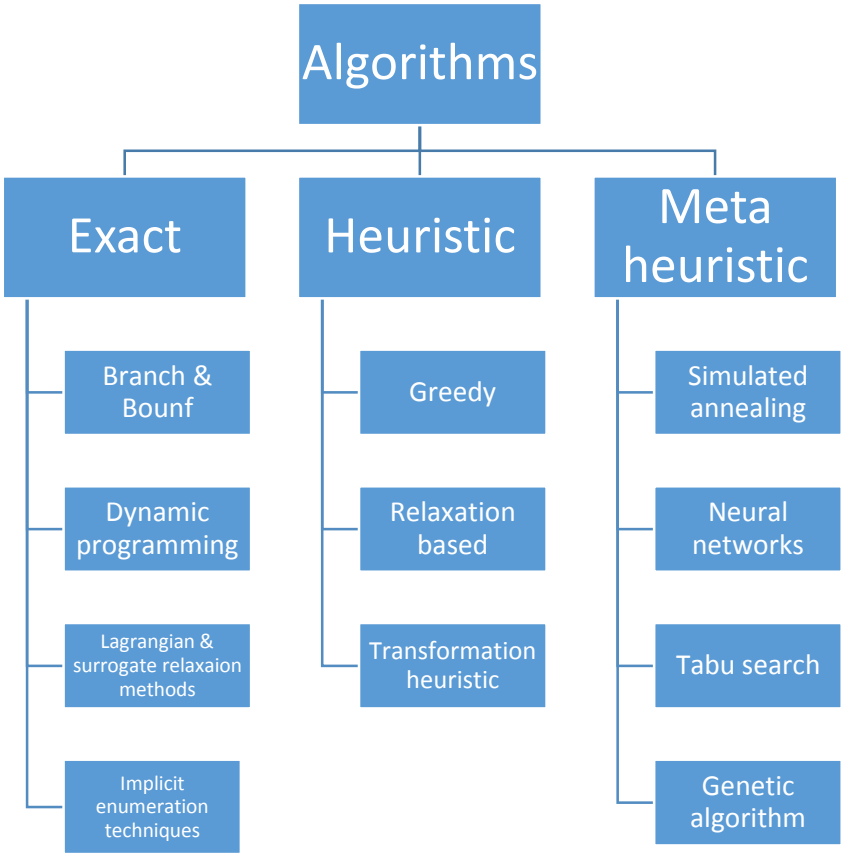


Figura 2 -Methods classification

3.2 Sequence-dependent setup time and unrelated parallel machines

Unrelated parallel machine scheduling is widely applied in manufacturing environments such as the drilling operations for printed circuit board fabrication [113] [114] and the dicing operations for semiconductor wafer manufacturing [115]. A lot of papers about HFS have been published in the literature but only a relatively minor fraction of them consider sequence dependent setup times. In the literature, there are **three types of parallel machines scheduling** even though most of researches are limited to situations in which the processing times are the same across all machines. This type is called identical parallel machines (P_m). In the second type, machines have different speed but each machine works at a consistent rate, Q_m [116] [117]. Finally, when machines are capable of working at different rates and when different jobs could be processed on a given machine at different rates, the environment is said to have unrelated parallel machines, R_m [116]. Developed algorithms to schedule unrelated parallel machines are capable of generating good solutions when applied to all kinds of parallel machine problems and that's why they are important. [118] pointed out that unrelated parallel machine problems remain relatively unstudied and presented a survey of algorithms for single-and multi-objective unrelated parallel machine deterministic scheduling problems. There is much research work considering parallel machines, but few unrelated parallel machines or sequence-dependent set-up times (SDST). In general, such a problem is composed of job allocation and job sequencing onto the machines, simultaneously, with similar but not necessarily identical capabilities. In [66] the makespan and total weighted tardiness are optimized simultaneously with a multi-phase genetic algorithm for searching Pareto optimal solutions of a hybrid flow shop group scheduling problem. [69] uses tabu search procedure to address a sequence-dependent group scheduling problem on a set of unrelated-parallel machines where the run time of each job differs on different machines. [70] proposes a new multi-objective approach for solving a no-wait k-stage flexible flow shop scheduling problem to minimize makespan and mean tardiness. Sequence-dependent setup times

are treated in this problem as one of the prominent practical assumptions. [71] investigates a bi-objective HFS problem with sequence dependent setup time for the minimization of the total weighted tardiness and the total setup time. To efficiently solve this problem, a Pareto-based adaptive bi-objective variable neighborhood search (PABOVNS) is developed and a job sequence based decoding procedure is presented to obtain the corresponding complete schedule. [72] paper presents an enhanced migrating bird optimization (MBO) algorithm and a new heuristic for solving a scheduling problem. The proposed approaches are applied to a permutation flow shop with sequence dependent setup times. Using an adapted neighborhood, a tabu list, a restart mechanism and an original process for selecting a new leader the MBO's behavior is improved and it gave state-of-the-art results when compared with other algorithms reference. [73] proposed a genetic algorithm and a memetic algorithm for the F/S_{ijk}, $prmu/C_{\max}$ [74] introduced an algorithm formed by a new heuristic and a local search improvement scheme for the combined objective of total weighted flow-time and tardiness. [75] offers a comprehensive review of scheduling research with setup times. From an industrial point of view, [119] solve a parallel machine scheduling problem inspired in an aluminium foundry considering objectives related to the metal flow; [120] consider a problem related to multi-product metal foundries; or more recently, [121] tackle, in a metal packaging industry, an unrelated parallel machines scheduling problem with job splitting and sequence-dependent setup times, [122] considers a problem identified in a shipyard modelled by unrelated parallel machines with jobs release dates, precedence constraints and sequence-dependent setup times where machines are eligible, and finally [123] considers a dynamic scheduling problem based on the features of parallel heat treatment furnaces of a manufacturing company.

To the best of our knowledge, the problem under consideration has not been addressed in the literature, although some papers address the unrelated parallel machines problem with machine eligibility and setup times with additional constraints.

Year	Authors	Constraints adopted
1985	Potts, C. N	Unrelated parallel machines
2002	Yu L, Shih HM, Pfund M, Carlyle WM, Fowler JW	Unrelated parallel machines
2002	Kim DW, Kim KH, Jang W, Chen FF	Unrelated parallel machines
2003	Hsieh JC, Chang PC, Hsu LC	Unrelated parallel machines
2003	Rajendran, C., Ziegler, H.	seuqence-dependent setup time
2004	Pfund, M., J. W. Fowler, and J. N. D. Gupta.	Unrelated parallel machines
2005	Ruiz, R., Maroto, C., Alcaraz, J.	seuqence-dependent setup time
2008	Pinedo, M.	Unrelated parallel machines
2008	Ali Allahverdi, C.T. Ng, T.C.E. Cheng, Mikhail Y. Kovalyov,	seuqence-dependent setup time
2010	N. Karimi, M. Zandieh, H.R. Karamooz,	seuqence-dependent setup time
2012	Mir Abbas Bozorgirad, Rasaratnam Logendran,	seuqence-dependent setup time
2014	H. Asefi, F.Jola, M.Rabiee, M.E. Tayebi Araghi	seuqence-dependent setup time
2016	Huixin Tian, Kun Li, Wei Liu	seuqence-dependent setup time
2018	A. Sioud, C. Gagné,	seuqence-dependent setup time

Figura 3- Constraints for HFS

3.3 Multi-objective hybrid Flow Shop

There are several different approaches to the multi-objective optimization. The most immediate and commonly employed methodology is the so-called "***a priori***" approach. As the name implies, this methodology requires to specify some desirability or a prioristic information given by the decision maker so to create a weighted combination of all objectives into one single mathematical function, which effectively turns the problem into a single-objective one. As one can imagine, the main drawback of this approach is about how to set the weights for each objective, which is not a trivial procedure. Furthermore, different objectives are usually measured in different scales, making the choice of the weighs even more complicated. On the other hand, there is a class of techniques referred to as "***a posteriori***" methods. The final goal of this kind of approach is to provide a set of non-dominated solutions that cover the trade-off between the selected objectives. This set of non-dominated solution is referred to as the Pareto frontier. The decision maker, after the optimization has been carried out, selects the desired solution from the Pareto frontier. The approach used in our RIPG methodology is "a posteriori".

In fact, in each step where selection must be performed, the Pareto frontier is identified, and the non-dominated solution set is chosen as the set that will continue in the algorithm.

The literature on multi-objective optimization is very rich. The few proposed multi-objective methods for the PFS and HFS problem are mainly based on evolutionary optimization and on local search techniques e.g. simulated annealing (SA) or tabu search. In [76], there is a comprehensive review of the literature related to this problem. Thus, here we restrict ourselves to only the most significant papers and to some other more recent published material. Focusing only on the "a posteriori" approach, the number of publications in the flowshop literature is reduced to the following works. A genetic algorithm (GA) was proposed by [77] which obtained a Pareto front for makespan and total tardiness. In this algorithm, referred to as MOGA (Multi Objective Genetic Algorithm), the selection phase employs a fitness value assigned to each solution as a function of the weighted sum of the objectives. The weights for each objective are randomly assigned at each iteration of the algorithm. Later, in [78], the authors extended this algorithm by means of a local search procedure applied to every newly generated solution. [79] present a comprehensive study about the effect of adding local search to their previous algorithm [78] The local search is only applied to good individuals and by specifying search directions. This form of local search was shown to give better solutions for many different multi-objective genetic algorithms. [80] proposes a Pareto-based simulated annealing algorithm for makespan and total flowtime criteria and the results of the proposed method is compared with [79]. The makespan and total flowtime objectives are studied in [81], which proposed simulated annealing methods. Two versions of these SA (MOSA and MOSA-II) are shown to outperform the GA of [78] . According to the comprehensive computational evaluation of [76], where 23 methods were tested for the multi-objective flowshop, an enhanced version of MOSA-II algorithm is shown to consistently outperform all other methods. NGSA-II was firstly presented by [82]. A variant of this procedure was presented by with

the name [83] with the name of hMGA and it uses a working population with dynamic size made of only heterogeneous solutions. This choice prevents the algorithm from getting stalled in local optima. [84] presented an iterated greedy (IG) procedure based on the NEH heuristic. This algorithm is an evolution of the IG basic principle for the multi-objective PFSP.

3.3.1 The Scheduling problem with Total Tardiness (TT) objective

Several metaheuristic methods have been used for total tardiness minimization. Most of the methods until now, have been used to the permutation flowshops but not only. [87] used algorithm (GA), [88] used tabu search, [89] proposed a simulated annealing algorithm with local search and [90] used a differential evolution algorithm. In particular, [91] presented a detailed review and comparative evaluation of numerous exact, heuristic and metaheuristic methods for the PFSP with total tardiness objective. [91] also proposed a benchmark suite that is used in this work, to be able to compare the algorithms on a common test set. In most recent works, [92] report the results for parallel and serial execution of several cooperative metaheuristics for both total tardiness and makespan criterion. [94] present a GA with path relinking, and [95] use an elite guided steady-state GA. [93] use a variable iterated greedy algorithm where the number of jobs that are removed from the current permutation varies from 1 to n – 1. [96] use an integrated iterated local search (IILS) that is based on Iterated Local Search (ILS). [97] presents an iterated local search algorithm hybridized with a variable neighborhood descent algorithm. The IG algorithm presented by [99] for the PFSP is a simple and easy to implement, yet powerful and effective metaheuristic [98] proposes a new evolutionary approach with a new mating scheme designed for the problem that can achieve better results as the size of the problem increases.

3.3.2 The scheduling problem with Total Setup Time (TST) objective

We already mentioned that in this work, sequence dependent setup time represents an important feature and constraint of our work. Very less studied is the case where Total Setup Time is considered as objective to minimize. Actually, what is much more frequent is the minimization of Total Flow Time, (TFT) which includes also idleness or the Total Completion Time (TCT) which considers the setup time with processing time and idleness. Actually, there exist some production environments where setup time and cost are much higher than pure processing, that's why it can be very interesting for this case to be investigated. In [104] the authors presented a mixed integer linear programming (MILP) model for the exact solution of small instances and a heuristic called iterative resource scheduling with alternatives (IRSA) for larger ones. [105] proposed three algorithms for the job shop problem with processing alternatives. [106,107] focused on RCPSP with alternatives that is close to the Total Setup Time minimization in Production Scheduling with 13 alternative job shop problems and proposed agent based metaheuristic algorithms to minimize the makespan. [108] presented an integrated model of process planning and scheduling problems which are carried out simultaneously. The authors developed a genetic algorithm to minimize the schedule length. [109] dealt with the setup times in general and published a survey in which many different problems related to the setup times are summarized. The authors also reported on solution approaches and proposed a notation for all of these problems. [110] published a study for a metal casting company concerning the minimization of the total setup costs in which the authors demonstrate the importance of setup times by calculating the savings to the company. The authors proposed a two-phase Pareto heuristic to minimize the makespan and the total setup costs. In the first phase, the makespan is minimized and, in the second phase, the total setup costs are minimized, while the makespan is not allowed to get worse. [111] focused on a single machine earliness and tardiness problem with sequence dependent

setup times. The objective function is to minimize the total setup time, earliness and tardiness. [112] proposed a hybrid simulated annealing algorithm for the single machine problem with sequence dependent setup times. The objective function is given by the sum of the setup costs, delay costs and holding costs.

Year	Authors	Objectives
1981	N. Baba	Total Tardiness
1981	J. Solis Francisco, Wets, J.B. Roger	Total Tardiness
1997	N. Mladenovic, P. Hansen	Total Tardiness
2001	P. Hansen, N. Mladenovic	Total Tardiness
2003	Kis, T.	Total setup time
2004	Yuan, X.M., Khoo, H.H., Spedding, T.A., Bainbridge,	Total setup time
2007	R. Ruiz, T. Stützle	Total Tardiness
2008	Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M.Y.	Total setup time
2009	Shao, X., Li, X., Gao, L., Zhang, C.	Total setup time
2010	[94] E. Vallada, R. Ruiz	Total Tardiness
2010	T. Kellegöz, B. Toklu, J. Wilson	Total Tardiness
2010	. Leung, C.W., Wong, T.N., Maka, K.L., Fung, R.Y.K.	Total setup time
2010	Li, X., Zhang, C., Gao, L., Li, W., Shao, X.	Total setup time
2012	Capek, R., ˇ S°ˇucha, P., Hanz´alek, Z.	Total setup time
2013	T. Chen, X. L	Total Tardiness
2014	R. M'Hallah	Total Tardiness
2015	T. Cura	Total Tardiness

Figura 4- Objectives functions for HFS

3.4 The Restarted Iterated Pareto Greedy Algorithm (RIPG)

The Iterated Greedy (IG) algorithm was first proposed in [59] and its basic mechanism consists of iteratively destructing some elements of a solution, reconstructing a new one using a constructive greedy technique and, finally improving it by means of an optional local search procedure. Hence, the central core of this algorithm is identified by two main phases: the *destruction/construction* phase and the *local search*. During the destruction phase some solution components are removed from a previously constructed

complete candidate solution. The construction procedure then applies a greedy constructive heuristic to reconstruct a complete candidate solution. Once a candidate solution has been completed, an acceptance criterion decides whether the newly constructed solution will replace the incumbent solution. In other words, in the first, some elements of the current solution are randomly removed, and then reinserted in such a way that a new complete, and hopefully better solution is obtained. IG iterates over these steps until some stopping criterion is met. These two *destruction/construction* phases constitute the so-called *greedy phase*. The initial sequence is generated by a well-known NEH heuristic, which build a starting sequence for each objective to optimize. NEH evaluates a total of $[n(n + 1)/2] - 1$ schedules; n of these schedules are complete sequences. This makes the complexity of NEH rise to which can lead to considerable computation times for large instances. However, [85] introduced a data structure that allows to reduce its complexity. The job sequence obtained by applying the NEH heuristic gives the initial solution of the IG algorithm. In our case we have two objectives, so the starting sequences generated for our algorithm will be two. So, basically, the NEH is a greedy constructive method that tests every removed element into all possible positions of the current partial solution until the sequence with the best objective function value is found. Of course this suggests that, The NEH heuristic can optimize only one objective at a time.

IG is currently being studied in many other research works, with similar assumptions to the one of our interest. For example, [61] extended the IG method to other objectives and to sequence-dependent setup times. [62] applied IG to multistage hybrid flow shop scheduling problems with multiprocessor tasks. [64] applied IG algorithms to train scheduling problems and [65] used IG for single machine scheduling problems with sequence-dependent setup times. Finally, [63] used Iterated Greedy algorithms for node placement in street networks.

To give a general scheme about how the algorithm is divided, it can be broken into five phases:

1. ***Initialization***. In this first phase, an initial set of good solutions is generated using two NEH heuristics, each one designed to attain good values for a specific criterion.
2. ***Selection***. The second phase, chooses one solution from the current working set for the next phase.
3. ***Greedy phase***. This phase represents the real core of the entire procedure. It is constituted by the two phases of *destruction* and *construction*.
4. ***Local search***. This phase is applied usually after greedy phase, over a selected element of the current working set.
5. ***Restart*.** This is the last phase procedure is implemented to prevent the algorithm from getting stuck in local optima.

The detailed procedure will be described later in chapter 4.

Chapter 4

[4] Methodology

4.1 A model for the SDST Hybrid Flowshop

As an example, we consider a problem with four jobs (n = 4) and two stages (m = 2), with two machines at stage one ($|M_1|$ = 2) and three machines at stage two ($|M_2|$ = 3). The processing times and setup times are given in Tables 1 and 2, respectively. In this example we consider the special case of unrelated parallel machine where the processing time of the jobs on all parallel machines are identical. A schedule chart is shown in Figure 5.

	J_1	**J_2**	**J_3**	**J_4**
Stage 1	4	5	4	3
Stage 2	5	5	3	2

Tabella 3 - Processing times example

	Stage 1					**Stage 2**			
$J_i \backslash J_j$	**J_1**	**J_2**	**J_3**	**J_4**	**$J_i \backslash J_j$**	**J_1**	**J_2**	**J_3**	**J_4**
J_1	2	3	2	2	J_1	2	3	2	2
J_2	2	2	3	4	J_2	2	1	3	4
J_3	4	2	2	3	J_3	3	3	2	3
J_4	3	3	2	2	J_4	4	3	2	2

. Tabella 4 - Processing times example

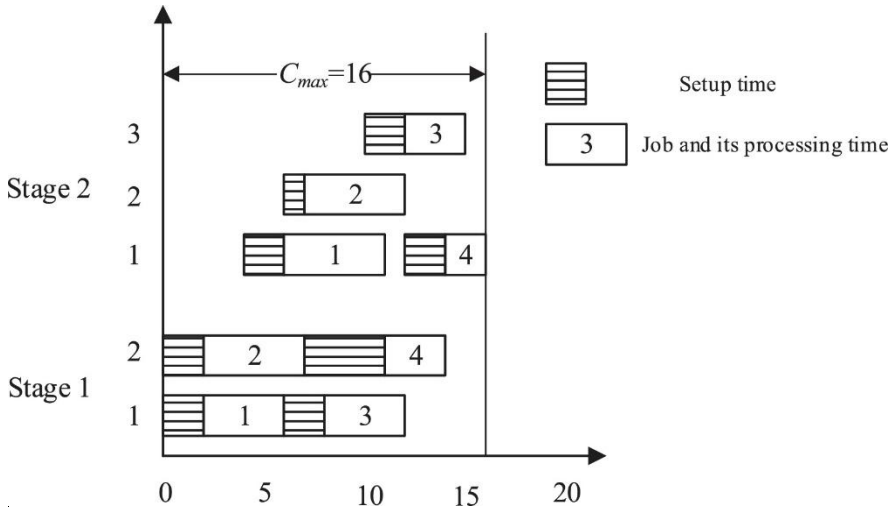


Figura 5- Scheduling example

4.2 Encoding methods

Encoding method consists of representing a schedule by a string of decision variables, or saying, chromosome. A schedule is defined by indicating the start and finish times for each operation on the machine to which it is assigned. Since we are optimizing regular objectives like Setup Time and Total Tardiness, all operations are expected to be started as early as possible. There can be distinguished basically two types of encoding: a **direct encoding**, which usually involves large solution space, that may render inefficient the searching procedure. In fact, It has been demonstrated that the more detailed the encoding, the worse the results. Indeed, **indirect encoding** employing surrogate heuristics in the decoding procedures for completing the solution is usually much efficient than a direct encoding. For this reason, most of the researches use the following indirect encoding scheme: a solution is encoded as a job permutation.

Encoding representations can be farther classified into nine categories

Direct Encoding	- Operation based - Job based - Job pair relation based - Completion time based - Random keys
Indirect Encoding	- Preference list based - Priority rule based - Disjunctive graph based. - Machine based.

Figura 6- Encoding/decoding classifications

These classification takes more sense when contextualized in GA environment with job shops and these nine categories can be grouped into the already mentioned two basic encoding approaches—direct and indirect. So the point is that In direct approach, a schedule is encoded as a chromosome and genetic operators are used to evolve better individual ones. Categories 1 to 5 are examples of this category. In case of indirect approach, a sequence of decision preferences will be encoded into a chromosome. In this, encoding, genetic operators are applied to improve the ordering of various preferences and a Πj schedule is then generated from the sequence of preferences. Categories 6 to 9 are examples of this category. Some words need to be spent on Random Keys Representation (RK), because more than others, it is used sometimes also in Hybrid Flow Shop environment. In this representation, each gene is represented with random numbers generated between 0 and 1. These random numbers in a given chromosome are sorted out and are replaced by integers and now the resulting order is the order of operations in a chromosome. This string is then interpreted into a feasible schedule. Any violation of

precedence constraints can be corrected by a correction algorithm incorporated.

4.3 Decoding methods

Decoding is to derive a schedule from the encoded solution. The encoding procedures described does not contain all the required information and decision variables for constructing a HFS schedule. These missing information like for example machine selection decisions, are determined by some heuristics during the decoding procedure. That's why the solution quality strictly depends on the decoding method. Also in this case, the most used decoding methods are:

1) **List scheduling (LS).** It is a decoding method adopted in many researches (as shown in Table 1); an initial job list L_1 is created in the first stage, according to some objective to optimize. Then jobs are picked out from L_1 sequentially and scheduled as early as possible on the machine selected by a machine assignment rule. In the remaining stages, it is applied the same procedure as in stage 1 except that L_i ($i > 1$) is created by the First-come-first-served (FCFS) rule, that is sorting the jobs increasingly by their completion time in the precedent stage. As a consequence, it happens than especially with increasing number of stages, the sequence of processed jobs can change stage by stage because each time, the next job of the list is scheduled on the machine that is available first. If a tie exists, then usually the job is scheduled on the machine with the smallest index. List schedules are also used in branch-and-bound algorithms for problems in which the set of list schedules is dominant, i.e., contains at least one optimal solution.

2) **Permutation scheduling (PS).** This method is similar to LS except that the job lists in each remaining stage are equal to π as well.

Both these methods are widely used in literature and in HFS environment. Anyway both these methods have drawbacks. When scheduling has to be done, the objective that is to be optimized is very crucial for deciding which process job has the priority to be processed first than others and these urgent jobs, called “hot jobs” must be completed as soon as possible. That’s why they are placed in the left part of the solution. Yet this is only for stage 1 but makes no guarantees for the subsequent stages where jobs are queued by the FCFS rule. For this reason, it is very difficult to control the propagation of the schedule, especially when a higher number of jobs in the sequence is considered. That’s why this kind of approach leads to the difficult handling of urgent jobs. If in the scheduling decision process, one wants to precisely define when a job will start and when it will finish, this become very hard to do when number of operations and number of jobs increases. Such we call the **controllability problem**. On the contrary, this problem with PS this problem is not present because we schedule the jobs in each stage by the same sequence π. Yet, this leads to another problem: unnecessary machine idleness. More specifically, when the sequence that jobs exiting from the stage is different from π, to schedule the jobs at the stage *i* by sequence π we have to delay the starting time of some jobs, which may lead to unnecessary machine idleness. So idleness will be higher when the processing times of the operations of each jobs are variable from one stage to another and also between the jobs themselves. It ends up with less tightness of the schedule, given this approach so static. We call this the ***tightness* problem.** In this research, literature review was conducted also to better understand which decoding method was the better to use in the application of this algorithm. As shown in the table, a resume of part of the investigated paper has been summarized.

ID	YEAR	AUTHORS	PROBLEM	METHOD/TECHNIQUE	ENCOD	DECODING	MSR	OBJECTIVES
1	2003	Rajendran, C., Ziegler, H	flowshop with SDST	efficient heuristc	Job Permutation	PS and LS	no rule	∑wjTj, ∑wjCj
2	2005	Oguz and Ercan	SP with class of multiprocessor tasks	SA, TB, GA	Task mapping	LS	FAM	Cmax
3	2005	Ruiz, R., Maroto, C., Alcaraz, J.	flowshop with SDST	Advanced metaheuristic	Job Permutation	PS	ECT	Cmax
4	2011	Egin et al.	HFS scheduling problem	GA with multiprocessor task problems	direct encoding	PS	dpr	Cmax
5	2011	Karimi et al.	group scheduling in hybrid FFS	multi-phase approach (ICA)	Random key	not specified	FAM	Cmax and TWT
6	2012	Bozorgirad and Logendran	group scheduling SDST and UPM	MILP, meta-heuristics based on TS	Job Permutation	Dynamic Sched.	FAM	Fl (∑wjCj, ∑wjTj) (R, rj)
7	2013	Ciavotta, Minella, and Ruiz	PFS scheduling problem	RIPG	Job Permutation	PS and LS	no rule	C_{max}, ,TFT and C_{max}, ,TWT
8	2013	Lahimer, Lopez, Haouari	HFS scheduling problem	discrepancy search method	Job Permutation	LS	FAM	Cmax
9	2014	Asefi et al.	no-wait flexible flowshop SP	A hybrid NSGA-II and VNS	Job Permutation	LS	FAM	Cmax, ∑Tj
10	2015	Li and Pan	HFS SP with limited buffers	TABC	Job Permutation	LS	FAM	Cmax
11	2015	Fernandez-Viagas and Framinan	distributed PFS SP	Simple evolutionary algorithm	Job Permutation	PS	dpr	Cmax
12	2016	Tian, Li and Liu	Bi-objective HFS SP	PABOVNS	Random key	LS	FAM	Cmax, Ttot
13	2017	Ying et al., 2017)	Distributed no-idle PFS SP	Rated reference greedy algorithm	Job Permutation	PS	ECT	Cmax
14	2018	Sioud and Gagné	PFS with SDST	EMB optimization algorithm	Job Permutation	various	dpr	Cmax

Tabella 5 - Resume table per paper

Here some notation used in table 5:

MSR = Machine selection rule
UPM = unrelated parallel machines
SP = scheduling problem
PFS = Permutation flowshop
HFS = hybrid flowshop
SA = simulated annealing
TB = Tabu search
GA = Genetic alghorithm
RIPG = Restarted iterated pareto greedy
TABC: Tabu search bee colony algorithm
PABOVNS = A Pareto-Based Adaptive Variable Neighborhood Search
EMB = enanched migrating birds
LS = List Scheduling
PS = Permutation Scheduling
FAM = First available machine
ECT = earliest completion time
JP = job permutations

Tabella 6 -- Notation

This very small literature review, has been distinguished from the wider one conducted in chapter 2, because has the goal to identify some features especially concerning the encoding and decoding methods used.

As shown in the tables, different methods and approaches have been chosen, but encoding and decoding methods are essentially the same. In particular, job permutation represents the much used encoding method. As said before, this kind of indirect encoding is much more efficient. Indeed, no particular preference seems shown about the decoding method. Given that we are optimizing objectives like Total Tardiness and Total Setup Time, it is somehow logic to expect that, especially when the number of jobs is increasing, it can be much more important to have control on the jobs we are scheduling. Moreover, it is important to remember the assumption of unrelated parallel machines with machine eligibility. For completeness, also the machine selection criteria has been specified in the tables. It can be seen that the first

available machine rule is preferred, especially given that in most cases, the maximum makespan is the objective to minimize.

4.4 The restarted iterated pareto greedy

In the literature review, it has been already said like the Iterated Greedy procedure belogs to the class of the stochastic local search techniques (SLS). Now, similarly to [7], we propose a procedure named *Restarted Iterated Pareto Greedy* (RIPG). The logic behind this algorithm is very simple: a greedy multi-objective strategy is iteratively applied over a set of non-dominated solutions. The proposed RIPG is an extension of the above described IG. In fact, the main drawback of IG procedure is that they are prone to get stuck in local optimum solutions. The reason lies behind their very nature as they are greedy methods. RIPG is no different. To avoid this potential problem, we have included a simple, yet reliable restart phase. This procedure merely consists of storing all the elements of the current working set in a separate archive and then creating a new random working set of 100 elements. The main advantage of this restart procedure is that it is a very fast way to introduce diversification inside our metaheuristic scheme, whereas its main inconvenience consists of the difficulty in choosing of a suitable restarting criterion. To give a general scheme about how the algorithm is divided, it can be broken into five phases:

1) ***Initialization***. In this first phase, an initial set of good solutions is generated using two NEH heuristics [127], each one designed to attain good values for a specific criterion. The first one for the Total Tardiness objective (TT) and the second one for the minimization of the Total Setup Time (TST). After that, the remaining four phases are iteratively repeated and constitute the main loop of the algorithm.

2) ***Selection***. The second phase, chooses one solution from the current working set for the next phase. The procedure adopted to do this is the so called Modified Crowding Distance Assigned procedure (MCDA). This method was originally presented in [86] has been developed in order to carry out the selection process. At each solution is assigned a value (*Crowding Distance*) which depends on the normalized Euclidean distances between it and the solutions that precedent. The main difference between the classical one and this new modified version resides in the fact that the modified procedure considers the number of times each solution has been already selected in previous iterations (*Selection Counter*), and uses this information to calculate the Modified Crowding Distance (MCD). This modification prevents allocating computing resources to search the same regions. The element with the highest value of MCD is selected as the starting point for the Greedy or local search phases.

3) ***Greedy phase***. This phase represents the real core of the entire procedure. It is constituted by the two phases of *destruction* and *construction*. The destruction procedure is applied to a permutation π of n jobs and it chooses randomly, without repetition d jobs. These d jobs are then removed from π in the order in which they were chosen. The result of this procedure are two subsequences, the first being the partial sequence π_D with $n - d$ jobs, that is the sequence after the removal of d jobs, and the second being a sequence of d jobs, which we denote as π_R. π_R contains the jobs that have to be reinserted into π_D to yield a complete candidate solution in the order in which they were removed from π. The construction phase starts with subsequence π_D and performs d steps in which the jobs in π_R are reinserted into π_D.. This process is iterated until π_R is empty.

4) ***Local search***. This phase is applied usually after greedy phase, over a selected element of the current working set.

5) ***Restart*.** This is the last phase procedure is implemented to prevent the algorithm from getting stuck in local optima.

In *Figure 1* it is presented a general scheme of the procedure. The detailed procedure will be described later in chapter 4.

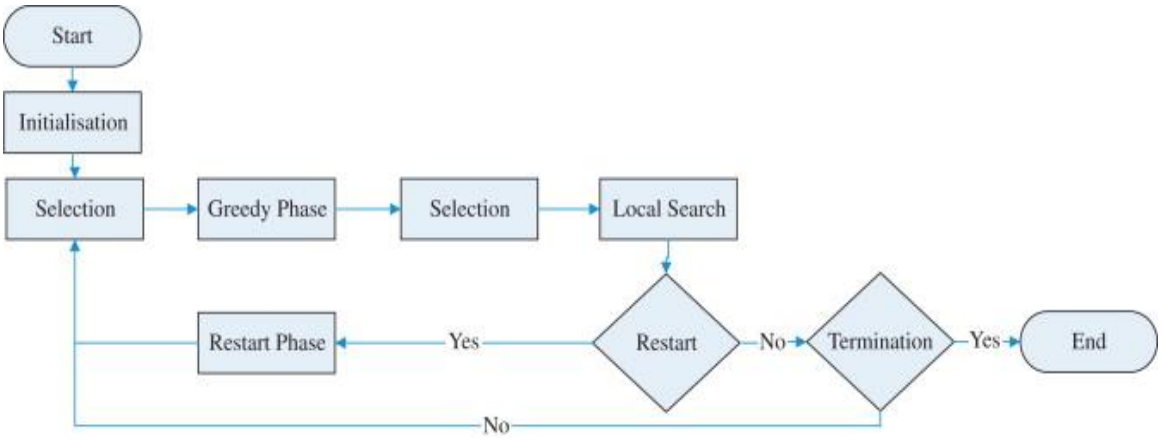


Figura 7 - RIPG scheme

4.4.1 The inizitalization phase

Conducetd experiments done in literature, clearly showed that a good initial working set greatly improves the quality of RIPG. This is certainly expected as it is also the case with the single-objective PFSP. In the first phase, an initial set of good solutions is generated using two well-known **NEH heuristics**, each one designed to attain good values for a specific criterion. In fact, it is intuitive that the generated initial solutions represent the starting point on which more complex elaborations had be done. From this perspective, the initial solutions play an underlying role in creating a high performing algorithm. By the way,

since our analysis focuses on reaching not one but two objectives, the choice in the adopted heuristics was due to choose heuristics the return of good enough solutions but that also ensure a sufficient level of diversity. Further considerations are needed. First, it is not sure that these initial solutions will be well spread in the Pareto front. Also, this algorithm methodology adopts the selection process and the greedy phase many times. This approach is capable of greatly improving solutions. Selecting only one of the initial solutions for the greedy phase could have a negative result: all other initial solutions could be dominated after this phase. As a result, there is loss of diversity and coverage in the Pareto front. So, discarding some initial solutions can be a mistake because you lose the possibility to go toward promising directions. That's why, in the first step of the RIPG, all initial solutions are processed by the greedy phase, without applying the selection operator, and for each one, a non-dominated set is obtained.

So summarizing, the results of this heuristics generates the *Initial Solution Set* (ISS), making use of two well-known NEH heuristics, will soon generate respectively two initial solutions, one optimizing the objective of the Total Tardiness (TT) and one optimizing the objective of Total Setup Time (TST). This is the starting point of the procedure, which will step by step be replaced by better solutions. In a first step, all initial solutions are processed by the **Greedy Phase** one by one. The obtained solutions of this process are added to the ISS and then, the dominated ones are removed and the initial *current working set (CWS)* is conformed. The word "dominated" refers to the concept of **Pareto dominance**. The concept of Pareto dominance is of extreme importance in multi-objective optimization, especially where some or all of the objectives and constraints are mutually conflicting. In such a case, there is no single point that yields the "best" value for all objectives and constraints. Instead, the best solutions, often called a Pareto or non-dominated set, are a group of solutions such that selecting any one of them in place of another will always sacrifice quality for at least one objective or constraint, while improving at least one other. In our case, the aim behind this policy, adopted in the initial phase, is

to avoid that a likely large improvement during the initial iterations might generate a set of solutions that dominate the remaining initial solutions, impoverishing the quality and diversity of the working set too early. At each iteration of the algorithm, the **selection phase** is applied. Its role is to point the search towards promising directions. Selection achieves this goal by choosing one solution from the *current working set* on the basis of considerations related to their quality. The detailed description of it, will be done later in the text. In this way, only those solutions that are more likely to increase the quality of the current working set will be kept, speeding up the whole search process. After this initialization, the working set of non-dominated solutions is ready for the main algorithm phases. The diagram and the pseudocode is in [APPENDIX - 8.1].

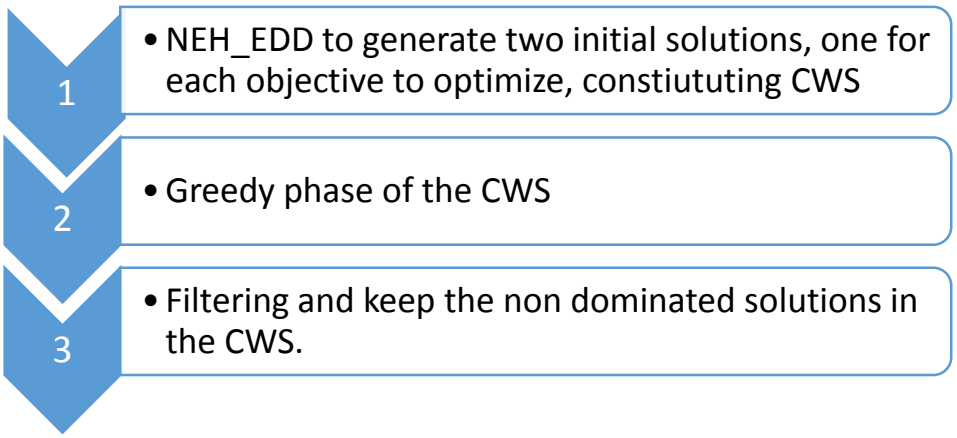


Figura 8 - Initialization procedure scheme

4.4.2 The selection phase

As already mentioned before, a selection phase is often applied before the greedy phase to choose which solutions are more convenient to elaborate, based on a criteria presented by [124], for the first time. The aim of the selection phases to select a candidate solution belonging to a less crowded region of the Pareto front and at the same time has already been selected a small number of times To do so, a modified version of the *Crowding Distance Assignment* (CDA) procedure, has been developed in order to carry out the selection process. The original CDA method divides the working set into dominance levels, i.e., the set of non-dominated solutions form the first-level Pareto front. Once we remove these elements, we have another non-dominated set of solutions, which correspond with the second-level Pareto front. This procedure is repeated until all solutions are assigned to a Pareto front. Here, we do not consider this distinction in levels, because it's not useful for our objectives. Afterwards, it assigns to each solution a value (*Crowding Distance*) dependent on the normalized Euclidean distances between it and the solutions that precede and follow it. Such technique favors the selection of the most isolated solutions of the first frontier This CDA will represent a sort of priority value and create a hierarchy inside the field of the solutions. Therefore, applying the standard Crowding Distance procedure results in an algorithm that gets easily stuck, as if no improvements are found after the greedy and local search phases, the Pareto fronts do not change and the same solution is selected repeatedly. To avoid this, we add a selection counter (*n_sel*) to each solution which counts the number of times each solution has been selected. This represents the main difference between the normal CDA procedure and the MCDA adopted here. Then, it uses this information to calculate the Modified Crowding Distance (MCD). At the end, the solution with the highest value of MCD is selected as the starting point for the Greedy or the local search phases. The use of such an operator demonstrated, in preliminary experiments, to significantly improve the Pareto front in terms of

quality and spread of its solutions. The diagram of this phase and the pseudocode are presented in [APPENDIX – 8.2] .

4.4.3 The greedy phase

This is the heart and most innovative part of the algorithm even though the structure of the original IG is still kept unchanged. However, there are some important differences between the greedy phase adopted in the first original IG procedure, and the one adopted here inside the RIPG. In particular, in the original procedure where only one partial solution is maintained and a NEH-like greedy heuristic is applied in one unique step at each iteration of the algorithm. The innovation here, is that, in RIPG, the Greedy phase becomes an iterative process, that works with a set of partial solutions and returns a set of non-dominated permutations. Let's go into the details; the Greedy phase can be basically divided into two steps: In the first one, called ***Destruction Phase***, a block of *d* consecutive elements is randomly removed from the MCDA-selected solution. This is the other important difference with the original GP because there the removal was not carried out by groups of elements. So, The *Destruction* step chooses a randomly a starting position k, and a block of *d* consecutive elements are removed from the selected solution. This k parameter is of great importance and will be one of the parameters that will be tuned afterwards in the calibration of the algorithm. The second step, called **Construction phase**, iteratively reconstructs the solution by reinserting, one by one, all the *d* removed elements into all possible positions of a group of partial solutions. This inserting scheme was already effectively used in [125] and it is possible thanks to the use of Pareto dominance. At each step, a new set of partial solutions is generated. More specifically, let *n* be the length of the initial solution and d the size of the block of removed elements. During the first iteration, the first of the *d* removed elements is inserted in all possible positions of the partial solution. This generates $(n - d + 1)$ new partial solutions. The next removed element, will

be inserted in all positions of all previous $(n-d+1)$ partial solutions, generating a new set of partial solutions of size $(n-d+1)\times(n-d+2)$ and so on. This process is repeated until the last removed element is inserted and a set of complete solutions is generated. At the end of this process the total number of generated complete solutions would be equal to $\prod_{i=1}^{d}(n-d+1)$. This defines an upper bound for the number of solutions generated by the greedy phase of the algorithm, that is:

$$\prod_{i=1}^{d}(n-d+i) \geq \prod_{i=1}^{d}(n-d)+\prod_{i=1}^{d} i = (n-d)^d + d!$$

Regardless of this, the bound is very far from being tight because, at each iteration, all the dominated incomplete sequences are removed. And this is actually the main drawback of this procedure. When *d* values become large, the size of the partial solutions grows exponentially. For example, for *n*=20 and *d*=5 the number of complete final solutions would be more than 1.860.000. To overcome this problem, each time a set of partial solutions is generated, only the non-dominated partial solutions are kept and the dominated ones are discarded. Actually, the *Construction* step is a variation of the NEH insertion scheme used in the initialization phase. The main difference from that heuristic is in the use of Pareto dominance to maintain not just one incomplete partial solution at each iteration (as in NEH), but a whole set of non-dominated partial solutions generated during the insertion process. These solutions are added to the current working set, and then the MCD selection procedure is applied. In this way, a solution is selected to be processed by **the *local search* phase**, which is explained next. This greedy phase is precisely described in the pseudo-code [APPENDIX -8.3]

4.4.4 The local search phase

In order to maintain the algorithm as simple and fast as possible we focused our effort in obtaining a simple and fast local search procedure, This phase has been demonstrated to be very helpful in improving the quality of solutions in the single as well as in the multi-objective cases. Also here, as it has been for all other phases until now, there are some differences between the original Ig and the RIPG. It is important to remember that the Local search procedure allows only one sequence at a time as input. In the original IG, the local search procedure uses as input the outcome of the greedy phase. For the multi-objective case, the greedy phase returns a set of non-dominated elements, which in most of the cases are made by more than one solutions. So it is intuitive that, to better choose which of the current working set element must undergo the local search, the previously described selection process is performed after the greedy phase and the solution with the highest MCD value is chosen and processed by the local search phase. In order to maintain the algorithm as simple and fast as possible we focused our effort in maintaining the structure of the local search much simple and fast as possible. Here it is a detailed description: n_{sel} elements belonging to the selected solution are randomly chosen, removed and re-inserted into n_{neigh} consecutive positions, half of which usually precede and half follow the original position of the element. The symmetricity of the neighborhood with respect of the original selected position is not guaranteed because it strictly depends on the distance of the original position from the beginning or from the end of the sequence. Local search in a multi-objective setting is not as simple as one might think. The above procedure is repeated *Selection Counter* (n_{count}) times. This is because if a solution has been selected previously, its closest neighborhood has been already explored. In the hope of improving the selected solution further, a deeper local search has to be carried out. An upper bound is imposed to the number of removed elements. To further speed up this local search, we employ the well-known accelerations of [126]. Afterwards, Pareto dominance is checked and a final non-dominated set is

generated as a result. During the initial design phase a decision had to be taken for which were the most suitable values to assign to n_{sel} and n_{neigh}. According to [7] which already used this two parameter in their experiments, the best values were n_{neigh} = 5 and n_{sel} dynamically changing according to the value of n_{count} as explained in the following formula:

$$n_{sel} = \begin{cases} n_{count} & n_{count} \leq n/2 \\ n/2 & otherwise \end{cases}$$

However, we decided to keep the value of n_{neigh} as this one decided by the author, and test the variation of the performance of the algorithm in the phase of Calibration that will be treated later. The pseudocode of the local search procedure is presented [APPENDIX – 8.4]:

4.4.5 The restart phase

The last phase of the RIPG procedure is the restart phase. As we already mentioned, this whole algorithm, made up by a very large number of greedy phase followed by selections has the tendency to choose a certain direction and this can bring to a situation of stuck in local optimum points: this is the one main drawback of original IG methods. This is actually a problem that is present in all the greedy methods, because it intrinsically depends from its nature and of course RIPG is no different. To avoid this potential problem, we have included a simple, yet reliable restart phase. As one can intuitively understand, the idea is to set a certain condition that, if verified, certify that our algorithm is in a situation of stuck and so need to be restarted with the addition of some variability. This procedure merely consists of storing all the elements of the current working set in a separate archive and then creating a new random working set of 100 elements. This is the simplest possible restart scheme that still allows the algorithm to escape from a situation in which the current working set is stalled. This procedure has one main advantage and one

main disadvantage. The main advantage is that it is a very fast way to introduce diversification which opens new promising directions and also add necessary diversification. Its main inconvenience consists of the difficulty in choosing a suitable restart criterion, which is far from being a trivial decision. The general idea, also proposed in [7], was to execute a restart when the working set has not been changed during a sufficiently large number of iterations. Yet, this do not solve our decisional problem because in this case, we'll have to choose a reasonable limit number of iterations and so to understand when a working can be considered not changed. Initial tests and calibrations done by [7] results in choosing the maximum number of iterations as $n \times 2$ according to the size of the input instance. However this strategy is sometimes inaccurate because it cannot detect a change in the working set that does not affect its cardinality, yet it is very simple and fast. Anyway, we need also to consider the right lower bound for this number that is, be also careful that the minimum number of iterations does not generate too much restart procedures, otherwise the algorithm lose consistency. In this way, it could prevents reaching of a steady state condition in the search or being too seldom applied, it could waste valuable CPU time. Based on these considerations, we however decided to not use the same criterion as done in [7].

The **termination criteria** is applied when a maximum number of decodings is reached. This budget is calculated by the following formula:

$$MaxDecBudget = \frac{150.000\ x\ n_{jobs}}{100} + 5.000[ms]$$

Chapter 5

[5] Numerical results

5.1 Benchmark description

The aim of this section is to describe how numerical analysis has been conducted and the criteria adopted to select the most convenient benchmark for the calibration of the RIPG and the comparison with a competitor algorithm considered for our specific problem. In our case we will later compare the performances of RIPG with the ones of the NGSA-II.

It is to be noted that different test benchmarks have been adopted for the calibration phase and for the comparison. In fact, the RIPG computational time of analysis rises exponentially when a large number of jobs and stages is considered. Moreover, it is even rare that in the industrial field of application of this algorithm, a very high number of stages is considered. That's why, for these practical reasons, a **10 instance file** for the calibration of the algorithm has been generated, each with a number of jobs chosen randomly among among {15, 30, 45, 60}, random number of stages among {4, 8, 12} and a random number of machines (with at least one machine per stage).

The obtained test benchmark is summarized in the following table:

#instance	N_jobs (n)	N_stages (m)	N_machines (l)
1	60	4	13
2	45	4	14
3	30	4	16
4	15	12	39
5	45	4	12
6	30	12	35
7	60	8	21
8	45	12	38
9	15	8	23
10	30	4	10

Tabella 7 - benchmark description

As shown in the table, an instance is defined by 3 parameters:

n and *m* indicate the number of jobs and number of stages, respectively. *L* indicates the number of machines. The operation processing times are generated with the following criteria: the common pattern is to generate the processing time for each operation independently using a uniform distribution.

For any instance, the job release dates are set to 0, and the due dates are generated using the method of (Choi et al., 2005). Given two parameters *TF* and *DR* called tardiness factor and due date range respectively, the due date is calculated by:

$$DU(P\left(1-TF-\frac{DR}{2}\right),(P\left(1-TF+\frac{DR}{2}\right))$$

where *P* is a lower bound on the makespan.

We set TF=0.1, DR=0.8 for a proper simulation of real situations. In each stage the number of parallel machines is *DU*(2, 4). For each job, not all but at least one machine in a stage is eligible to process it. The probability that a machine is not eligible to process a job is set as 20%.

The sequence-dependent setup times between jobs on all machines are random integers sampled from [1,99].

5.2 RIPG calibration

With a full factorial experimental design, the three factors to calibrate in the RIPG algorithm are the following:

- **"*k*" parameter**: as explained in chapter 4, this value is involved in the *Destruction* step of the Greedy phase. It represents starting position of the job

sequence where the block of d consecutive elements is removed. Potentially this value could be randomly generated in the interval between 1 and the length of the job sequence, but in these extreme cases, the functionality of the greedy phase is lost. So, to ensure at the same time good functionality and enough variability of the returned solutions, some reasonable values for the calibration of the *k* parameter are chosen as ***k* = {3, 5, 7};**

- **"*n_neigh*" parameter**: this parameter stands for "number of neighborhoods" and is involved in the Local Search phase where *n_sel* elements belonging to the selected solution are randomly chosen and reinserted into *n_neigh* consecutive positions, half of which usually precede and half usually follows the original position of the removed element. As also suggested in Ciavotta, Minella and Ruiz 2013, where n_neigh = 5 is indicated as optimal parameter, we decide to calibrate the algorithm also testing n_neigh = 3 and n_neigh = 7. So the values chosen for the calibration are **n_neigh= {3, 5, 7}**;
- **"dec_fun":** it stands for decoding function. We have to choose among Permutation Scheduling (PS) and List Scheduling (LS). So we have that **dec_fun = {PS, LS}.** From the literature review we have seen that probably for the characteristics of our problem,

So summarizing with a full factorial design of experiments (DoE), the total number of tests is:

$$10\ instances\ \times 5\ replications\ \times 2\ dec_{fun}\ \times 3\ n_{neigh} \times 3\ k = 900$$

To evaluate the performance of the algorithm the Hypervolume indicator has been considered. The **hypervolume indicator** is a set measure used in evolutionary multiobjective optimization to evaluate the performance of search algorithms and to guide the search. The hypervolume indicator, first introduced by Zitzler et al. as the 'size of the space covered', is used in many cases as the underlying indicator function. Up to now, it is the only known indicator that is compliant with the concept of Pareto-dominance, i.e., whenever a set of solutions dominates another set, its hypervolume indicator

value is higher than the one of the latter. A theoretical understanding why hypervolume-based algorithms outperform their Pareto-dominance based counterparts is still missing.

Classical definitions of the hypervolume indicator, also known as Lebesgue measure or S-metric are based on volumes of polytopes [15] or hypercubes [6] and assume that Pareto dominance is the underlying preference relation. Without loss of generality, we assume that k objective functions f = (f1, . . . , fk) that map solutions x ∈ X from the decision space X to an objective vector f(x) = (f1(x), . . . , fk(x)) ⊆ Rk have to be maximized. The goal for hypervolume-based algorithms is to maximize the hypervolume indicator IH. The hypervolume indicator $I_H(A)$ of a solution set A ⊆ X can be defined as the hypervolume of the space that is dominated by the set A and is bounded by a reference point r = (r1, . . . , rk) ∈ Rk :

$$I_H(A) = \lambda[\cup_{a \in A} (f_1(a), r_1) \times (f_2(a), r_2) \times \ldots \times (f_k(a), r_k)]$$

where λ(S) is the Lebesgue measure of a set S and [f1(a), r1] × [f2(a), r2] × · · · × [fk(a), rk] is the k-dimensional hypercuboid consisting of all points that are weakly dominated by the point a but not weakly dominated by the reference point. Note that the hypervolume indicator is Pareto-dominance compliant. Fixing the maximal number μ of solutions in an evolutionary algorithm A, the goal of maximizing the hypervolume indicator changes to finding a set of μ solutions that have the maximal hypervolume indicator value among all sets of μ solutions. Given one or more fronts and a reference point, it measures the volume of the space region between them. In our case, being the objective functions the minimizations of Total setup time and Total tardiness, the reference point chosen for each instance is represented by the **Nadir Point** of

the fits, consisting in the maximum value for each of the objective functions. The following figure gives a simplified yet practical representation of what stated before. In the figure f1 and f2 are the objective function values.

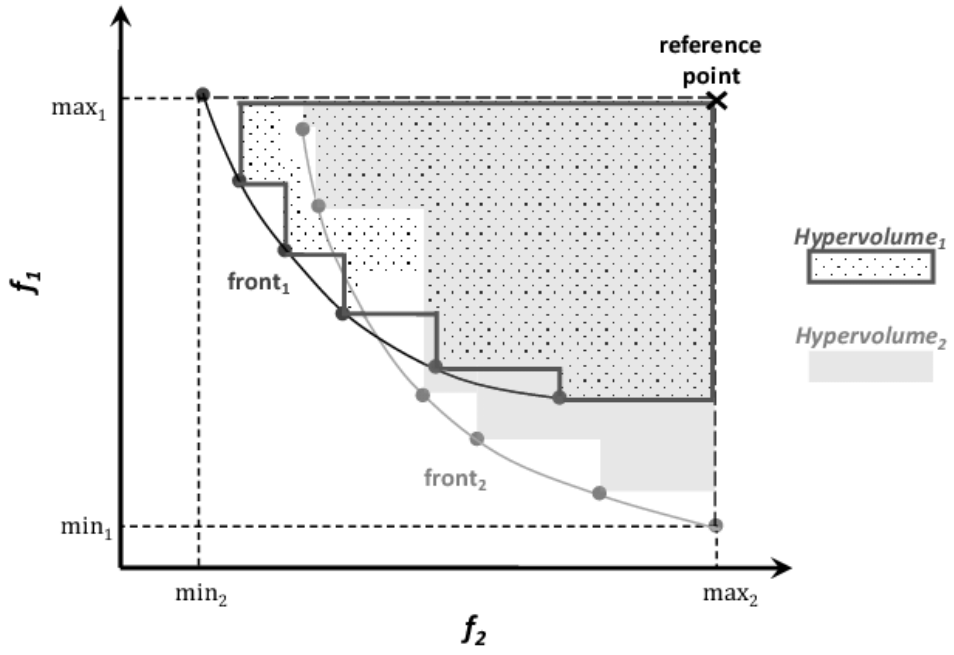


Figura 9- Hypervolume indicator

.

To compare different algorithms, the common measure for makespan criteria is the relative percentage increase (RPI), as in Pan and Ruiz (2012). Yet for tardiness criteria, RPI is no longer adaptable because it may provide a division by zero when the schedule has no tardy jobs (Naderi et al., 2009). For this reason we use the **relative deviation index (RDI)** as the response variable, which is defined as follow:

$$RDI = \frac{Alg_{sol} - Min_{sol}}{Max_{sol} - Min_{sol}} \times 100$$

where Alg_{sol} is the objective value of the current algorithm on the given instance, Max_{sol} and Min_{sol} are the worst and best objective value obtained by any of the algorithms in the comparison, respectively. Specially, in the case

that the Max_{sol} and Min_{sol} are equal to each other, the RDI will be 0 for all the algorithms.

The experiments are implemented in Matlab 2018a on a PC with Intel® Core™ i7 -2670QM CPU @ 2.2 GHz 8GB of RAM. To increase algorithm running speed, all decoding methods (which accounts for more than 90% of algorithm running time) are converted to C++ codes and called in Matlab environment.

The results of calibration experiments are analyzed by ANOVA. The ANOVA has been performed using MINITAB and choosing a significance level of 0.05. First the main effect plot shows which of the factors is likely to be significant. As shown here, the **decoding type** seems to have large influence on the performance of RIPG, while k and n_neigh seems not to be so relevant.

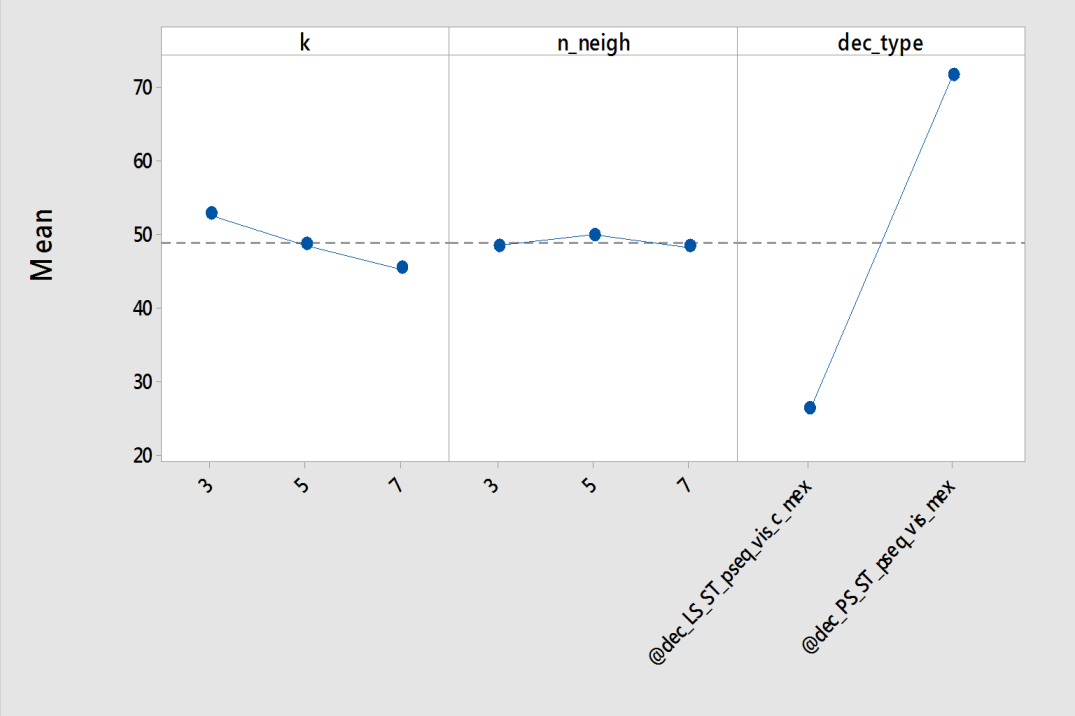


Figura 10- Main effect plot of calibration

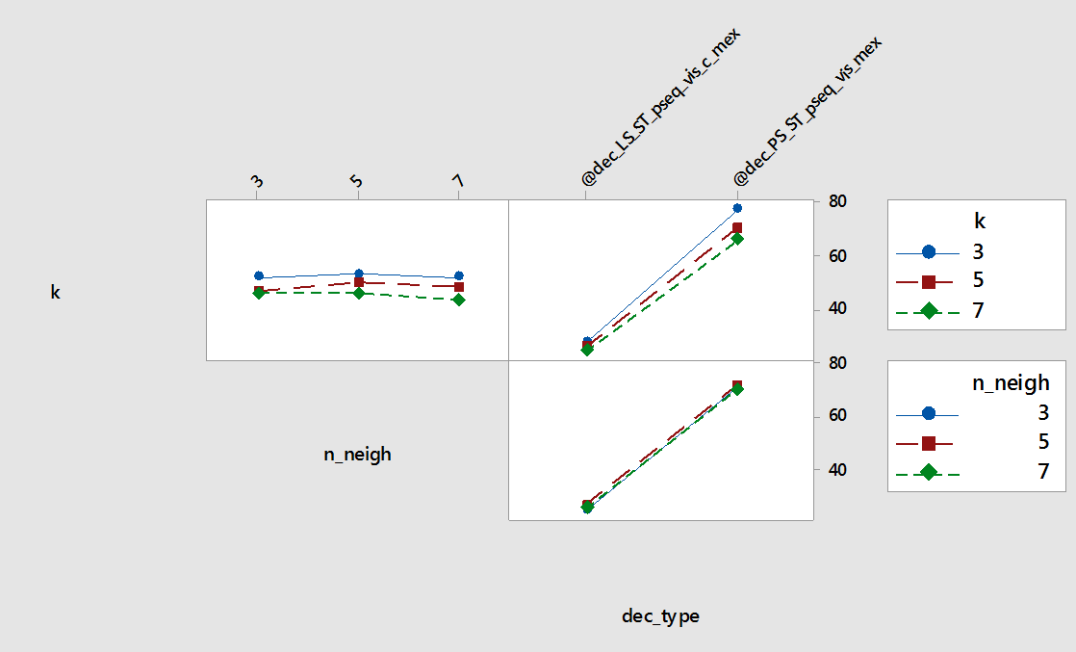


Figura 11- Interaction plot

Ciavotta, Minella and Ruiz 2013, as reference of our RIPG, used n_neigh =5 and k =5 based on their analisys. We made calibration in their neighbourhood, testing also 3 and 7 as suitable values The **interaction plot** doesn't show particular interactions. In almost every case lines seems to be quite parallel. Let us see now the effective **ANOVA table** for the model:

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
14,7565	71,12%	70,70%	70,20%

Factor Information

Factor	Type	Levels	Values
k	Fixed	3	3; 5; 7
n_neigh	Fixed	3	3; 5; 7
dec_type	Fixed	2	@dec_LS_ST_pseq_vis_c_mex; @dec_PS_ST_pseq_vis_mex

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
k	2	8431	4216	19,36	0,000
n_neigh	2	487	243	1,12	0,327
dec_type	1	462841	462841	2125,51	0,000
k*n_neigh	4	449	112	0,52	0,724
k*dec_type	2	2815	1407	6,46	0,002
n_neigh*dec_type	2	95	48	0,22	0,804
Error	886	192931	218		
Lack-of-Fit	4	615	154	0,71	0,588
Pure Error	882	192316	218		
Total	899	668049			

Tabella 8- ANOVA of calibration

Two considerations can be done here:

- **k and dec_type** are relevant and also their interaction.
- **R-sq** is high enough to consider a good reliability of the model

To verify if the reliability of the model is satisfied, we need also to verify the

hypothesis of normality of the standard residuals (SRES), their interval and the hypothesis of equal variance. The obtained plots are the following:

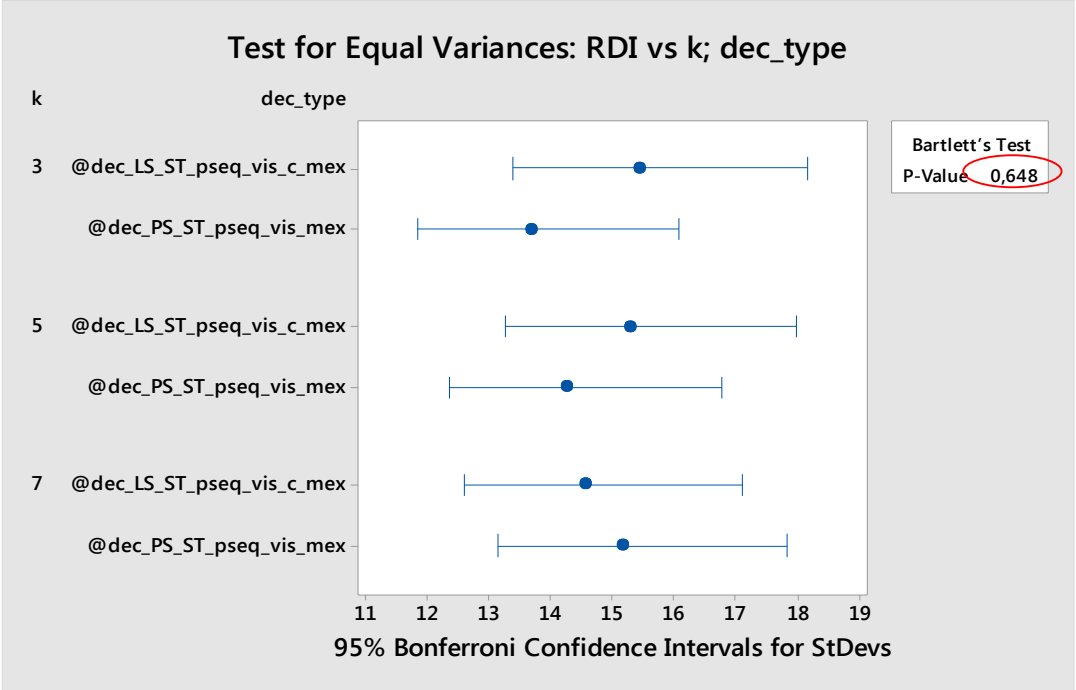


Figura 12- Test for equal variances

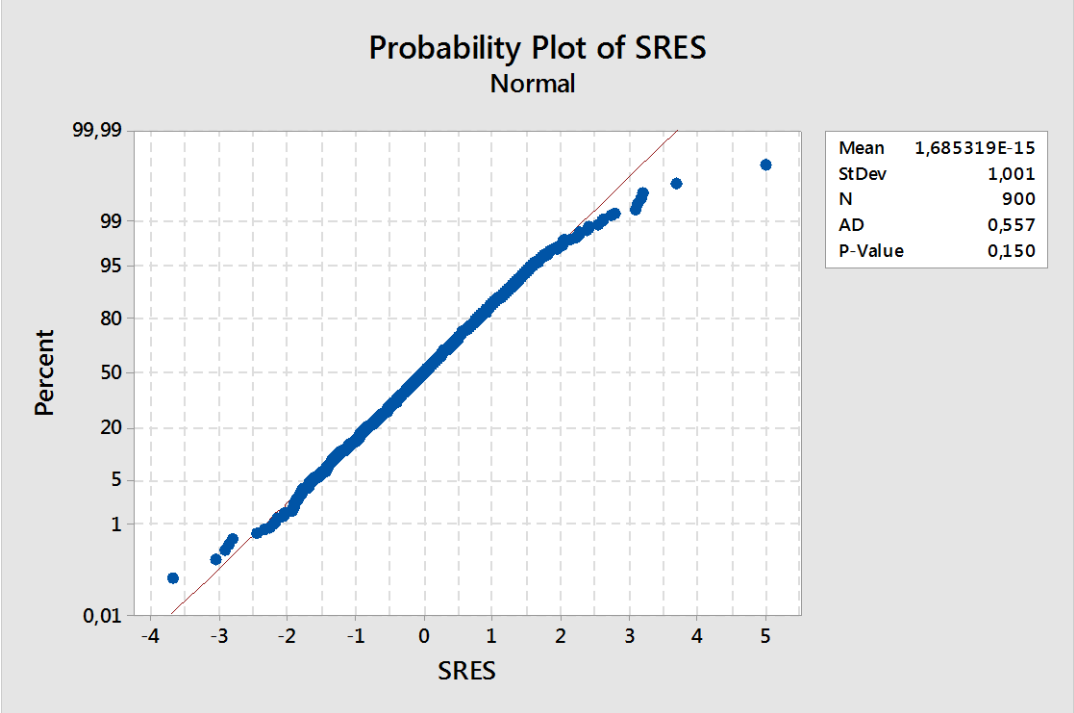


Figura 5- Normality test

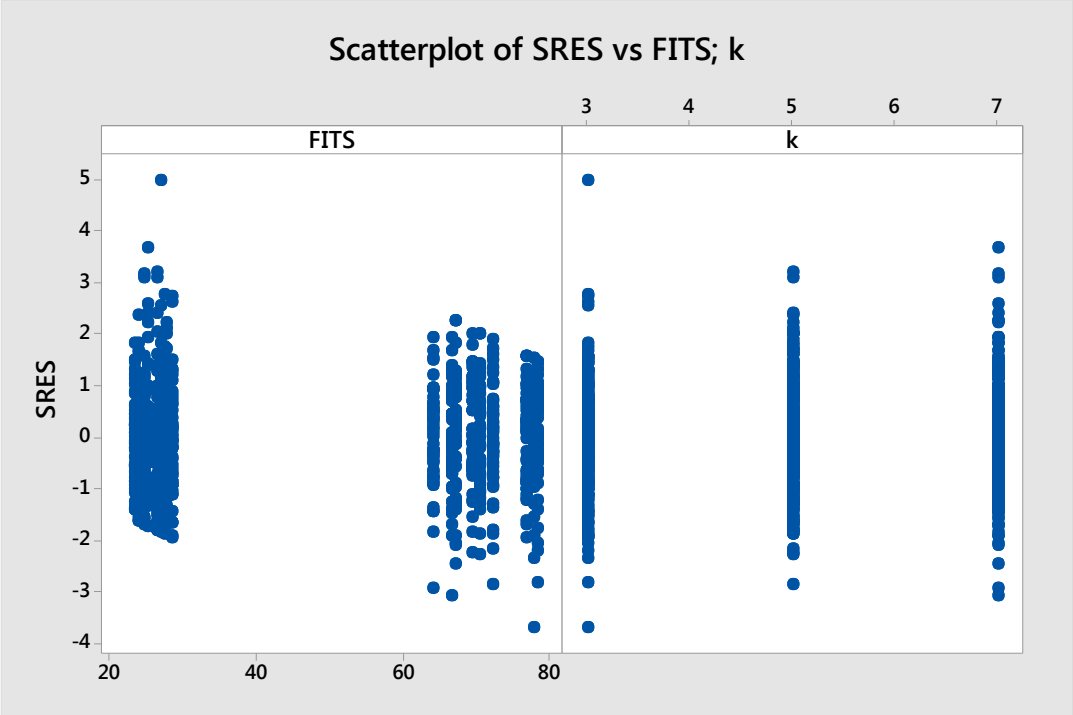


Figura 14- Scatterplot interval for SRES

From these plots we can say that:

- Normality of SRES is accepted;
- Scatterplot of SRES is almost concentrated between [+3; -3];
- The hypotesis of equal variances is accepted.

So the model is reliable and we can perform the **Tukey test** using a confidence interval of 95% **(CI=95%).**

Tukey Pairwise Comparisons: k

Grouping Information Using the Tukey Method and 95% Confidence

k	N	Mean	Grouping
3	300	52,7015	A
5	300	48,4559	B
7	300	45,2272	C

Means that do not share a letter are significantly different.

Tukey Pairwise Comparisons: dec_type

Grouping Information Using the Tukey Method and 95% Confidence

dec_type	N	Mean	Grouping
@dec_PS_ST_pseq_vis_mex	450	71,4723	A
@dec_LS_ST_pseq_vis_c_mex	450	26,1174	B

Means that do not share a letter are significantly different.

Tabella 9- Tukey test

From the ANOVA analysis, we can conclude that:

- As decoding type, **Permutation scheduling (PS) is always preferable,** independently by the choice of other parameters.
- **N_neigh is not significant**, so we keep the same value n_neigh=5, as (*Ciavotta, Minellla and Ruiz, 2013*).
- **The k=3 is better** than k=5, which is better than k=7.

So, for further analysis of comparison with NGSA-II, we will use the results of this section so to use the better performances obtainable from our RIPG.

5.3 Performance comparison with NGSA-II

For the comparison between the RIPG and NSGA-II, a set of 9 instances has been selected. Each instance has 10 cases. The instances are organized in

blocks of 3, each respectively with n_jobs= 20, 50, 100 and m_stages = 5,10, 20.

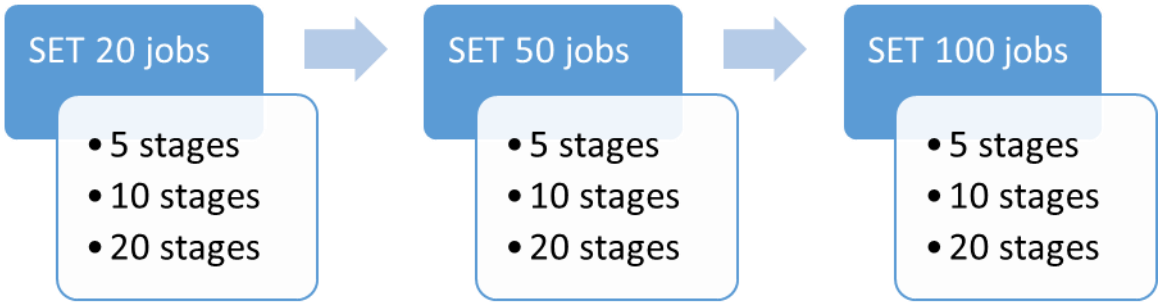
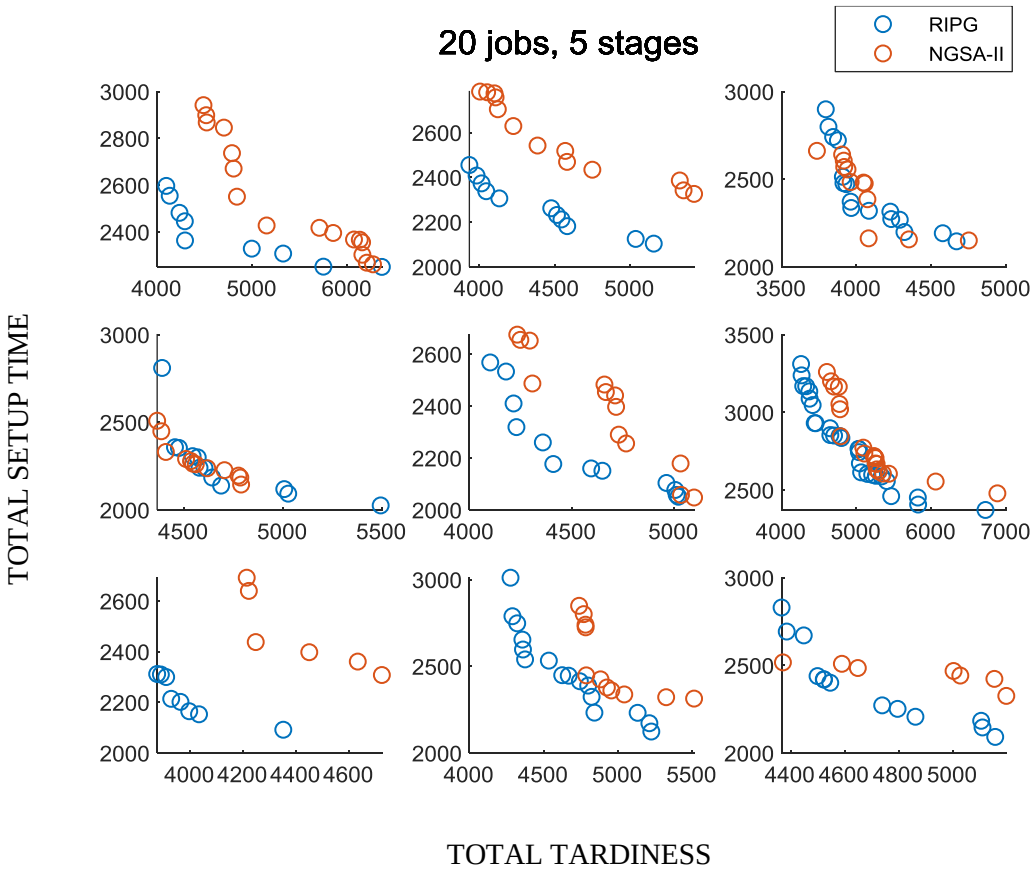


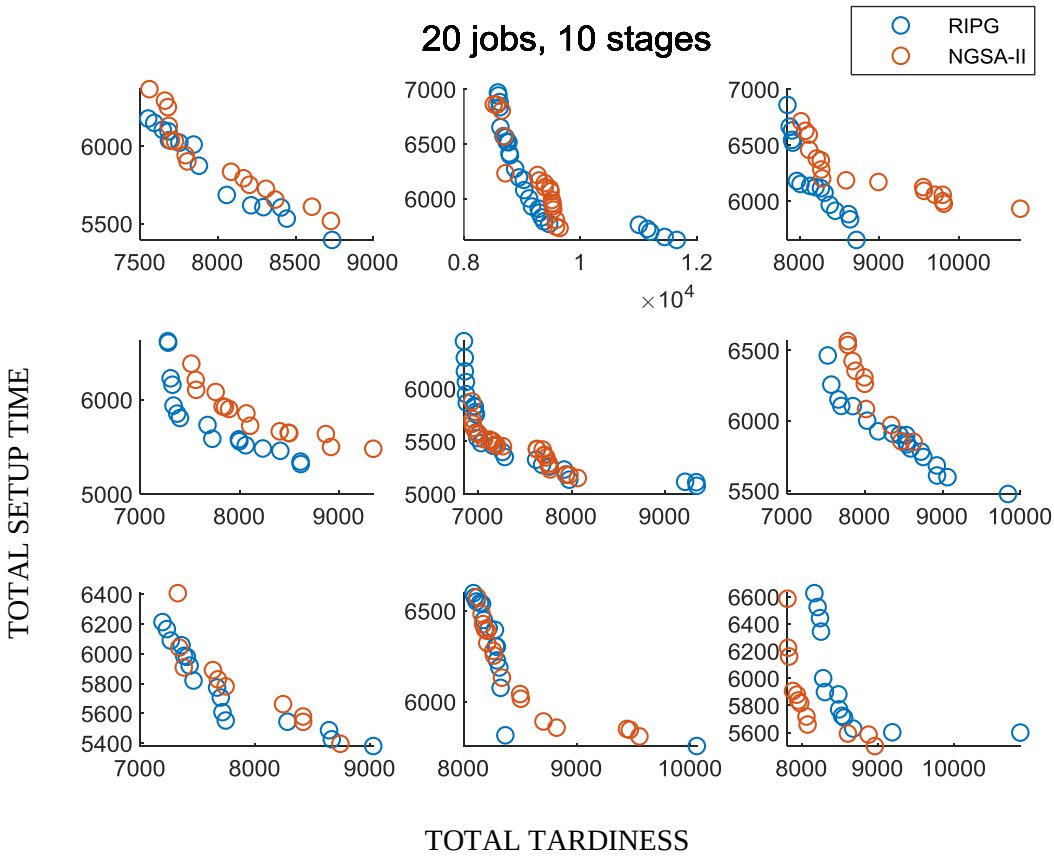
Figura 15 - Tests for comparison

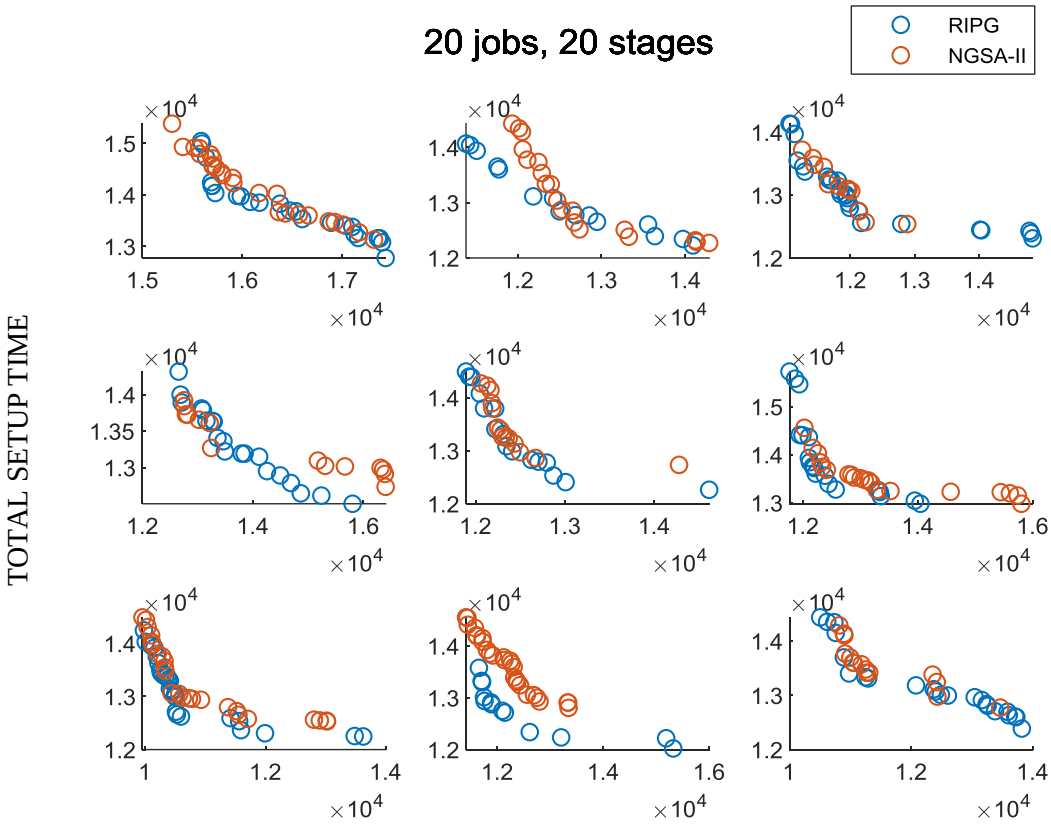
As before, we have:

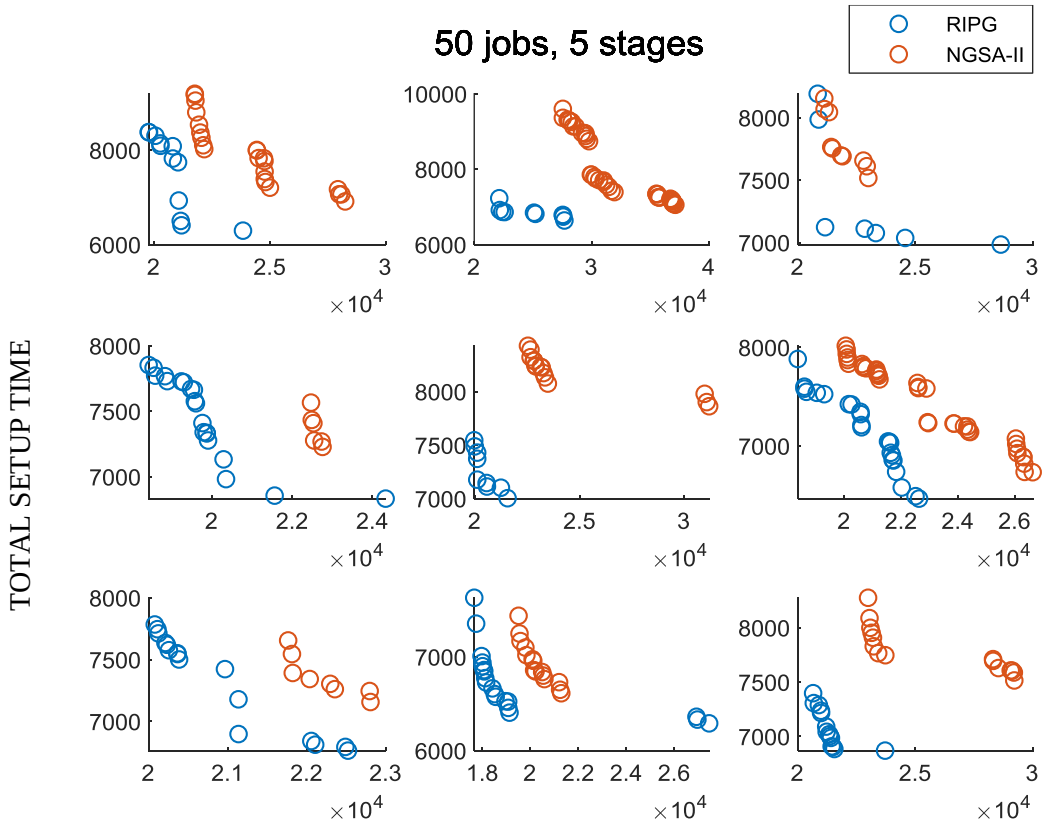
$$\mathbf{9}\ \boldsymbol{instances}\ \times \mathbf{10}\ \boldsymbol{cases}\ \times \mathbf{5}\ \boldsymbol{replications}\ \times \mathbf{2}\ \boldsymbol{algorithm}$$
$$= \mathbf{900}\ \boldsymbol{total\ cases}$$

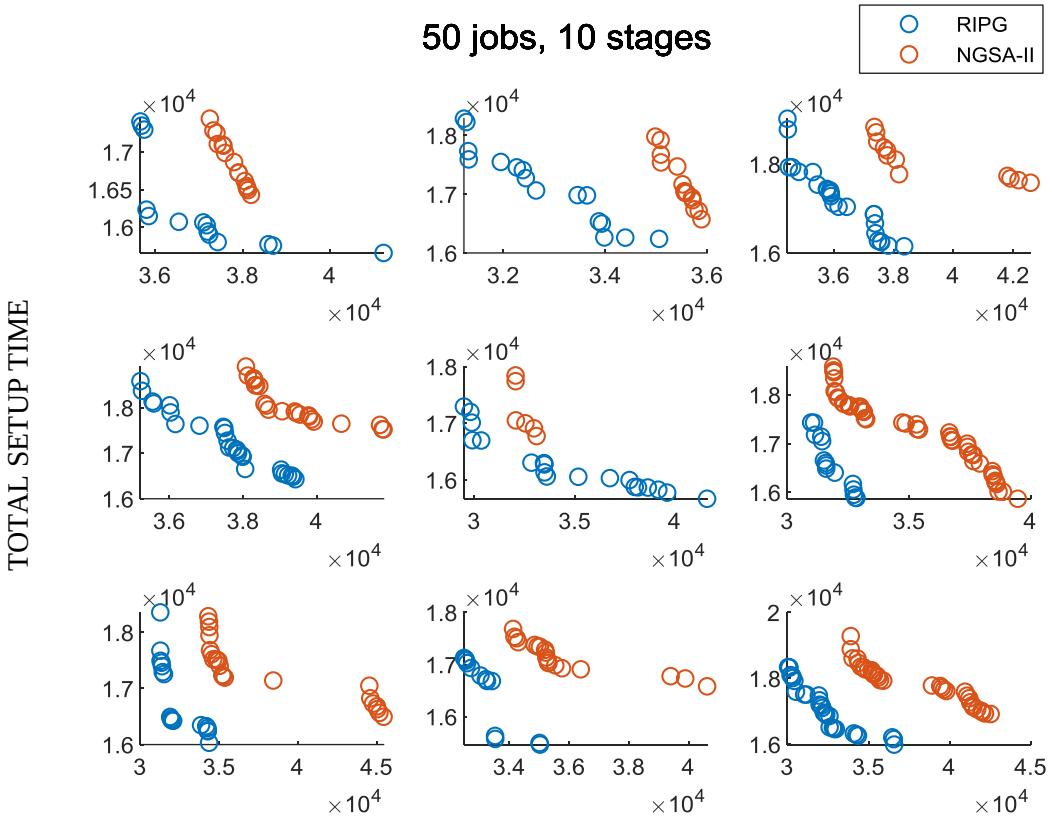
For a preliminary idea of the performance of the algorithms a plot of the obtained fronts is showed for each combination of jobs and stages. Later, results and interpretations of these graphs will be discussed.

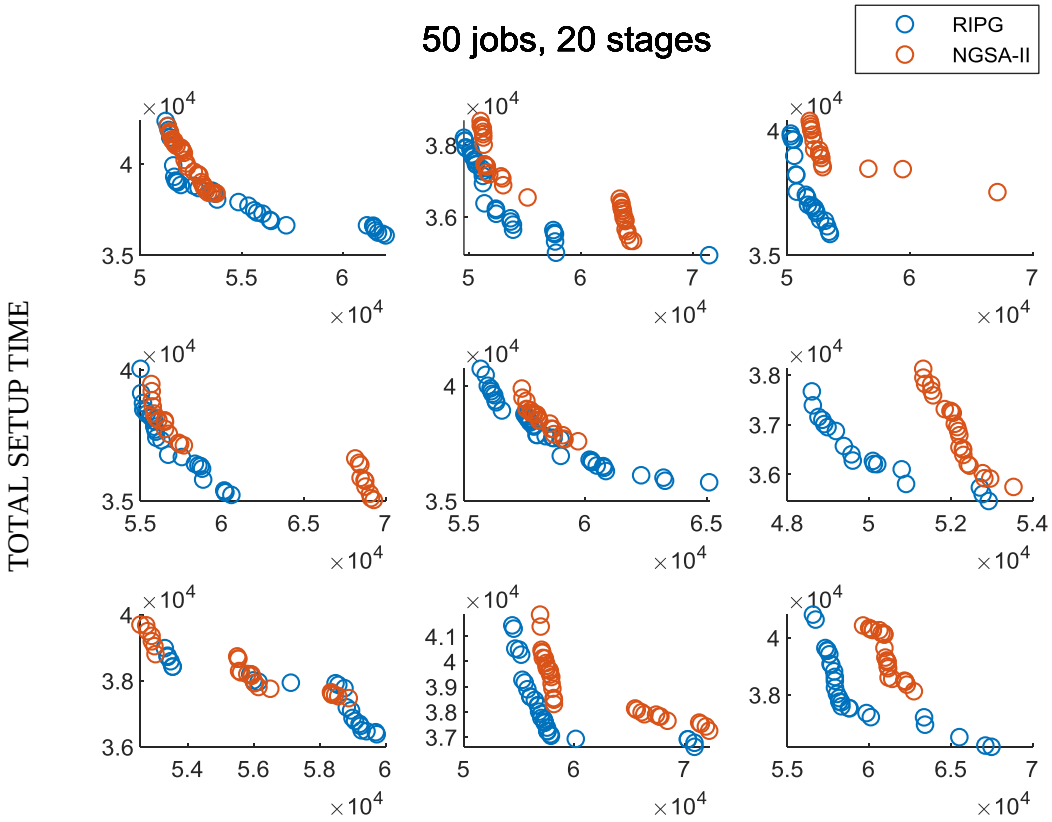

20 jobs, 5 stages
RIPG
NGSA-II
TOTAL SETUP TIME
TOTAL TARDINESS

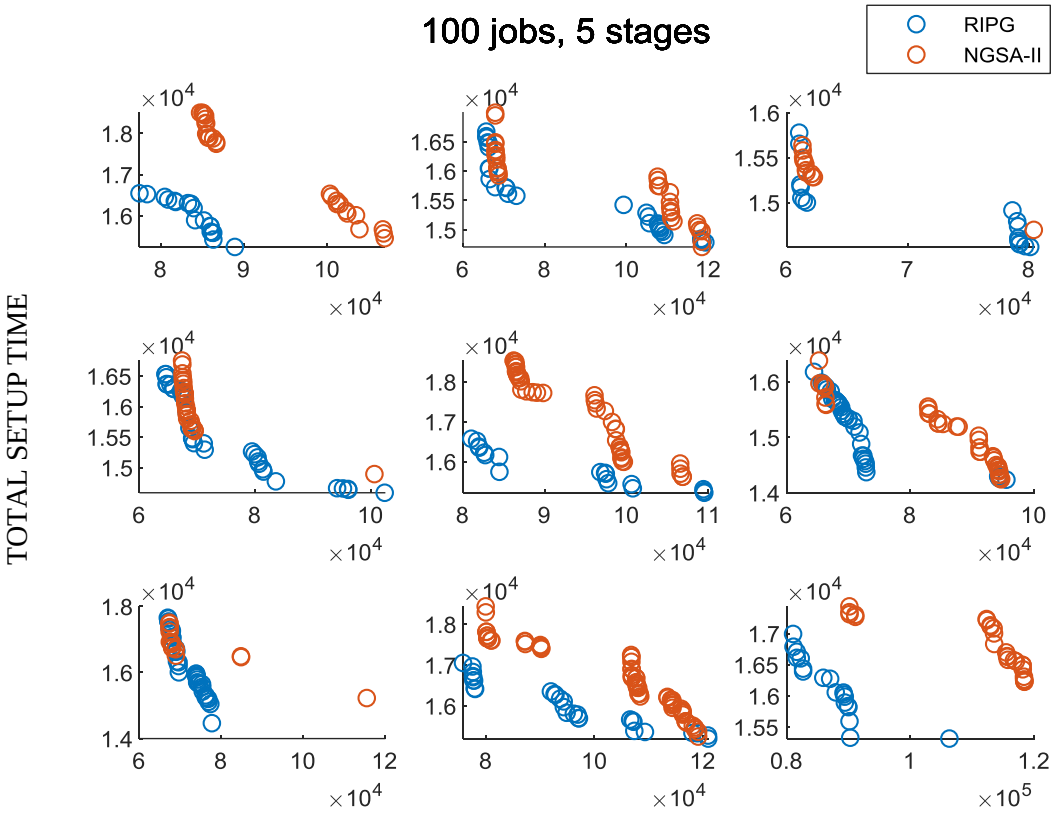

20 jobs, 10 stages
RIPG
NGSA-II
TOTAL SETUP TIME
TOTAL TARDINESS

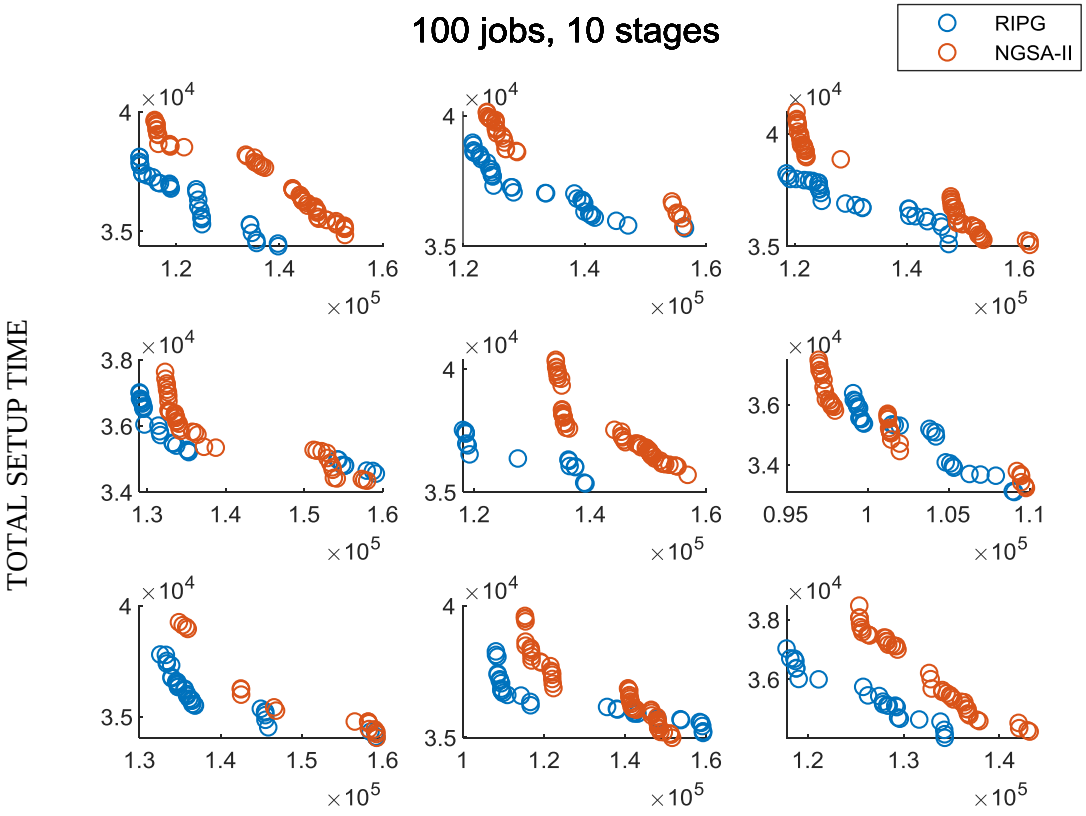

20 jobs, 20 stages
RIPG
NGSA-II
TOTAL SETUP TIME
TOTAL TARDINESS


50 jobs, 5 stages
RIPG
NGSA-II
TOTAL SETUP TIME
TOTAL TARDINESS


50 jobs, 10 stages
RIPG
NGSA-II
TOTAL SETUP TIME
TOTAL TARDINESS


50 jobs, 20 stages
RIPG
NGSA-II
TOTAL SETUP TIME
TOTAL TARDINESS


100 jobs, 5 stages
RIPG
NGSA-II
TOTAL SETUP TIME
TOTAL TARDINESS


100 jobs, 10 stages
RIPG
NGSA-II
TOTAL SETUP TIME
TOTAL TARDINESS

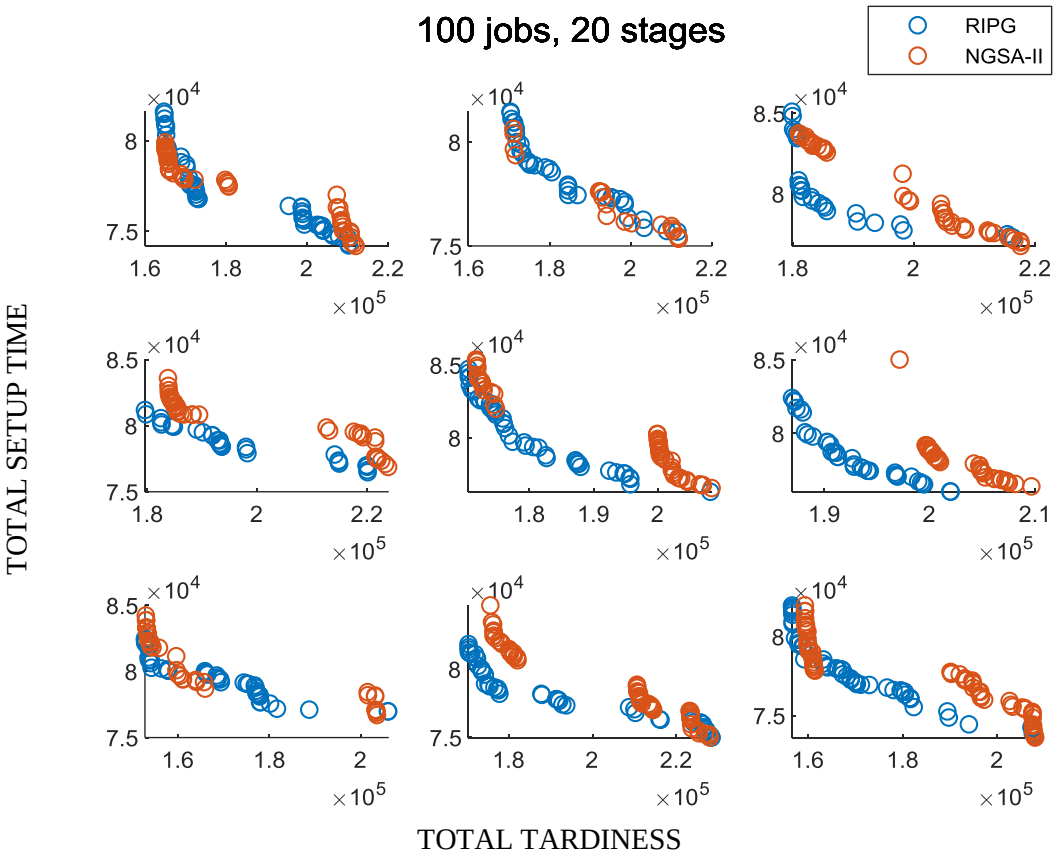


Figura 16 - Fronts comparison

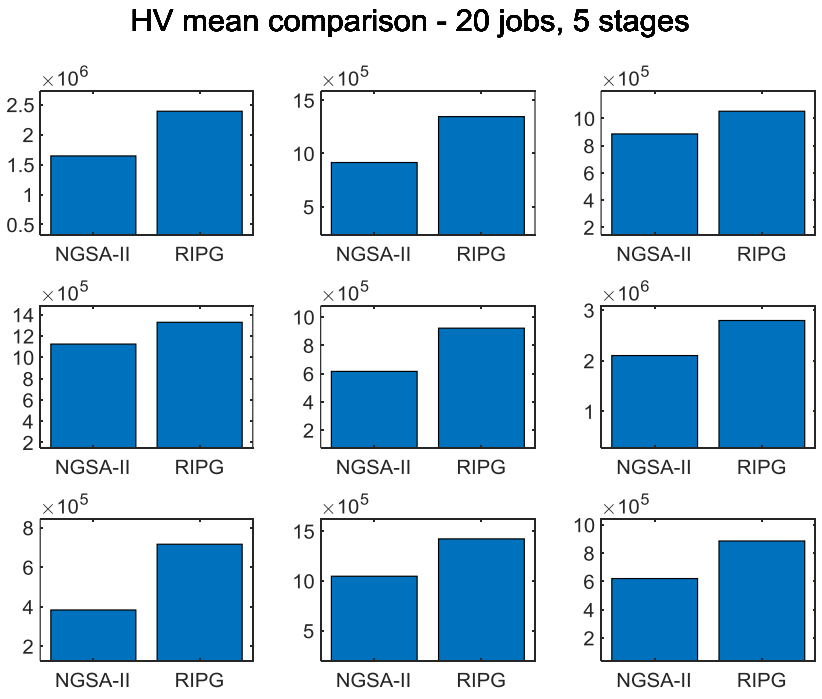


Figura 17 - HV means comparison 20 jobs 5 stages

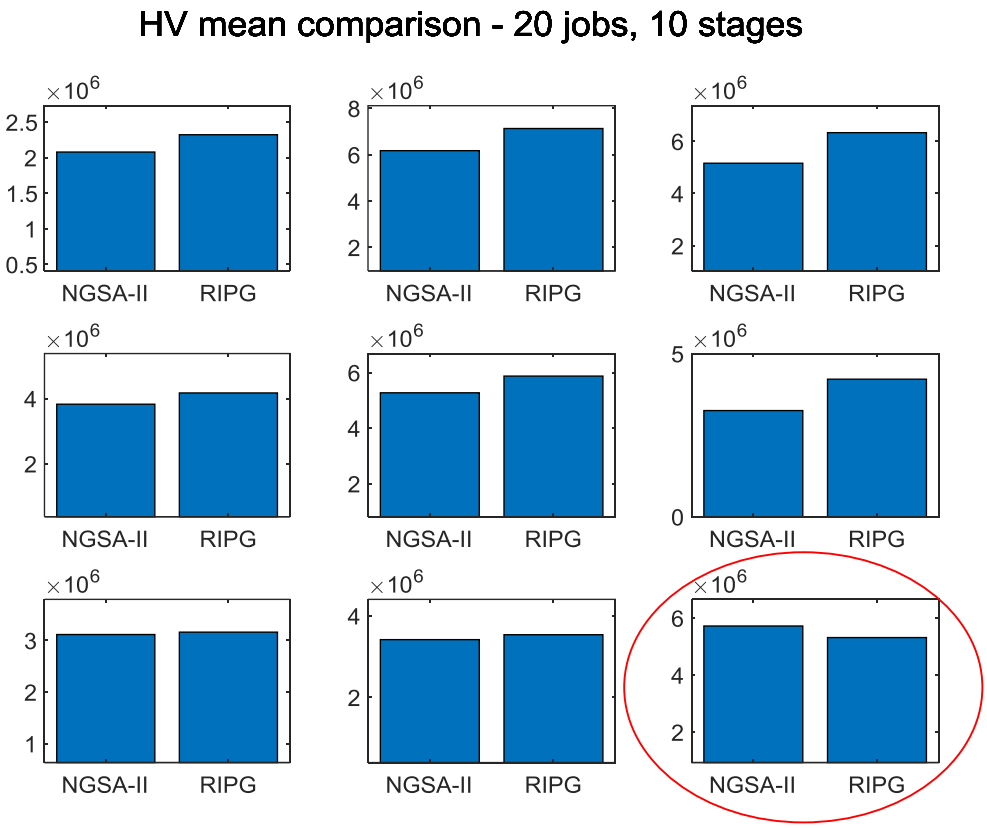


Figura 18 - HV means comparison 20 jobs 10 stages

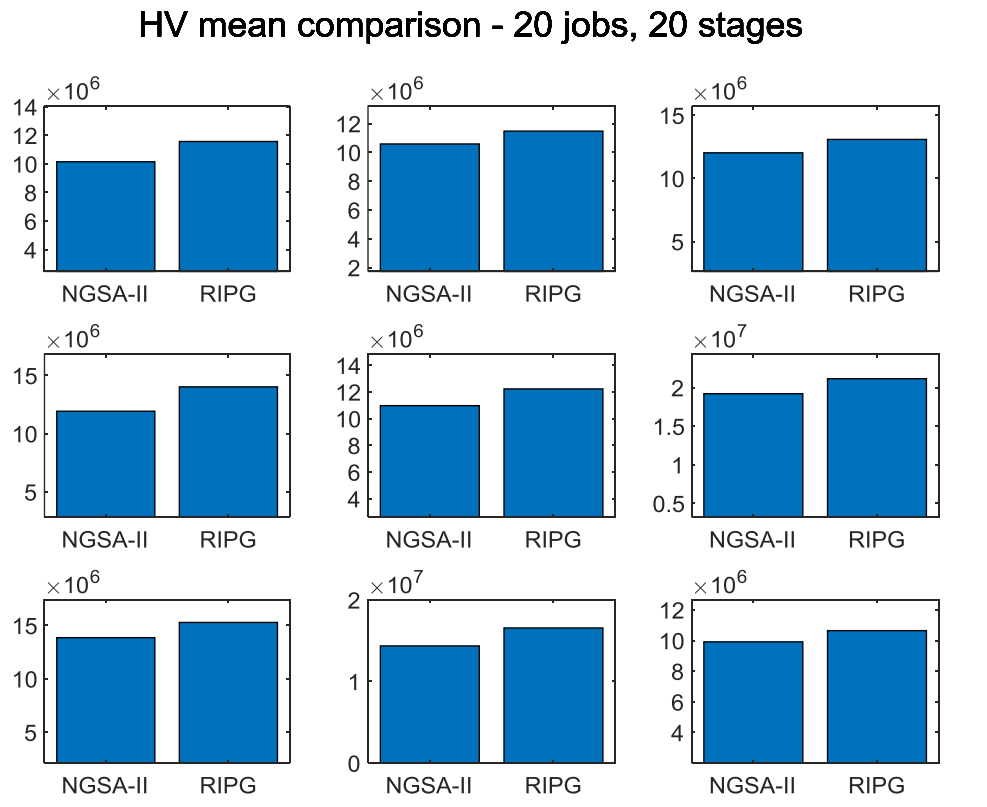


Figura 19 - HV means comparison 20 jobs 20 stages

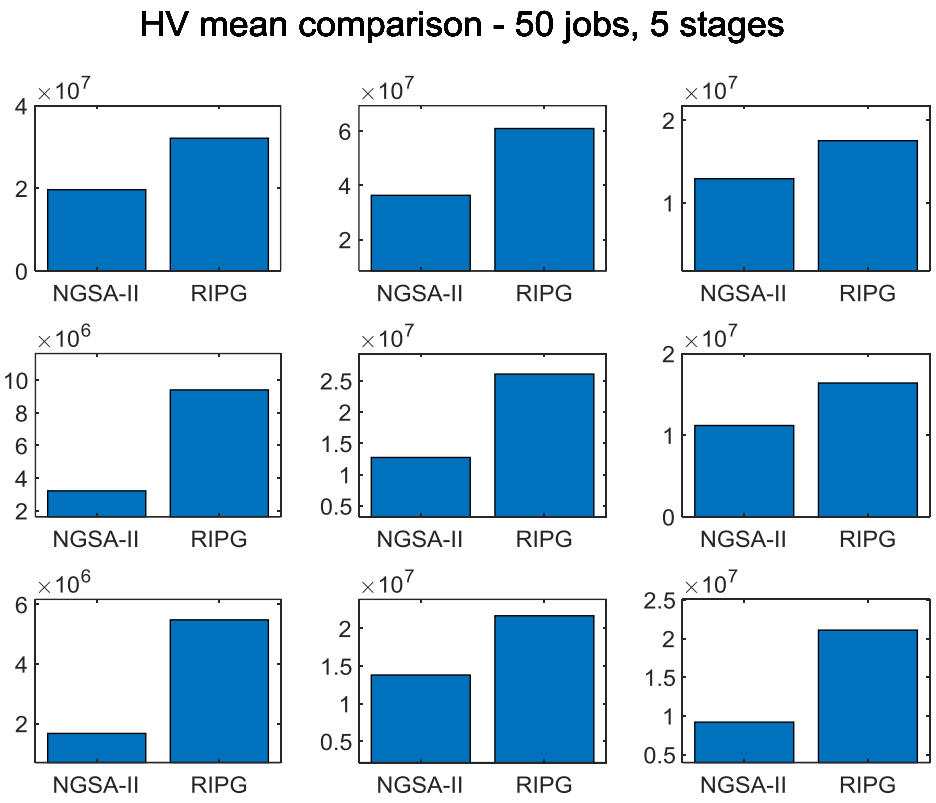


Figura 20- HV means comparison 50 jobs 5 stages

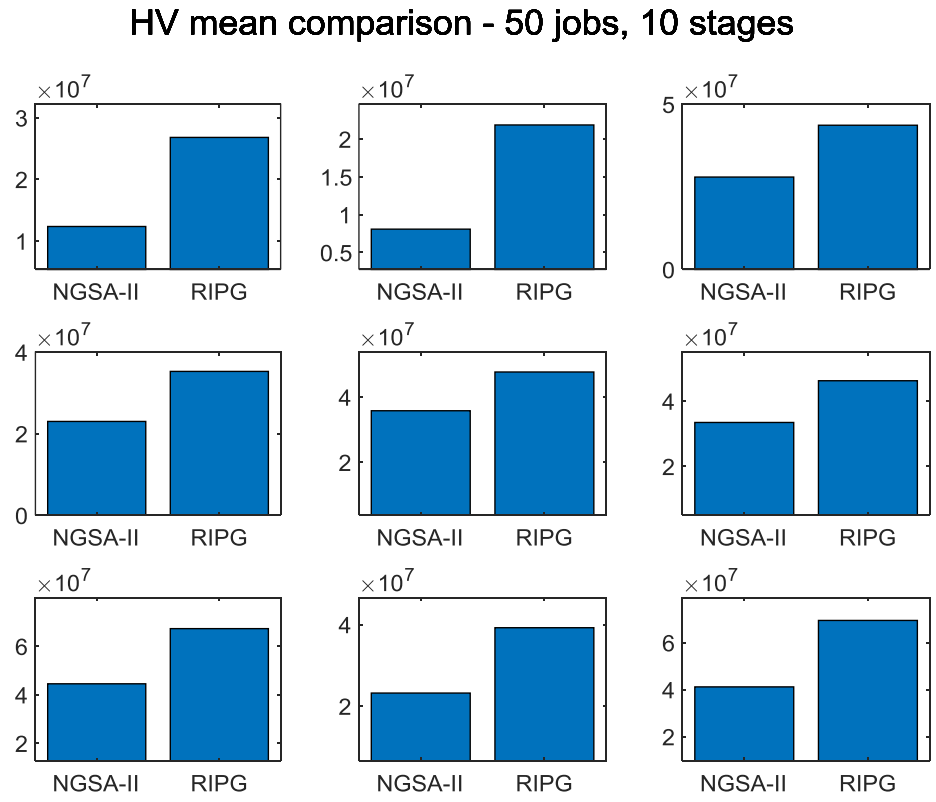


Figura 21- HV means comparison 50 jobs 10 stages

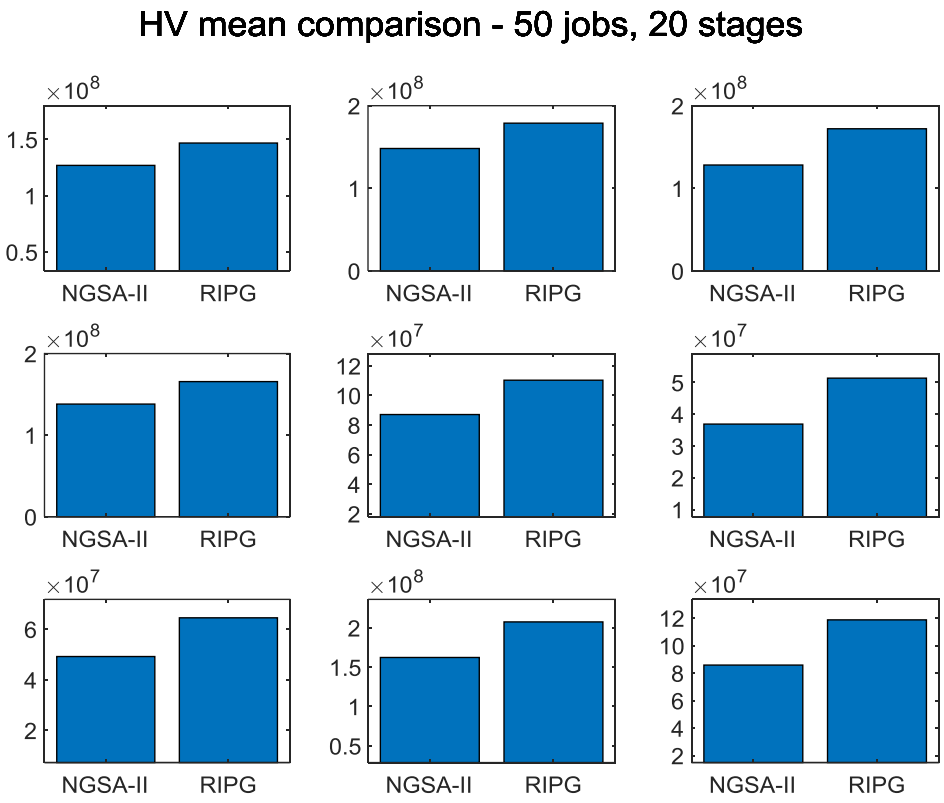


Figura 22- HV means comparison 50 jobs 20 stages

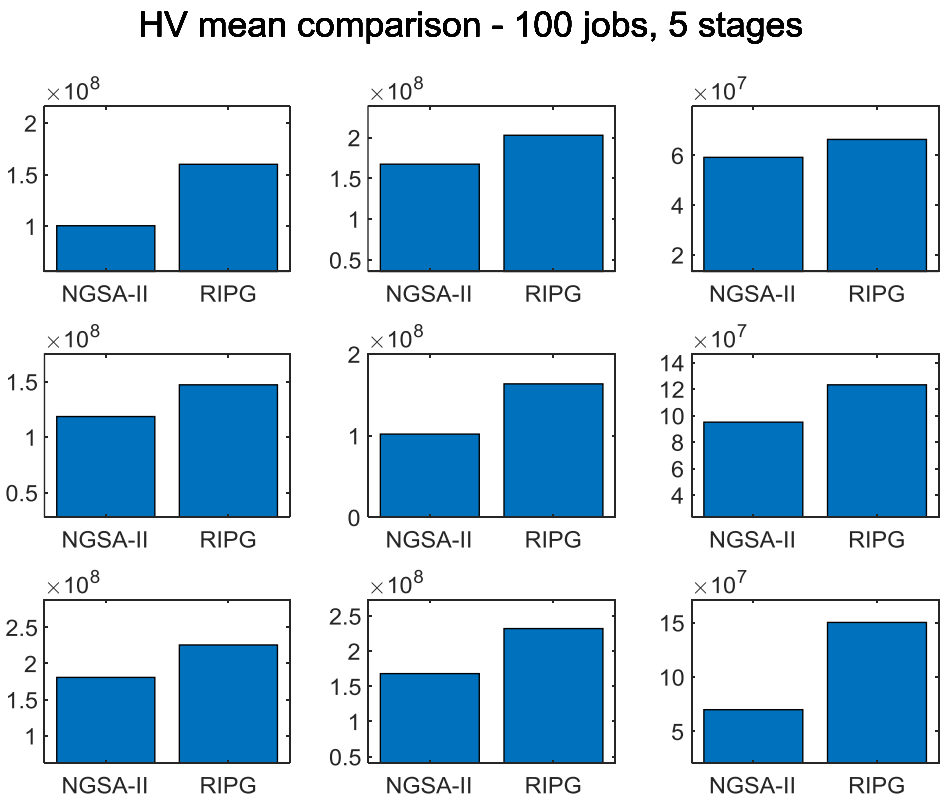


Figura 23 - HV means comparison 100 jobs 5 stages

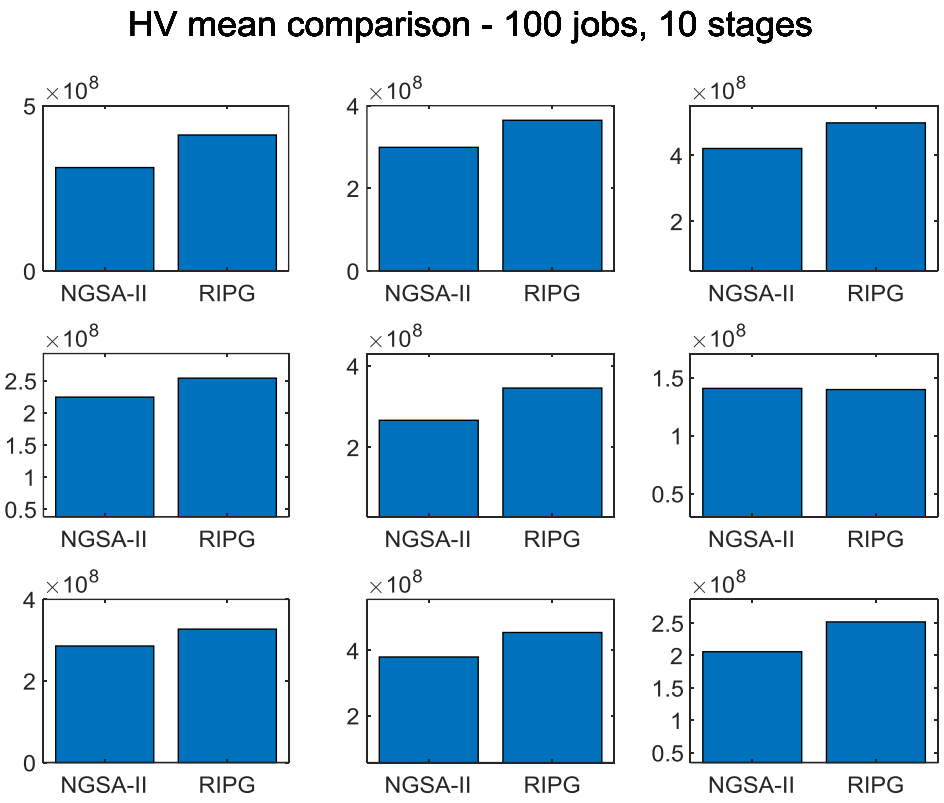


Figura 24- HV means comparison 100 jobs 10 stages

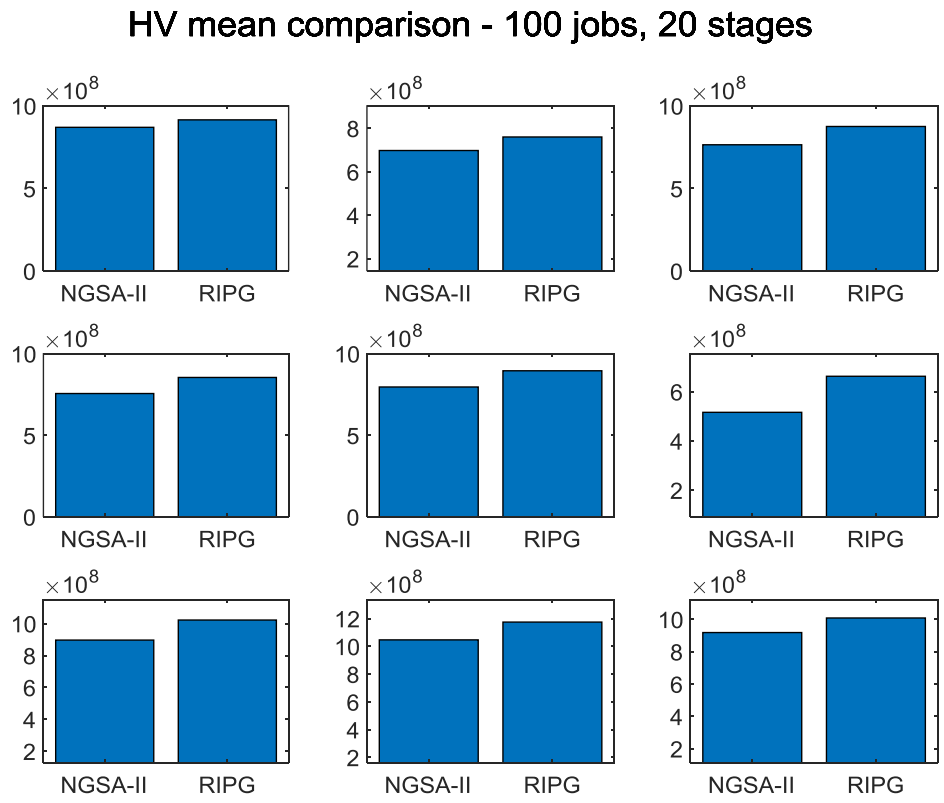


Figura 25 - HV means comparison 100 jobs 20 stages

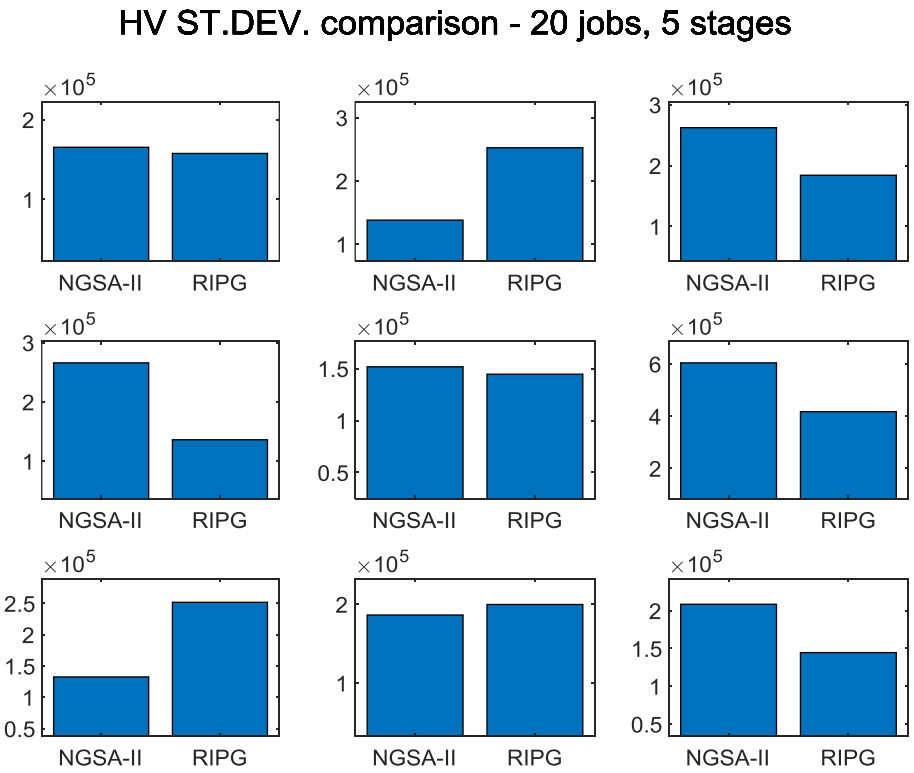


Figura 26- Standard deviation comparison 20 jobs 5 stages

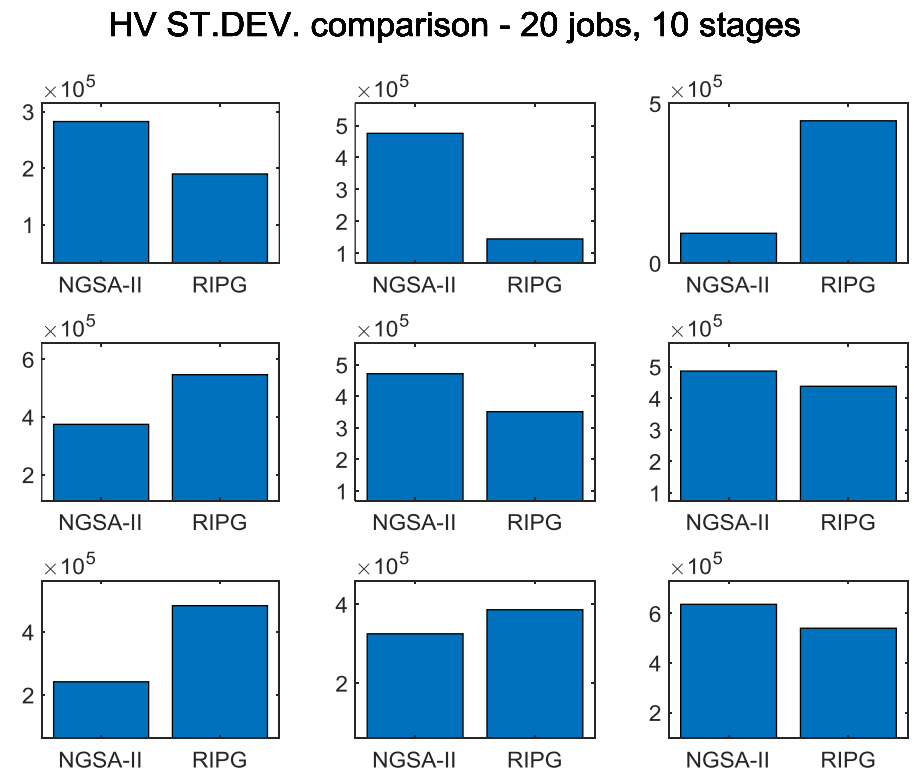


Figura 27 - Standard deviation comparison 20 jobs 10 stages

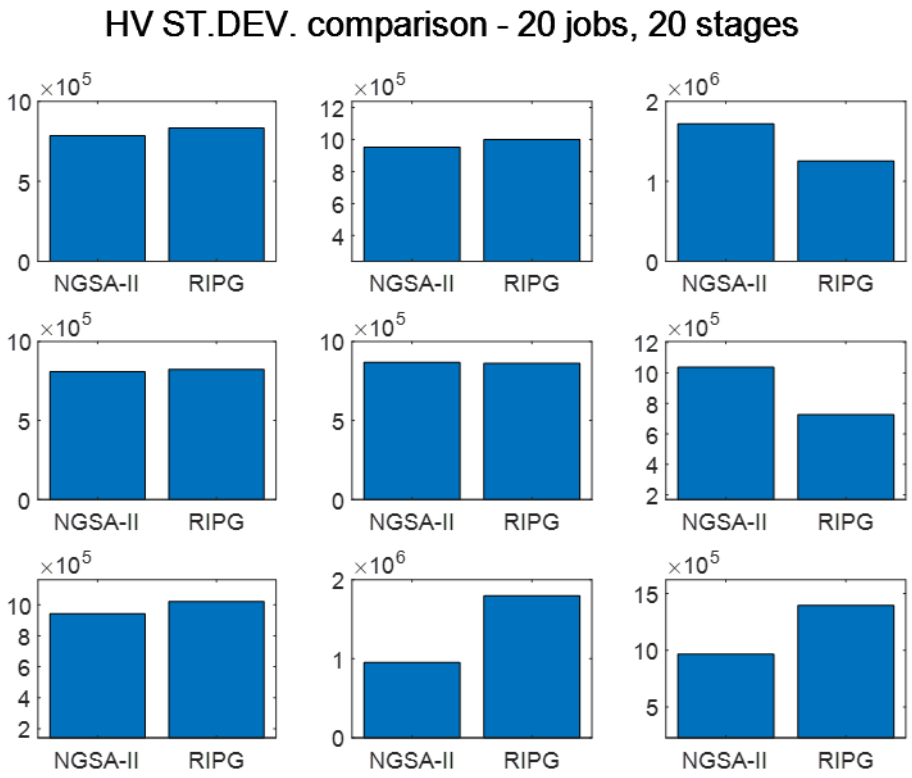


Figura 28 - Standard deviation comparison 20 jobs 20 stages

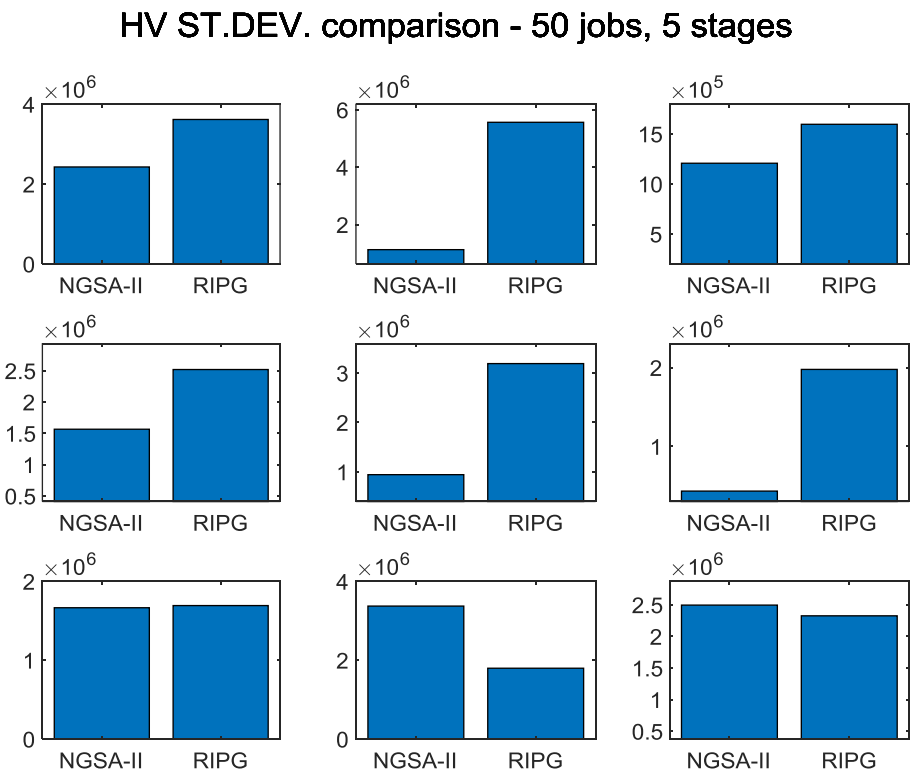


Figura 29 - Standard deviation comparison 50 jobs 5 stages

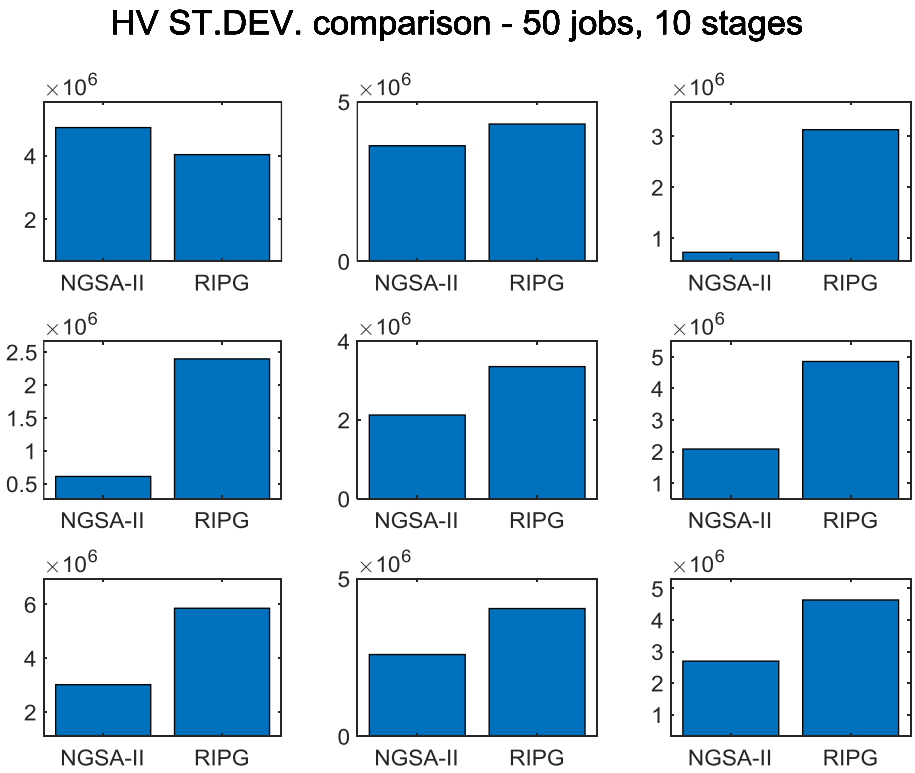


Figura 30- Standard deviation comparison 50 jobs 10 stages

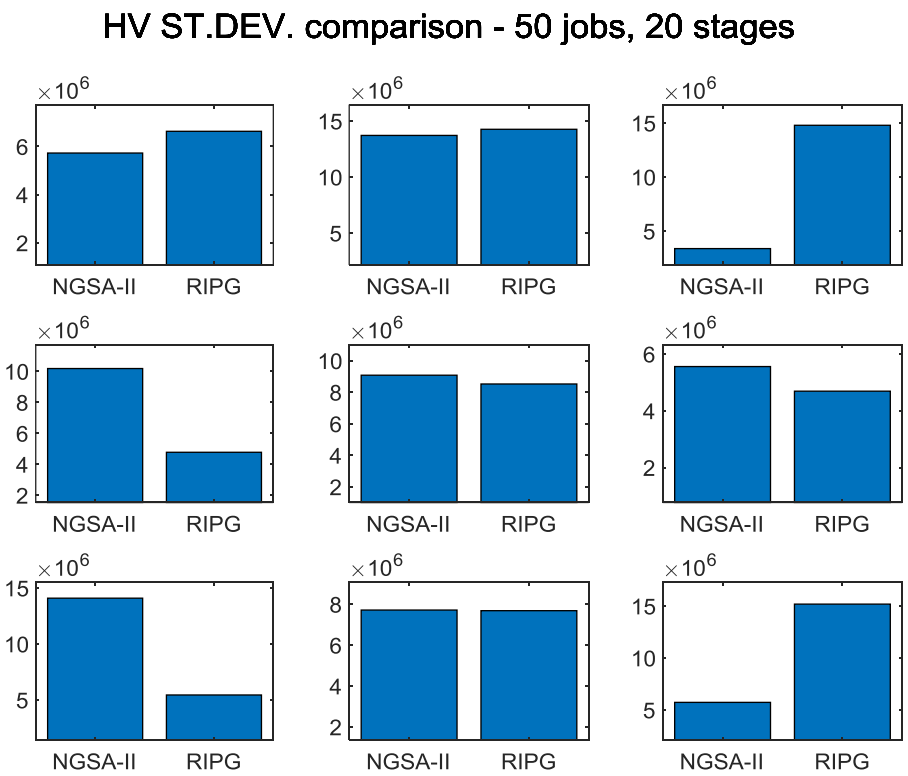


Figura 31 - Standard deviation comparison 50 jobs 20 stages

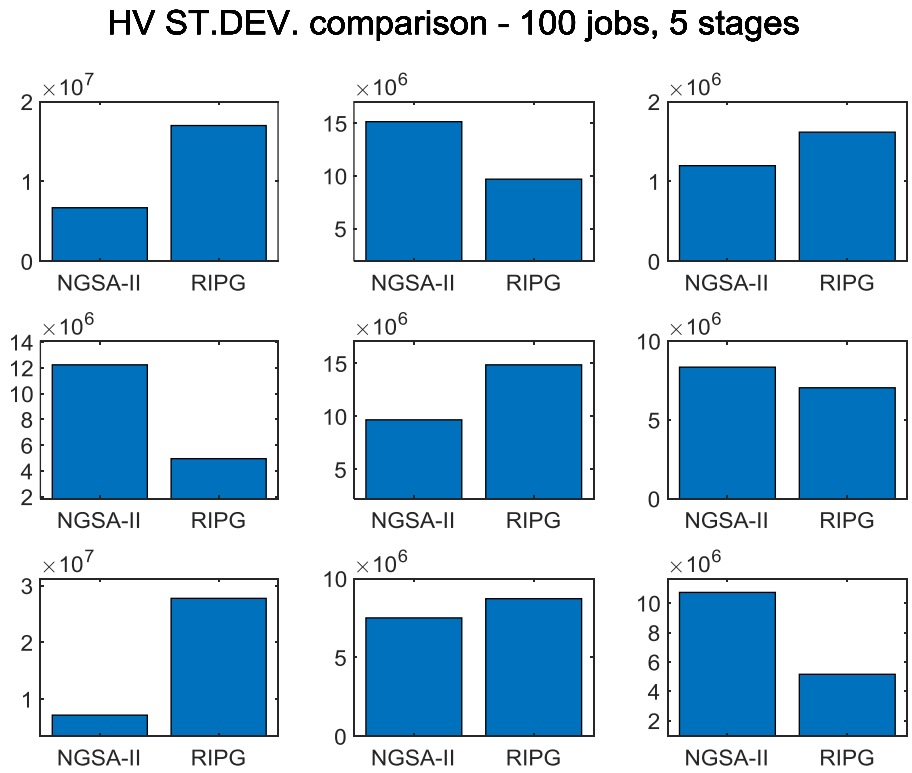


Figura 32 - Standard deviation comparison 100 jobs 5 stages

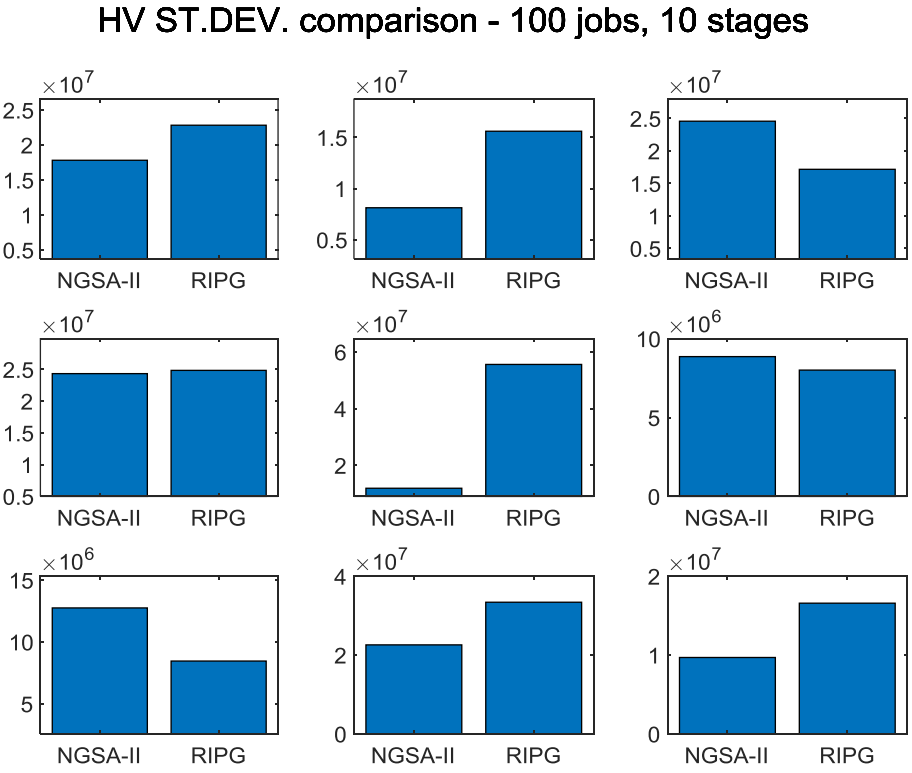


Figura 33 - Standard deviation comparison 100 jobs 10 stages

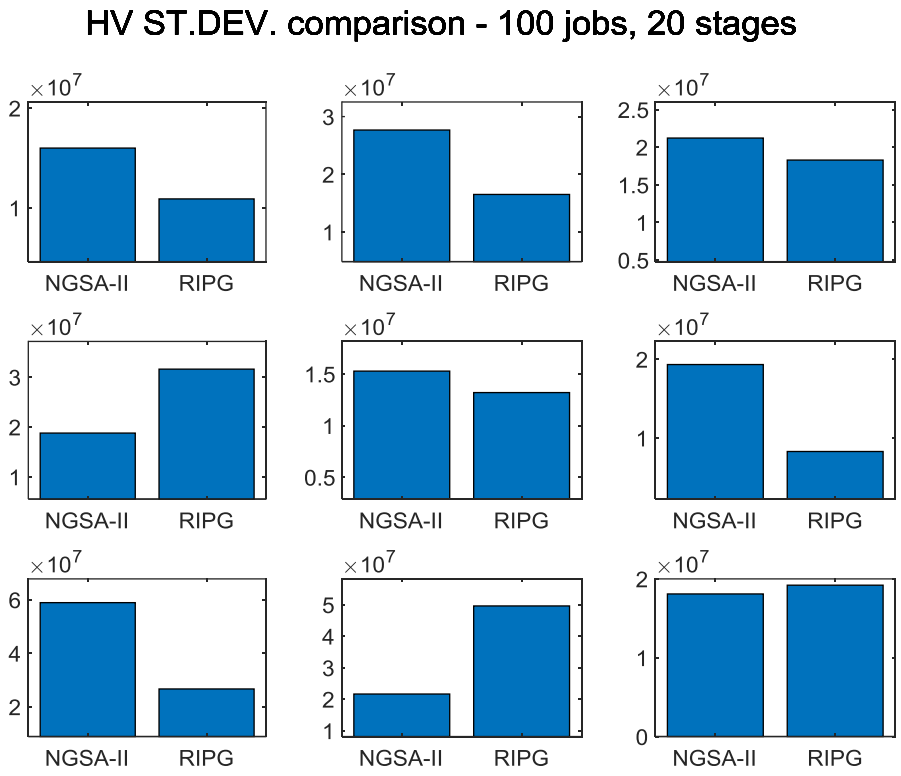


Figura 34 - Standard deviation comparison 100 jobs 20 stages

- Both the comparison plot of the fronts and the HV means for each instance shows that generally, it seems RIPG have better performances in almost each case we have considered.

  To have a statistical confirmation of what stated, ANOVA analysis is conducted in the following slides. The comparison is performed using RDI. The ANOVA results are listed below.

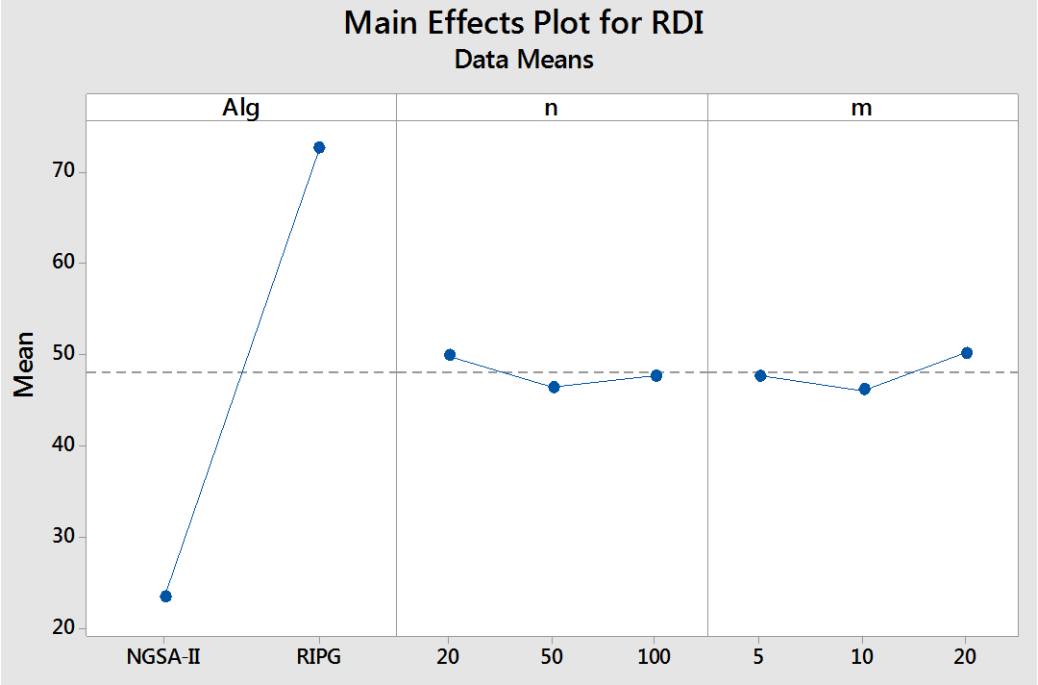


Figura 35 - Main effect plot for comparison

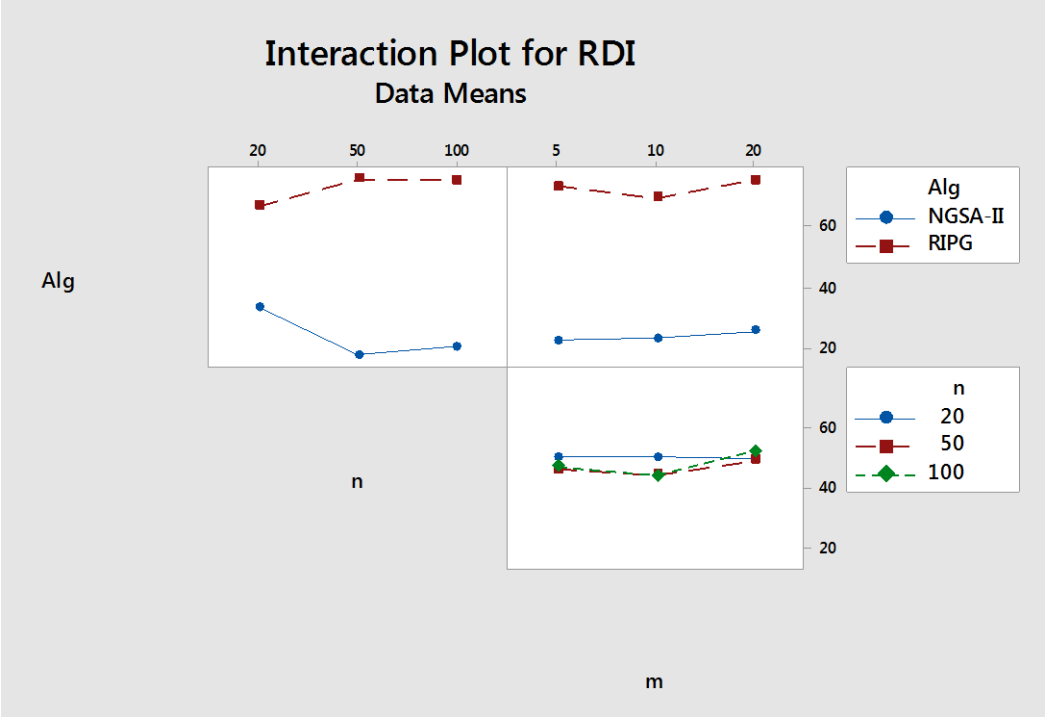


Figura 36- Interaction plot for comparison

Factor Information

Factor	Type	Levels	Values
Alg	Fixed	2	NGSA-II; RIPG
n	Fixed	3	20; 50; 100
m	Fixed	3	5; 10; 20

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Alg	1	544137	544137	1103,16	**0,000**
n	2	1853	926	1,88	0,153
m	2	2591	1296	2,63	0,073
Alg*n	2	27391	13695	27,77	**0,000**
Alg*m	2	881	441	0,89	0,410
n*m	4	1959	490	0,99	0,411
Error	886	437020	493		
Lack-of-Fit	4	6459	1615	3,31	0,011
Pure Error	882	430562	488		
Total	899	1015832			

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
22,2093	56,98%	**56,35%**	55,61%

Tabella 10 - ANOVA for comparison

ANOVA shows that the factor “Algorithm” and its interaction with the number of jobs n, seems to be the only relevant factors. Now the hypothesis of normality of SRES, their interval and test for equal variances must be verified to check if the results obtained by ANOVA can be accepted or further elaborations of the data are needed.

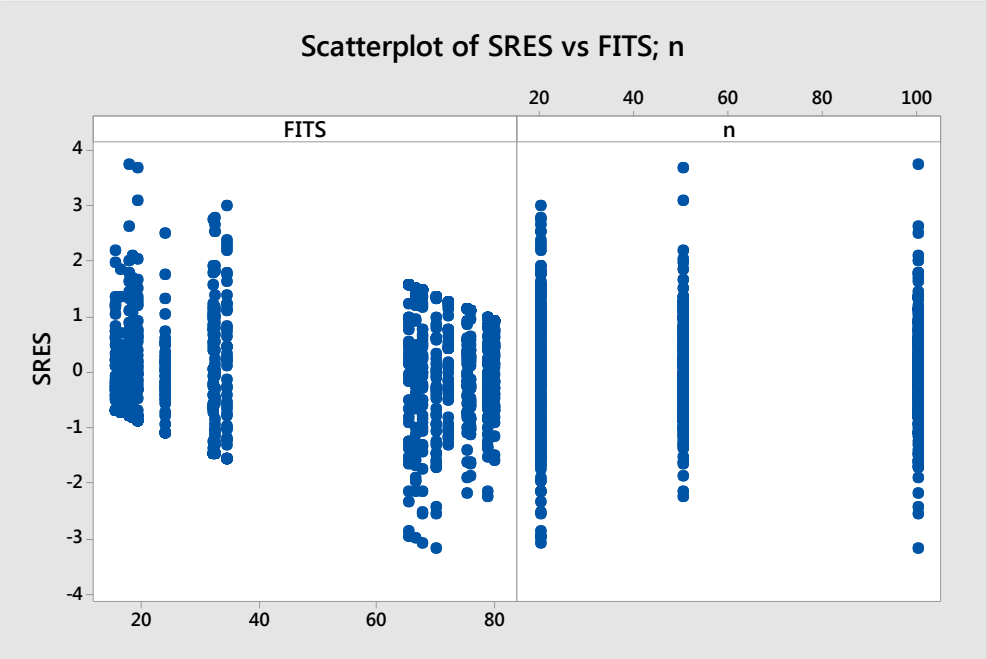


Figura 37- Scatterplot SRES for comparison

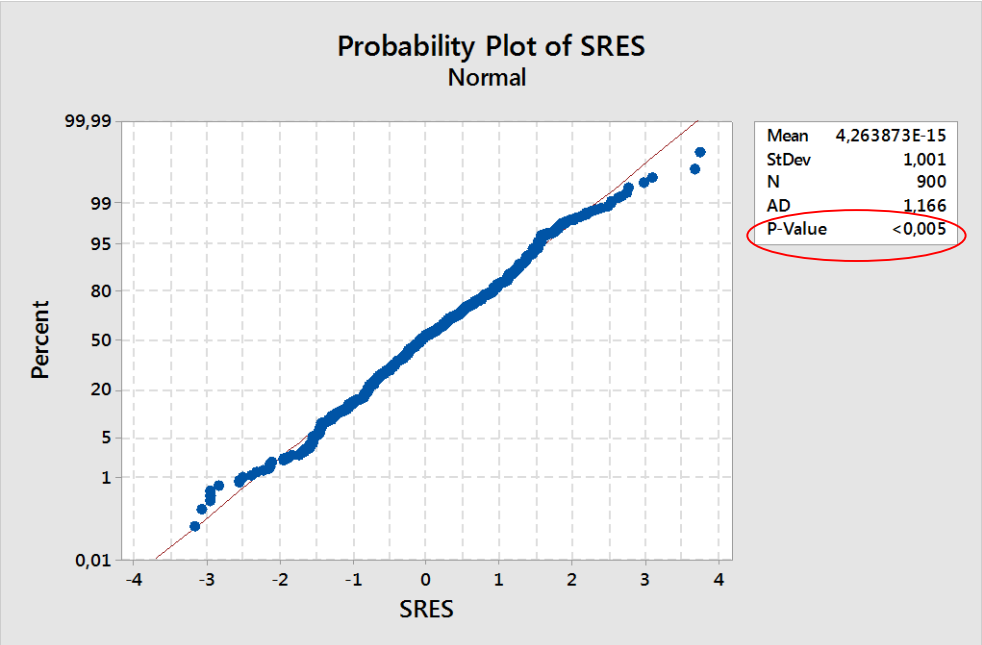


Figura 38 - Normality test for comparison

The hypothesis of **normality of SRES is rejected** because p-value is less than 0.05. Anyway the distribution and the shape of the data allow to keep the validity of conclusions obtained by the ANOVA. The SRES are **almost concentrated between [-3, 3].** So same considerations can be done.

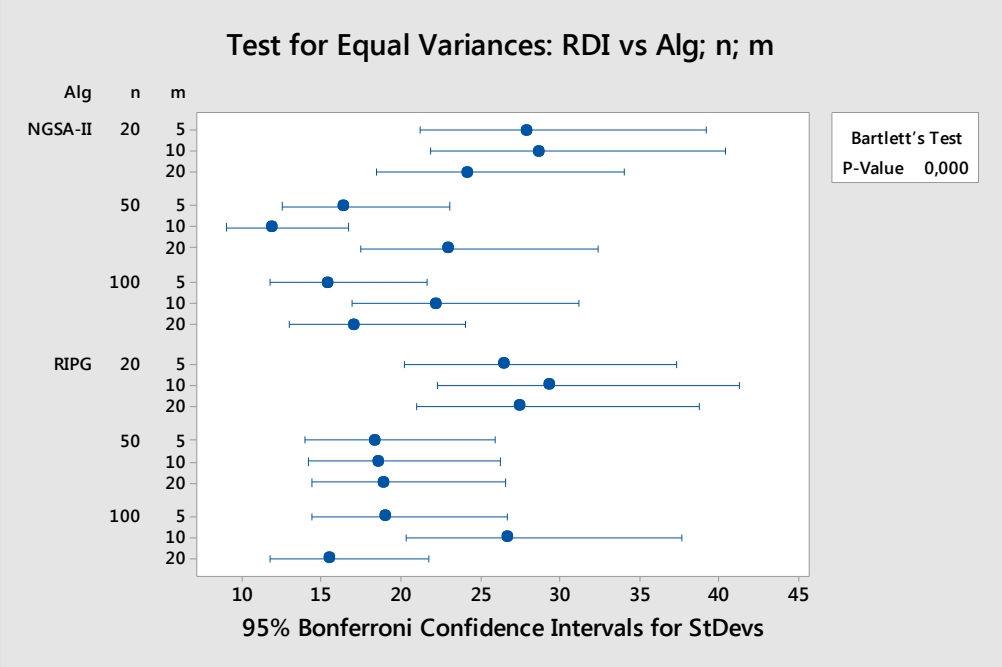


Figura 39 - Test for equal variances for comparison

Comparisons for RDI

Tukey Pairwise Comparisons: Alg*n

Grouping Information Using the Tukey Method and 95% Confidence

Alg*n	N	Mean	Grouping
RIPG 50	150	75,7169	A
RIPG 100	150	75,1984	A
RIPG 20	150	66,7053	B
NGSA-II 20	150	33,0011	C
NGSA-II 100	150	20,0301	D
NGSA-II 50	150	17,0580	D

Means that do not share a letter are significantly different.

Tabella 11 - Tukey test for comparison

So, from these graphs we can see that even if the normality of SRES in the ANOVA of comparison is rejected, the figures shows a trend which let us think the Tukey test can be performed and the ANOVA results are valid. The conclusion is that, is evident that the performance showed by the **RIPG is far better than NSGA-II** in almost every case we have considered, independently from the number of jobs and machines we are considering. The results, of this research, may be useful for future research, towards the development of new solution methods, and/or for the application of methods investigated in the context of real companies, with this kind of scheduling problem.

So from this analysis we can finally conclude that:

- Even if the normality of SRES in the ANOVA of comparison is rejected, the figures shows a trend which let us think the Tukey test can be performed and the ANOVA results are valid.
- The conclusion is that, is evident that the performance showed by the **RIPG are far better than NSGA-II** in almost every case we have considered, independently from the number of jobs and machines we are considering.
- The results, of this research, may be useful for future research, towards the development of new solution methods, and/or for the application of methods investigated in the context of real companies, with this kind of scheduling problem.

Chapter 6

[6] Conclusions

In this study we have proposed a RIPG algorithm to solve the hybrid flow shop scheduling problem with unrelated machines and machine eligibility constraint to minimize the total tardiness and total setup time. This research is motivated by considering some features common in the industrial field of the Hybrid Flowshop yet not well considered in the literature of the optimization of production scheduling. There have been several methods proposed in the literature for the a posteriori multi-objective flowshop problem. However, as far as we know, the setup times have seldom been considered in the objective function therein. This paper represents a first attempt to tackle this problem. A relatively new approach, the iterated greedy procedure, has been adopted for the problem. The proposed algorithm is analyzed in detail, and its performance is shown to be superior to the conventional multi-objective approach. Moreover, by analyzing two widely used decoding methods, permutation scheduling and list scheduling, we discover that in our context and for the considered objectives, PS is shown providing better results. Indeed, the PS has a higher controllability on the schedule building procedure, and it seems to be a fundamental factor for its superiority, especially when the number of jobs increases.

We have presented two main contributions to the field of the multi-objective flowshop. First, as said before, we have considered for this environment, the sequence-dependent setup times both in the shop model and the objective functions. Second, we have extended a new strategy which achieved state-of-the-art results for the single objective flowshop, the Iterated Greedy metaheuristic, in order to deal with several objectives . With this we highlight the relevance of a scientific and algorithm engineering approach in designing and developing algorithms for manufacturing systems. The limited yet focused campaign of tests by means of ANOVA have confirmed that this algorithm shows far better performances than a serious competitor well applied in this field as NGSAII. As a consequence, the proposed method can be considered the one of the state-of-art procedure for this scheduling problem.

Future research lines stem from the possibility of, first, applying this scheme to solve different or more realistic and complicated scheduling problems; second, investigating the possibility to incorporate user preference information into the search procedure for a more concentrated search on the objective space and to generate higher quality solutions aligned to the user preference.

[7] References

[1] Rubén Ruiz, [a]José Antonio Vázquez-Rodríguez[b] (2009);
The Hybrid Flow shop scheduling problem ;
European Journal of Operational Research,Volume 205, Issue 1, 2010, pp 1-18,

[2] Allahverdi A, Soroush H M.
The significance of reducing setup times/setup costs.
European Journal of Operational Research, 2008, 187(3): 978-984.

[3] Deb K, Pratap A, Agarwal S, et al.
A fast and elitist multiobjective genetic algorithm: NSGA-II.
IEEE transactions on evolutionary computation, 2002, 6(2): 182-197.

[4] Zitzler E, Laumanns M, Thiele L.
SPEA2: Improving the strength Pareto evolutionary algorithm.
TIK-report, 2001, 103.

[5] Stützle T, Ruiz R.
A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem,
European Journal of Operational Research, Volume 177, Issue 3, 2007, Pages 2033-2049,

[6] Ruiz R, Stützle T.
An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives.
European Journal of Operational Research, 2008, 187(3): 1143-1159.

[7] Ciavotta M, Minella G, Ruiz R.
Multi-objective sequence dependent setup times permutation flowshop: A new algorithm and a comprehensive study.
European Journal of Operational Research, 2013, 227(2): 301-313.

[8] J.N.D. Gupta,
Two-stage, hybrid flow shop scheduling problem,
Journal of theOperational Research Society 39 (4) (1988) 359–364

[9] J.A.Hoogeveen,J.K.Lenstra,B.Veltman,
Preemptive scheduling in atwo-stagemultiprocessor flow shop is np-hard,
European Journal of OperationalResearch 89 (1) (1996) 172–175.

[10] M.R. Garey, D.S. Johnson,
Computers and intractability: a guide to the theoryof NP-completeness,
A Series of Books in the Mathematical Sciences, W.H.Freeman, San Francisco, 1979

[11] H. Djellab, K. Djellab,
Preemptive hybrid flowshop scheduling problem of interval orders,
European Journal of Operational Research 137 (1) (2002) 37–49

[12] R.J. Wittrock,
Scheduling algorithms for flexible flow lines,
IBM Journal of Research and Development 29 (4) (1985) 401–412

[13] R.J. Wittrock,
An adaptable scheduling algorithm for flexible flow lines,
Operations Research 36 (3) (1988) 445–453

[14] C.-Y. Liu, S.-C. Chang,
Scheduling flexible flow shops with sequence-dependent setup effects,
IEEE Transactions on Robotics and Automation 16(4) (2000) 408–419

[15] Z.H. Jin, K. Ohno, T. Ito, S.E. Elmaghraby,
Scheduling hybrid flowshops in printed circuit board assembly lines,
Textile production systems: a succession of non-identicalparallel processor shops,
Production and Operations Management11 (2) (2002) 216–230

[16] H.D. Sherali, S.C. Sarin, M.S. Kodialam,
Models andalgorithms for a two-stageproduction process, Production Planning and Control
Production Planning & Control, (1990) pp. 27–39

[17] J. Grabowski, J. Pempera,
Sequencing of jobs in some production system,
European Journal of Operational Research 125 (3) (2000) 535–550

[18] A.G.P. Guinet,
Textile Production Systems: a Succession of Non-identical Parallel Processor Shops
Journal of the Operational Research Society 42 (8)(1991)655-671

[19] H. Tsubone, M. Ohba, H. Takamuki, Y. Miyake,
A production schedulingsystem for a hybrid flow-shop: a case-study,
Omega-International Journal of Management Science 21 (2) (1993) 205–214

[20] E.-H. Aghezzaf, H. Van Landeghem,
An integrated model for inventory andproduction planning in a two-stage hybrid production system,
International Journal of Production Research 40 (17) (2002) 4323–4339

[21] L. Adler, N. Fraiman, E. Kobacker, M. Pinedo, J.C. Plotnicoff, T.P. Wu, Bpss:
A scheduling support system for the packaging industry,
Operations Research41 (4) (1993) 641–648.

[22] D.E. Deal, T. Yang, S. Hallquist,
Job scheduling in petrochemical production:two-stage processing with finite intermediate storage,
Computers and Chemical Engineering 18 (4) (1994) 333–344.

[23] E.-H. Aghezzaf, A. Artiba, O. Moursli, C. Tahon,
Hybrid flowshop problems, adecomposition based heuristic approach,
Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'95,FUCAM-INRIA, 1995, pp. 43–56

[24] H.-T. Lin, C.-J. Liao,
A case study in a two-stage hybrid flow shop with setuptime and dedicated machines,
International Journal of Production Economics86 (2) (2003) 133–143

[25] S. Bertel, J.-C. Billaut,
A genetic algorithm for an industrial multiprocessorflow shop scheduling problem with recirculation,
European Journal of Operational Research 159 (3) (2004) 651–662

[26] T. Yang, Y. Kuo, I. Chang,
Tabu-search simulation optimization approach forflow-shop scheduling with multiple processors – a case study,
International Journal of Production Research 42 (19) (2004) 4015–403

[27] R. Ruiz, C. Maroto,
A genetic algorithm for hybrid flowshops with sequencedependent setup times and machine eligibility,
European Journal of Operational Research 169 (3) (2006) 781–800

[28] M. Dror, P.A. Mullaserif,
Three stage generalized flowshop: scheduling civilengineering projects,
Journal of Global Optimization 9 (3-4) (1996) 321–344

[29] A. Allahverdi, F.S. Al-Anzi,
Scheduling multi-stage parallel-processor servicesto minimize average response time,
Journal of the Operational ResearchSociety 57 (1) (2006) 101–110.

[30] L. Chen, L.-F. Xi, J.-G. Cai, N. Bostel, P. Dejax,
An integrated approach formodeling and solving the scheduling problemof container handling systems,
Journal of Zhejiang University Science A 7 (2) (2006) 234–239.

[31] L. Chen, N. Bostel, P. Dejax, J.C. Cai, L.F. Xi,
A tabu search algorithm for theintegrated scheduling problem of container handling systems in a maritime terminal,
European Journal of Operational Research 181 (1) (2007) 40–58.

[32] H. Holland
Adaptation in natural and artificial systems
The University of Michigan Press, Ann Arbor (1975)

[33] C.L. Chen, V.S. Vempati, N. Aljaber
An application of genetic algorithms for flow shop problems
Eur J Oper Res, 80 (1995), pp. 389-396

[34] T. Murata, H. Ishibuchi, H. Tanaka
Genetic algorithms for flow-shop scheduling
Comput Indus Eng, 30 (1996), pp. 1061-1071

[35] C. Oguz, M.Ercan
Performance of local search heuristics on scheduling a class of pipelined multiprocessor tasks
Comp. and Electrical Eng. (2005)

[36] F. Choong, S. Phon-Amnuaisuk, M.Y. Alias
Metaheuristic methods in hybrid flow shop scheduling problem
Expert Systems with Applications (2011)

[37] J. Jdrzêjowicz, P. Jdrzêjowicz
Population-based approach to multiprocessor task scheduling in multistage hybrid flow shops
Lecture Notes in Computer Science, 2773 (2003), pp. 279-286

[38] C. Oğuz, Y. Zinder, V. Do, A. Janiak, M. Lichtenstein
Hybrid flow-shop scheduling problems with multiprocessor task systems
European Journal of Operations Research, 152 (2004), pp. 115-131

[39] K.C. Ying, S.W. Lin
Multiprocessor task scheduling in multistage hybrid flowshops: An ant colony system approach
International Journal of Production Research, 44 (2006), pp. 3161-3177

[40] D.F. Shiau, S.C. Cheng, Y.M. Huang
Proportionate flexible flow shop scheduling via a hybrid constructive genetic algorithm
Expert Systems with Applications, 34 (2008), pp. 1133-1143

[41] C. Yu, Q.Semeraro, A.Matta
A genetic algorhitm for the hybrid flow shop scheduling with unrelated parallel machines and machine eligibility
Computers and Operations Research, 100, (2018), pp. 211-229

[42] W. Bozejko, J. Pempera, c.Smutnicki
Parallel tabu search algorithm for the hybrid flow shop problem
Computers & Industrial engineering , 65, 2013, pp. 466-474

[43] E. Nowicki, C. Smutnicki
The flow shop with parallel machines: A tabu search approach
European Journal of Operational Research, 106 (1998), pp. 226-253

[44] J. Kennedy, R. Eberhart
Particle swarm optimization
IEEE International Conference on Neural Networks, vol. 4 (1995), pp. 1942-1948

[45] Y. Shi, R.C. Eberhart
A modified particle swarm optimizer
Proc. Of the IEEE International Conference of Evolutionary Computation, IEEE Press, Piscataway (1998), pp. 69-73

[46] A. Przybylski, X. Gandibleux,
Multi-objective branch and bound
European Journal of Operational Research, Volume 260, Issue 3,(2017) pp. 856-872

[47] Peter Wilson, H. Alan Mantooth,
Model-Based Optimization Techniques,
Model-Based Engineering for Complex Electronic Systems, Newnes, 2013, pp. 347-367,

[48] M. Gilli, D. Maringer, E. Schumann,
Heuristic Methods in a Nutshell,
Numerical Methods and Optimization in Finance,Academic Press, 2011, pp 337-379,

[49] Etmaghraby S. E. & Kamoub R. E. (1997).
Production control in hybrid flowshops: an example from textile manufacturing.
The Planning and Scheduling of Production Systems (Artiba a. and Elmaghraby S. E. ed.). Chap. 6, Chapman & Hall, UK

[50] Vignier, Billaut, Proust, 1999
Les problèmes d'ordonnancement de type flow-shop hybride: état de l'art
RAIRO-Oper. Res., 33 (2) (1999), pp. 117-183

[51] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi,
Optimization by Simulated Annealing,
Readings in Computer Vision, Morgan Kaufmann, 1987, pp. 606-615

[52] M. Zandieh, S.M.T.F. Ghomi, S.M.M. Husseini
An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times
Appl. Math. Comput., 180 (1) (2006), pp. 111-127

[53] Z. Cui, X. Gu
An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems
Neurocomputing, 148 (2015), pp. 248-259

[54] J. Li, Q. Pan, P. Duan**An improved artificial bee colony algorithm for solving hybrid flexible flowshop with dynamic operation skipping**
IEEE Trans. Cybern., 46 (6) (2016), pp. 1311-1324

[55] F. Pargar, M. Zandieh
Bi-criteria SDST hybrid flow shop scheduling with learning effect of setup times: water flow-like algorithm approach
Int. J. Prod. Res., 50 (10) (2012), pp. 2609-2623

[56] M.K. Marichelvam, T. Prabaharan, X.S. Yang
Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan
Appl. Soft Comput., 19 (2014), pp. 93-101

[57] M.K. Marichelvam, T. Prabaharan, X.S. Yang
A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems
IEEE Trans. Evol. Comput., 18 (2) (2014), pp. 301-305

[58] D.H. Wolpert, W.G. Macready
No free lunch theorems for optimization
IEEE Trans. Evol. Comput., 1 (1) (1997), pp. 67-82

[59] R. Ruiz, T. Stützle
A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem
European Journal of Operational Research, 177 (3) (2007), pp. 2033-2049

[60] M. Nawaz, E.E. Enscore Jr, I. Ham**A heuristic algorithm for the *m* machine, *n* job flowshop sequencing problem**
Omega-International Journal of Management Science, 11 (1) (1983), pp. 91-95

[61] R. Ruiz, T. Stützle
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) (2008), pp. 1143-1159

[62] K.-C. Ying
An iterated greedy heuristic for multistage hybrid flowshop scheduling problems with multiprocessor tasks
IEEE Transactions on Evolutionary Computation, 60 (6) (2008), pp. 810-817

[63] F. Toyama, K. Shoji, J. Miyamichi
An iterated greedy algorithm for the node placement problem in bidirectional manhattan street networks
Proceedings of the 10th annual conference on genetic and evolutionary computation,(2008), pp. 579-584

[64] Y. Zhi, F. Armin, H. Henning, B. Prasanna, S. Thomas, S. Michael
Iterated greedy algorithms for a real-world cyclic train scheduling problem
Hybrid metaheuristics, Springer, Berlin (2008), pp. 102-116

[65] M.F. Tasgetiren, Q.-K. Pan, Y.-C. Liang
A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times
Computers & Operations Research, 36 (6) (2009), pp. 1900-1915

[66] N. Karimi, M. Zandieh, H.R. Karamooz,
Bi-objective group scheduling in hybrid flexible flowshop: A multi-phase approach,
Expert Systems with Applications,Volume 37, Issue 6,(2010) pp 4024-4032,

[67] H. Land, A & G. Doig, A. (1960).
An Automatic Method of Solving Discrete Linear Programming Problems.
Econometrica. 28. 497-520. 10.2307/1910129.

[68] G. Kiziltan, E. Yucaoglu
An algorithm for multiobjective zero-one linear programming
Management Science, 29 (1983), pp. 1444-1453

[69] Mir Abbas Bozorgirad, Rasaratnam Logendran,
Sequence-dependent group scheduling problem on unrelated-parallel machines, Expert Systems with Applications,
Volume 39, Issue 10, 2012, pp. 9021-9030,

[70] H. Asefi, F.Jola, M.Rabiee, M.E. Tayebi Araghi
A hybrid NGSA-II and VNS for solving a bi-objective no-wait flexible flowshop scheduling problem
The international Journal of Advanced Manufacturing Technology, (2014), Volume 75, Issue 5-8, pp 1017-1033

[71] Huixin Tian, Kun Li, Wei Liu
A Pareto-Based Adaptive Variable Neighborhood Search for Biobjective Hybrid Flow Shop Scheduling Problem with Sequence-Dependent Setup Time
Mathematical Problems in Engineering Volume 2016, Article ID 1257060, 11 pages

[72] A. Sioud, C. Gagné,
Enhanced migrating birds optimization algorithm for the permutation flow shop problem with sequence dependent setup times,
European Journal of Operational Research, Volume 264, Issue 1, 2018, Pp 66-73,

[73] Ruiz, R., Maroto, C., Alcaraz, J.
Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics
(2005) European Journal of Operational Research, 165 (1), pp. 34-54.

[74] Rajendran, C., Ziegler, H.
Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times
(2003) European Journal of Operational Research, 149 (3), pp. 513-522.

[75] Ali Allahverdi, C.T. Ng, T.C.E. Cheng, Mikhail Y. Kovalyov,
A survey of scheduling problems with setup times or costs,
European Journal of Operational Research, 2008, pp. 985-1032,

[76] Eva Vallada, Rubén Ruiz, Gerardo Minella,
Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics,
Computers & Operations Research, Volume 35, Issue 4, 2008, pp. 1350-1373

[77] Tadahiko Murata, Hisao Ishibuchi, Hideo Tanaka,
Multi-objective genetic algorithm and its applications to flowshop scheduling,
Computers & Industrial Engineering, Volume 30, Issue 4, 1996, pp 957-968,

[78] H. Ishibuchi and T. Murata,
A multi-objective genetic local search algorithm and its application to flowshop scheduling,
in *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 28, no. 3, pp. 392-403, Aug. 1998.

[79] H. Ishibuchi, T. Yoshida and T. Murata,
Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling,"
in *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 204-223, April 2003.

[80] R. K. Suresh and K. M. Mohanasundaram,
Pareto archived simulated annealing for permutation flow shop scheduling with multiple objectives,
IEEE Conference on Cybernetics and Intelligent Systems, 2004., Singapore, 2004, pp. 712-717.

[81] T.K. Varadharajan, Chandrasekharan Rajendran,
A multi-objective simulated-annealing algorithm for scheduling in flowshops to minimize the makespan and total flowtime of jobs,
European Journal of Operational Research, Volume 167, Issue 3, 2005, Pages 772-795,

[82] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.
A fast and elitist multiobjective genetic algorithm: NSGA-II
(2002) IEEE Transactions on Evolutionary Computation, 6 (2), pp. 182-197

[83] Yandra, H.Tamura
A new multiobjective genetic algorithm with heterogeneous population for solving flowshop scheduling problems
(2007), International Journal of Computer integrated Manufacturing, Volume 20, Issue 5, pp. 465-477

[84] Framina N,, Jose M, Leisten, Rainer,
A multi-objective iterated greedy search for flowshop scheduling with makespan and flowtime criteria
OR Spectrum, 2008", volume 30, pp. 787--804",

[85] E. Taillard
Some efficient heuristic methods for the flow shop sequencing problem
European Journal of Operational Research, 47 (1) (1990), pp. 67-74

[86] K. Deb
A fast and elitist multiobjective genetic algorithm: NSGA-II
IEEE Transactions on Evolutionary Computation, 6 (2) (2002), pp. 182-197

[87] G.C. Onwubolu, M. Mutingi
Genetic algorithm for minimizing tardiness in flow-shop scheduling
Production Planning and Control, 10 (1999), pp. 462-471

[88] V.A. Armentano, D.P. Ronconi
Tabu search for total tardiness minimization in flow-shop scheduling problems
Computers and Operations Research, 26 (1999), pp. 219-235

[89] S. Hasija, C. Rajendran
Scheduling in flowshops to minimize total tardiness of jobs
International Journal of Production Research, 42 (2004), pp. 2289-2301

[90] G. Onwubolu, D. Davendra
Scheduling flow shops using differential evolution algorithm
European Journal of Operational Research, 171 (2006), pp. 674-692

[91] E. Vallada, R. Ruiz, G. Minella
Minimising total tardiness in the *m*-machine flowshop problem: A review and evaluation of heuristics and metaheuristics
Computers & Operations Research, 35 (4) (2008), pp. 1350-1373

[92] E. Vallada, R. Ruiz
Cooperative metaheuristics for the permutation flowshop scheduling problem
European Journal of Operational Research, 193 (2009), pp. 365-376

[93] J.M. Framinan, R. Leisten
Total tardiness minimization in permutation flow shops: A simple approach based on a variable greedy algorithm
International Journal of Production Research, 46 (22) (2008), pp. 6479-6498

[94] E. Vallada, R. Ruiz
Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem
OMEGA, The International Journal of Management Science, 38 (2010), pp. 57-67

[95] T. Kellegöz, B. Toklu, J. Wilson
Elite guided steady-state genetic algorithm for minimizing total tardiness in flowshops
Computers & Industrial Engineering, 58 (2010), pp. 300-306

[96] T. Chen, X. Li
Integrated iterated local search for the permutation flowshop problem with tardiness minimization
Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC). Manchester, 13–16 October 2013 (2013)

[97] R. M'Hallah
An iterated local search variable neighborhood descent hybrid heuristic for the total earliness tardiness permutation flow shop
International Journal of Production Research, 52 (13) (2014), pp. 3802-3819

[98] T. Cura
An evolutionary algorithm for the permutation flowshop scheduling problem with total tardiness criterion
International Journal of Operational Research, 22 (3) (2015), pp. 366-384

[99] R. Ruiz, T. Stützle
A simple and effective iterated greedy algorithm for the permutation flow shop scheduling problem
European Journal of Operational Research, 177 (3) (2007), pp. 2033-2049

104] Capek, R., ˇ S˚ˇucha, P., Hanz´alek, Z.
Production scheduling with alternative process plans.
Eur. J. Oper. Res. 217(2), 300–311 (2012)

[105] . Kis, T.
Job-shop scheduling with processing alternatives.
Eur. J. Oper. Res. 151(2), 307–322 (2003)

[106] . Leung, C.W., Wong, T.N., Maka, K.L., Fung, R.Y.K.
Integrated process planning and scheduling by an agent-based ant colony optimization.
Comput. Ind. Eng. 59(1), 166–180 (2010)

[107] Li, X., Zhang, C., Gao, L., Li, W., Shao, X.
An agent-based approach for integrated process planning and scheduling.
Expert Syst. Appl. 37(2), 1256–1264 (2010)

[108] Shao, X., Li, X., Gao, L., Zhang, C.
Integration of process planning and scheduling - a modified genetic algorithm-based approach.
Comput. Oper. Res. 36(6), 2082– 2096 (2009)

[109] Allahverdi, A., Ng, C., Cheng, T., Kovalyov, M.Y.
A survey of scheduling problems with setup times or costs.
Eur. J. Oper. Res. 187(3), 985–1032 (2008)

[110] Yuan, X.M., Khoo, H.H., Spedding, T.A., Bainbridge, I., Taplin, D.M.R.
Minimizing total setup cost for a metal casting company.
Winter Simul. Conf. 2, 1189–1194 (2004)

[111] Wang, L., Wang, M.
A hybrid algorithm for earliness-tardiness scheduling problem with sequence dependent setup time.
Proceedings of the 36th Conference on Decision and Control, pp. 1219–1223. IEEE (1997)

[112] Mirabi, M.
A hybrid simulated annealing for the single-machine capacitated lotsizing and scheduling problem with sequence-dependent setup times and costs and dynamic release of jobs.
Int. J. Adv. Manuf. Technol. 54(9–12), 795–808 (2010)

[113] Yu L, Shih HM, Pfund M, Carlyle WM, Fowler JW (2002)
Scheduling of unrelated parallel machines: an application to PWBmanufacturing.
IIE Trans 34, pp. 921–9312.

[114] Hsieh JC, Chang PC, Hsu LC (2003)
Scheduling of drillingoperations in printed circuit board factory.
Comp Industr Engin44 pp. 461–473.

[115] Kim DW, Kim KH, Jang W, Chen FF (2002)
Unrelated parallelmachine scheduling with setup times using simulated annealing.
Robo Comput-Integr Manuf 18 pp.223–231,

[116] Pinedo, M. 2008.
***Scheduling Theory, Algorithms, and Systems*.**
3rd ed. New York: Prentice Hall.

[117] Potts, C. N. 1985.
Analysis of a Linear Programming Heuristic for Scheduling Unrelated Parallel Machines.
Discrete Applied Mathematics 10: pp. 155–164.

[118] Pfund, M., J. W. Fowler, and J. N. D. Gupta. 2004.
A Survey of Algorithms for Single and Multi-objective Unrelated Parallel-machine: Deterministic Scheduling Problems.
Journal of the Chinese Institute of Industrial Engineers 21, pp. 230–241.

[119] Gravel, M., Price, W.L., Gagné, C.
Scheduling jobs in an Alcan aluminium foundry using a genetic algorithm
(2000) International Journal of Production Research, 38 (13), pp. 3031-3041.

[120] Dolgui, A., Eremeev, A.V., Kovalyov, M.Y., Kuznetsov, P.M.
Multi-product lot sizing and scheduling on unrelated parallel machines
(2010) IIE Transactions (Institute of Industrial Engineers), 42 (7), pp. 514-524.

[121] Fu, L.-L., Aloulou, M.A., Triki, C.
Integrated production scheduling and vehicle routing problem with job splitting and delivery time windows
(2017) International Journal of Production Research, 55 (20), pp. 5942-5957.

[122] Afzalirad, M., Rezaeian, J.
A realistic variant of bi-objective unrelated parallel machine scheduling problem: NSGA-II and MOACO approaches
(2017) Applied Soft Computing Journal, 50, pp. 109-123.

[123] Baykasoğlu, A., Ozsoydan, F.B.
Dynamic scheduling of parallel heat treatment furnaces: A case study at a manufacturing system
(2018) Journal of Manufacturing Systems, 46, pp. 152-162.

[124] K. Deb
A fast and elitist multiobjective genetic algorithm: NSGA-II
IEEE Transactions on Evolutionary Computation, 6 (2) (2002), pp. 182-197

[125] J.E.C. Arroyo, V.A. Armentano
A partial enumeration heuristic for multi-objective flowshop scheduling problems
Journal of the Operational Research Society, 55 (9) (2004), pp. 1000-1007

[126] E. Taillard,
Some efficient heuristic methods for the flow-shop sequencing problem
European Journal of Operational Research, 47 (1) (1990), pp. 65-74

[127] Muhammad Nawaz, E Emory Enscore, Inyong Ham,
A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem,
Omega, 1983, Pages 91-95,

[8] Appendix

8.1 Pseudocode of initialization phase

```
%% INITIALIZATION PHASE

% Declare number of objectives and greedy phase counter

n_obj= set the number of objectives;
n_dec_gp= 0;

% NEH_EDD is used to generate two initial solutions, one for
% each objective to optimize, which will create the CWS

[Current working set or CWS] = NEH_EDD (decoding method, input data);


% decoding of CWS
% initialize counter n_eval

n_eval=0;

for i = 1: length(CWS)

      [CWS] = decoding (CWS, input data);

end

%Greedy phase of ISS
%update counters


for i= 1: length(CWS)

   [CWS, n_dec]= greedy_phase (CWS, input data);
   n_eval = n_eval+1 ;
   n_dec_gp = n_dec_gp + n_dec;

end

%put obtained solutions in the current working set
CWS = initial solutions

% filters the doubles
CWS = CWS without doubles;

% keep the non dominated solutions.
CWS= paretofront(CWS)
```

8.2 *Pseudocode of MCDA*

```
function [d] = MCDA_(X, N_eval)

% X is the solution set
% N_eval is the vector of counters
% d is the MCDA value

% return the number of points and number of objectives
[DimSet, n_obj] = size(X);

% Define the whole matrix PS
% the last column as a point index

PS = [X N_eval zeros(DimSet,4) [1:DimSet]'];


for m = 1,…,n_obj

    PS= sort_rows(PS(m));
    PS_dist=PS(:,4)';


    if DimSet>2

        % save one line
        PS([1,end], m+3)=-1;

    else

        % To tackle the special case where DimSet = 2
        PS([1, end], m+3)= 1;

    end

    for i=2,…,DimSet-1

        fmax = max(PS(:,m));
        fmin = min(PS(:,m));

        PS_dist(i)= PS_dist(i) + ((PS(i+1,m)-PS(i-1,m))/(fmax - fmin));


        PS(i, m+3)= PS_dist(i);

    end

end

PS( PS(:,5) == -1, 5) = max(PS(:,5));
PS(:,6) = PS(:,5) + min( max(PS(:,5),0));

% return to the original order
PS(:,7) = PS(:,6)./(PS(:,3)+1);
PS= sort_rows(PS, 8);
d = PS(:,end - 1);

end
```

8.3 Pseudocode of Greedy phase

```
%% GREEDY PHASE

% define the k value, that is the number of elements to remove
k= 5;

% destruct function outputs are π_d, π_r
[π_d, π_r] = destruct(selected solution, k);

% construct function. It outputs a set of non-dominated
% solutions to add to the CWS
[SET]= construct(π_d, π_r, data input);

% count the number of decodings
n_dec= count number of decodings
```

8.4 Pseudocode of local search

```
function [LS]= Local_search(Sol, n_sel, n_neigh, data input)

% Sol is the selected solution for the local search
% n_neigh is the number of positions a element could be moved to the left or to the
%right

% Pos is a n_sel array which stores the positions of the elements to
%remove
Pos = random_array(1 length(Sol));


for i=1,…,n_sel


    if Pos(i) <= n_neigh/2   then

        n_left = Pos(i)-1;
        n_right = n_neigh-n_left;

    else
        if Pos(i) > length(Sol)-n_neigh/2    then

            n_right = length(Sol) - Pos(i);
            n_left = n_neigh - n_right;

        else

            n_right= round(n_neigh/2);
            n_left= round(n_neigh/2);

        end
    end

% Remove job in position Pos(i)
Sol_new =Job_remove(Sol, Pos(i));
```

```
% Insert the removed element in Pos(i)-j
    for j=1,…,n_left

        X_left(j,:) = Job_insert(Sol_new, Sol(Pos(i)),  Pos(i)-j);

    end

 % Insert the removed element in Pos(i)-j
    for j= 1:n_right

        X_right(j,:) = Job_insert(Sol_new, Sol(Pos(i)), Pos(i)+j);

    end

%store in LS, n_sel x n_neigh insertions
LS= [LS; X_left; X_right];

end

% Evaluation and final solution of LS
% initialize counter
for i=1,…,number_of_solutions_in_LS


     [TotTard, SPS] = decoding(LS(i), data input);
     n_eval = 0;

end


  PF= paretofront(LS));
  LS=LS(PF);


End
```