UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

TADEU KNEWITZ ZUBARAN

A Study on Shop Scheduling Problems

Ph.D. thesis

Advisor: Prof. Marcus Ritt

Porto Alegre
April 2018

“One of the painful things about our time is that those who feel certainty are stupid, and those with any imagination and understanding are filled with doubt and indecision.”

—

Bertrand Russell. (1951)

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

ABC	Artificial Bee Colony
DG	Disjunctive Graph
FSS	Flow Shop Scheduling
HSS	Hybrid Scatter Search
ITS	Iterated Tabu Search
JSS	Job Shop Scheduling
JSS-PA	Job Shop Scheduling with Processing Alternatives
JSS-PM	Job Shop Scheduling with Parallel Machines
MSS	Mixed Shop Scheduling
NSP	Neighbourhood Search Procedure
OSS	Open Shop Scheduling
PSO	Particle Swarm Optimization
PSS	Partial Shop Scheduling
PFSS	Permutation Flow Shop Scheduling

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Shop scheduling is a combinatorial optimization type of problem in which we must allocate machines to jobs for specific periods time. A set of constraints defines which schedules are valid, and we must select one that minimizes or maximizes an objective function. In this work we use the makespan, which is the time the last job finishes.

The literature contains several studies proposing techniques to solve shop problems such as the job shop and open shop. These problems allow the steps of the production processes to be either fully ordered or not ordered at all. With increasing complexity and size of industrial applications we find, more recently, several works which propose more general shop problems to model the production processes more accurately. The mixed shop, group shop and partial shop are examples of such problems.

In this work we propose an iterated tabu search for the partial shop, which is a general problem and includes several other more restrictive shop problems. The most important novel components of the solver are the initial solution generator, the neighbourhood, and the lower bound for the neighbourhood. In computational experiments we were able to show that the general partial shop solver is able to compete with, and sometimes surpass, the state-of-the-art solvers developed specifically for the partial, open, mixed and group shops.

Sometimes a machine is a bottleneck in the production process, and is replicated. In the literature the parallel machines case has being included in several extensions of shop problems. In this thesis we also propose a technique to schedule the parallel machines heuristically, without including them explicitly in the representation of the problem. We use general techniques for the non-parallel machine cases to produce a fast tabu search heuristic results for the job shop with parallel machines.

Keywords: Shop Scheduling. Heuristics. Tabu Search. Iterated Greedy. Partial Order. Parallel Machines.

RESUMO

Escalonamento de processos é um tipo de problema de otimização combinatória no qual devemos alocar máquinas à tarefas por períodos específicos de tempo.

A literatura contém diversos estudos propondo técnicas para resolver modelos de escalonamento de processos como o job shop e o open shop. Esses modelos permitem que os passos no processo produtivo sejam ou completamente ordenados ou sem ordenação alguma. Com o aumento da complexidade das aplicações industriais no encontramos, mais recentemente, diversos trabalhos que propõe problemas de escalonamento de processos mais gerais para modelar mais precisamente os processos produtivos. O mixed shop, group shop e partial shop são exemplos de tais modelos.

Nesse trabalho nós propomos uma busca tabu iterada para o partial shop, que é um modelo geral que inclui diversos modelos mais restritivos. Os componentes novos mais importantes da técnica são o gerador de solução inicial, a vizinhança e o limite inferior para a vizinhança. Em experimentos computacionais nós conseguimos demonstrar que a heurística genérica e única é capaz de competir, e as vezes superar, as técnicas de estado de arte desenvolvidas especificamente para partial, open, mixed e group shop.

Algumas vezes uma máquina é o gargalo de um processo produtivo, e é replicada. Na literatura o caso das máquinas paralelas foi incluído em diversas extensões de problemas de escalonamento de processos. Nessa tese nós também propomos uma técnica para escalonar as máquinas paralelas, sem incluí-las explicitamente na representação do problema. Nós usamos técnicas gerais para os casos sem máquinas paralelas para produzir uma busca heurística tabu rápida, e estado da arte, para o caso do job shop com máquinas paralelas.

Palavras-chave: Escalonamentos de Processos, Heurística, Busca Tabu, Ordem Parcial, Máquinas Paralelas.

1 INTRODUCTION

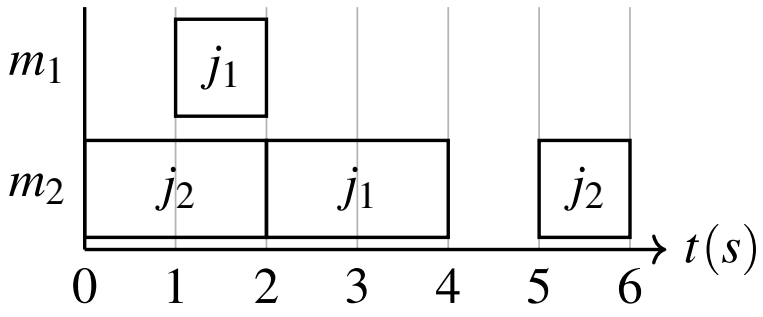
Combinatorial optimization is a topic that consists of finding good solutions, ideally an optimal solution, from a set of possible feasible solutions. A feasible solution is one that obeys a set of constraints, and the quality of the solutions is measured by an objective function.

Scheduling is a decision-making process that deals with the allocation of resources to tasks for specific periods of time. The goals is to optimize the process according to some objective function. Shop scheduling problems consist of machines and operations. The operations are partitioned into jobs, and must be processed on the machines. Commonly some precedences between the operations of the same job is enforced. A solution is a schedule, where for each operation we provide an allocation of one or more time intervals to one or more machines, which can be represented by a Gantt chart. Shop models are widely used in many manufacturing and services industries [68, 69].

In Figure 1.1 we see an example of a Gantt chart showing a schedule with two machines and two jobs. Job j_1 starts the execution on machine m_1 at $1s$ and occupies it for $1s$, while on machine m_2 it starts at $2s$ and finishes at $4s$. Job j_2 does not use machine m_1, but it is scheduled twice on machine m_2, from $0s$ to $2s$, and from $5s$ to $6s$.

In many shop problems it is impractical to perform an exhaustive search, and the alternative is to use more refined methods. For decades much effort has being put into developing techniques to solve shop problems such as flow shop problems, job shop problems and open shop problems. The famous job shop instance with ten machines and jobs proposed by Fisher and Thompson [21] was solved optimally twenty years later by a branch-and-bound algorithm by Carlier and Pinson [14]. With time, the scale and intricacies of the real-world applications has increased, Voß and Witt [86] presents a German steel manufacturer case with 30,000 jobs. In this thesis we study several shop problems, and propose techniques to optimize the scheduling of these problems.

Figure 1.1: Example of a Gantt chart.



1.1 Formal Definition of the Shop Models

We introduce some notations and definitions to show the mathematical model which will be used as a framework throughout this work.

Let $\mathscr{O} = \{1, \ldots, O\}$ be a set of operations, which are sometimes called tasks, $\mathscr{J} = \{j_1, \ldots, j_n\}$ be the set of jobs, and $\mathscr{M} = \{m_1, \ldots, m_m\}$ be the sets of machines. Each job is a set of operations. Jobs are are sometimes called commodities, job lots, or production lots. Each machine is also a set of operations, and can be called facility, work station, or (process) stage [38]. Each operation is associated with an unique machine and a job, therefore both the set jobs and the set of machines are partitions of the set operations

$$\mathscr{O} = j_1 \cup \ldots \cup j_n$$

and

$$\mathscr{O} = m_1 \cup \ldots \cup m_m.$$

From these partitions we have two induced total functions. Let $\mathfrak{j} : \mathscr{O} \to \mathscr{J}$ be the function that maps an operation o to the job $j_{\mathfrak{j}(o)}$ that contains it, and let $\mathfrak{m} : \mathscr{O} \to \mathscr{M}$ be the function that maps it to the machine $m_{\mathfrak{m}(o)}$ that contains it .

In some cases the machines are replicated, and partitioned into stages. A stage is comprised of an ensemble of k identical machines. In this case $\mathscr{M} = \{m_{11}, m_{12}, \ldots, m_{1k}, m_{21}, m_{22}, \ldots, m_{mk}\}$ is the set of machines, where $\{m_{s1}, m_{s2}, \ldots, m_{sk}\}$ is the set of machines of stage s.

Most problems we address in this thesis do not allow recirculation. That means that no two operations of the same job can be on the same machine. In this case for a job j_j and machine m_m we uniquely identify an operation, which we call o_{jm}.

A partial order $\prec$ may be given for the operations, such that if o precedes o' we write $o \prec o'$.

Each operation o has a processing time p_o, also represented as p_{jm} where $\mathfrak{j}(o) = j$ and $\mathfrak{m}(o) = m$.

A solution for the problem is a schedule, which is an assignment of the operations to the machines for specific periods of time. A valid schedule is one where

- each operation $o \in \mathscr{O}$ is scheduled on its associated machine $\mathfrak{m}(o)$ for p_o units of time,
- each operation is processed uninterrupted (also called non-preemptive scheduling),
- no two operations of the same machine are being processed at the same time,
- no two operations of the same job are being processed at the same time,
- if $o \prec o'$, then o' starts to execute after o is finished.

Therefore a valid schedule can be defined by the starting or finishing time of each operation. We call S_o and C_o the starting and completion time, respectively, of operation o in a schedule. We do not allow for missing operations, therefore the functions $\mathfrak{j}$ and $\mathfrak{m}$ are surjective. We also do not allow for operations with null processing time.

In case of parallel machines, we modify the conditions for a valid schedule such that an operation o can be scheduled on any of the k replicated machines of its stage. A stage may process more than one operation at a time, but a single machine cannot.

The objective is to find a valid schedule that optimizes some objective function. Examples are flow time which is the sum of the completion time of the last operation of each job, weighted flow time which is similar but different jobs have different weights in the objective function, tardiness where we minimize the delay respective to due dates give for each job, and the makespan where the objective function is simply the time the last operation is competed. The makespan is the most common objective function in the literature and this work focuses on it.

1.2 Specification of the Shop Models

The General Shop Scheduling Problem is an umbrella term for the shop problems. It is the case where there may be any kind of precedence relations between any operations, including between operations of distinct jobs.

1.2.1 Partial Shop Scheduling Problem and its Special Cases

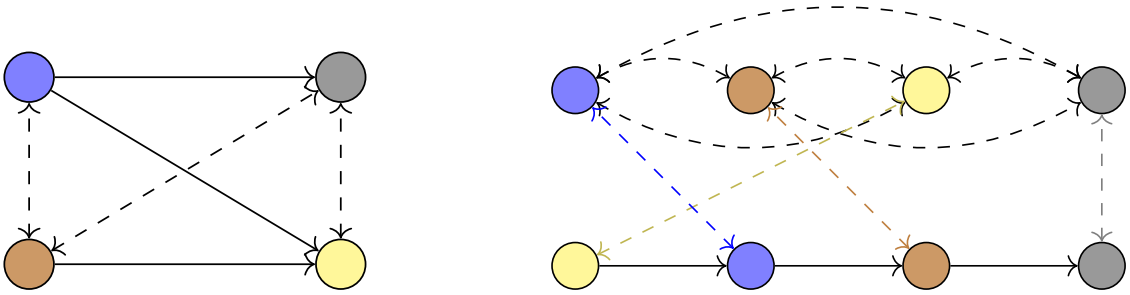
The *Partial Shop Scheduling* (PSS) is the particular case of the General Shop Scheduling problem where only operations in the same job may have precedences and for each job the operations must obey a given partial order precedence relation.

The most well studied special cases of the PSS problem are the *Flow Shop Scheduling* (FSS), the *Job Shop Scheduling* (JSS), and the *Open Shop Scheduling* (OSS) problems. In all these problems no precedence relation exists between operations of distinct jobs. For FSS the operations each job must be processed on all machines in a particular linear order, and this order is the same for all jobs. For JSS each job is also processed on the machines in a linear order, but the order for a job may be different than the others. For OSS the operations can be processed in any order. In these problems it is common in the literature to add the restriction that no recirculation is allowed.

In the literature we find further special cases of the PSS problem. The *Mixed Shop Scheduling* (MSS) is a scheduling problem where we combine JSS and OSS. Each job follows the scheme of one of the two problems. The operations can be processed in any order or in one particular linear order. Formally MSS is the case of PSS such that the relation $\prec$ is either empty or a total order.

In a *Group Shop Scheduling* (GSS) problem, the operations of a job are partitioned into stages. The operations of each stage are independent, and the stages are linearly ordered. Operations of a stage can only begin processing when all operations of the previous stage have finished.

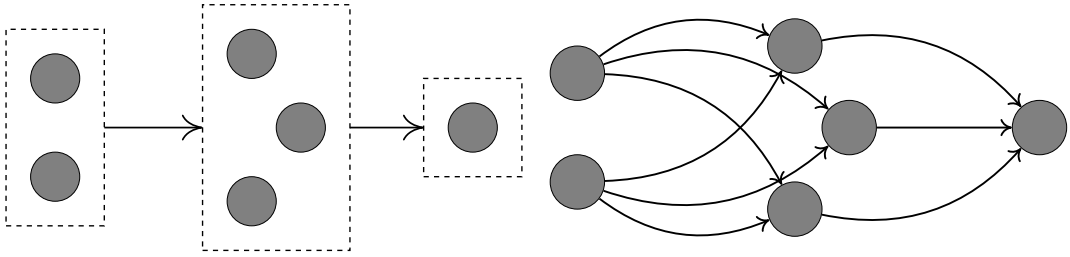
All these problem variants are NP-hard, since they either generalize either the OSS or the



(a) The smallest PSS instance that is not a GSS instance.

(b) Example of a MSS instance with two jobs and four machines.

Figure 1.2: Examples of instances of the different problems. Solid arcs represent precedences, dashed double-arcs independent operations.



(a) A GSS instance.

(b) Associated PSS instance.

Figure 1.3: Transformation of GSS (Multi-Component Scheduling) instance into a PSS instance.

JSS, and both are NP-hard [66].

The main difference between them lies in different flexibility of modeling production processes, due to the different structures of the ordering of the operations in each job. In Figure 1.2a we see the smallest example of a PSS instance that cannot be modeled as a GSS instance, since there is no selection of groups for the operations that yields the same precedences, and consequently can also not be modeled as an MSS instance. Figure 1.3a shows a GSS instance. As mentioned above it can be modeled as a PSS instance (as shown in Figure 1.3b), but it cannot be reduced to a MSS instance, since its order is neither empty nor full. Finally, an example of an MSS instance is shown in Figure 1.2b. Industrial applications of the PSS can be found in electronic and auto repair, logistics, and employee work scheduling [41], in fire engine assembly plants [44], in production processes which need different assembly stages which in turn use different shop problems [42]. The different flexibility has also consequences for secondary problem characteristics. When modeling a given production process by a PSS, the search space size will be larger than the one of a corresponding GSS problem, which in turn will be larger of a corresponding MSS problem. Similarly, the order strength will in general increase from the PSS to the GSS. Clearly, problem characteristics related to processing times such as job or

machine loads will not be affected by the model.

1.2.2 Permutation Flow Shop Scheduling Problem

Permutation Flow Shop Scheduling (PFSS) adds the restriction to the FSS problem with no recirculation that, for all machines, the operations of each job must be processed in the same order. This means that the valid schedules are only *permutation schedules*. In a permutation schedule, all jobs have to be processed on all machines in the same order, which reduces the number of possible solutions to $n!$ A valid solution for PFSS is always a valid solution for the associated FSS problem, therefore the optimal solution for a PFSS is an upper bound for the associated FSS problem.

1.2.3 Job Shop Scheduling Problem with Parallel Machines

The *Job-Shop Scheduling Problem with Parallel Machines* (JSS-PM) extends the JSS and introduces k parallel identical replicas of each machine. It is not a particular case of the general shop, which does not include parallel machines. A valid solution will generate a schedule, that may use any of the parallel machines but, like in the previous problems a single machine may not process more than one operation at any given time. For this problem the formal definition must be changed such that to define a valid schedule, for each operation, we must also choose one of the parallel machine to allocate it. The JSS-PM problem is NP-Hard in the strong sense and since the JSS is a particular case of JSS-PM with $k = 1$, JSS-PM also is strongly NP-Hard NP-hard [66].

1.3 The contribution of this Thesis

In this work we study several shop problems, their relations, and solutions to such problems. We focus on PFSS, JSS-PM and mainly on PSS and its sub-cases.

The main contribution of this work are the techniques used in the proposed algorithm for PSS, namely the initial solution generator, neighbourhood structure, and the lower bound for new solutions generated by such neighbourhood. We highlight the large scope of different shop problems for which the proposed solver performs well. We compared our results with the state-of-the-art solvers designed specifically for PSS, but also for GSS, MSS and OS. Therefore we defend the thesis that we can solve PSS and several particular cases of this problem with a general solver and obtain a solution quality on par with the state-of-art with equivalent computational effort.

The secondary contributions are the novel components associated with the algorithm for JSS-PM, namely the initial solution generator, the neighbourhood and the scheduling technique that ignores parallel machines and enables the heuristic to solve a JSS-PM instance without

increasing the search space due to the parallel machines.

We also present some minor contributions: We organized and studied many shop problems, according to their relations and the properties of their instances. We proposed a simple, but effective, solver for PFSS. For PSS we performed a small study of the influence of the different components for constructing an heuristic solver for the problem. For the parallel machines we also proposed a simple learning technique to improve the speed of the heuristic for JSS-PM, and a *mixed integer programing* model which can be used to solve small instances of the problem exactly.

2 RELATED WORK

In this section we present the relevant literature on the shop problems studied in this thesis. We consider the problems JSS, PSS, MSS, GSS, OSS, PFSS, JSS-PM more important, and we present a more comprehensive review. For the other problems we focus on their definition, most important works and the differences between the problems. At the end of the section we discuss the relation between the presented shop scheduling problems.

2.1 Partial Shop Scheduling and Related Models

We explained the PSS problem in the previous section, and it encompasses several other more restrictive problems. Here we show the works related to PSS and to the most important particular cases of it. We present first the problems we work with directly, then the related problems. In those two categories we present them in order of the most general to the most restrictive.

For a comprehensive overview of the work on scheduling problems, in particular for JSS, OSS and FSS problems we refer to the survey by Potts and Strusevich [70].

2.1.1 Partial Shop Scheduling

Nasiri and Kianfar [60] propose the PSS problem as a natural extension of both JSS and OSS. It is defined such that it can model the practical problems encountered in industry more accurately. The authors define a mixed-integer programming formulation, and propose a hybrid scatter search, coupled with a tabu search, to generate heuristic solutions of larger instances in a timely manner. To test the algorithms Nasiri and Kianfar [60] propose a set of instances for the PSS problem, which are generated using the processing times of the instances for the JSS proposed by Taillard [81], with a randomly generated partial order of the jobs. Zubaran and Ritt [90] propose a iterated greedy algorithm and test the quality of the solver on this instance set, which improved the quality of the solutions for this test set.

2.1.2 Group Shop Scheduling

Several authors focus on the GSS problem, which was introduced by Blum and Sampels [7]. Blum and Sampels [7] propose an ant colony optimization heuristic and define a set of benchmark instances for GSS. They generalize the neighbourhood of Nowicki and Smutnicki [62] with the concept of stage blocks (or group blocks) which takes into account the fact that we can change the execution order of operations at the same stage. To the best of our knowledge the first instance of GSS was proposed earlier, without defining the problem formally, in a competition for children [19]. Liu et al. [50] present a tabu search for GSS, and show em-

pirically that it improves the results of Blum and Sampels [7] on their instances. Nasiri and Kianfar [59] propose a mixed-integer programming formulation and a hybrid genetic algorithm and tabu search heuristic for the GSS problem (which they call the *Stage Shop Problem*). They also define a new set of benchmark instances for GSS. Nasiri [58] proposes an artificial bee colony algorithm, which improves the results of Nasiri and Kianfar [59].

2.1.3 Mixed Shop Scheduling

The MSS problem was proposed by Masuda et al. [54]. The authors focus on the particular case of jobs which all have the same order (i.e. the FSS case) and investigate the structure of the problem with two machines. Liu and Ong [49] propose a neighbourhood for MSS which is a combination of three smaller neighbourhoods based on the critical path. They use the neighbourhood in three different heuristics, of which a tabu search performed best. The authors also define a set of instances, which combines JSS with OSS.

2.1.4 Open Shop Scheduling

An early development for the OSS problem is the branch-and-bound algorithm from Brucker et al. [10]. In this paper the authors also propose a set of benchmark instances for the problem. Guéret and Prins [29] propose two constructive heuristics which produce better solutions than the previous solvers. More recently several meta-heuristics have been proposed for OSS: Liaw [47] designed a tabu search algorithm, Liaw [48] a simulated annealing algorithm, Liaw [48] and Prins [72] genetic algorithms and Blum [6] an ant colony optimization heuristic. Currently the best performing meta-heuristic is the particle swarm optimization proposed by Sha and Hsu [77]. It uses a greedy decoding scheme, based on a constructive heuristic that produces only non-delay schedules. The authors modify the decoding technique by introducing a relaxation, which allows allows the heuristic to produce several distinct possible schedules. The authors combine this heuristic with a beam search technique, to select on of schedules that can be generated by the decoding scheme. Dorndorf et al. [18] proposed a branch-and-bound solver for the OSS problem, which is currently the best exact solver for OSS.

2.1.5 Job Shop Scheduling

The literature on JSS is extensive, and here we highlight some of the most important contributions to the research. Several works use neighbourhoods based on a transposition of adjacent operations on a critical path. Early works that use this neighbourhood structure are [85] and [83]. Nowicki and Smutnicki [62] developed a tabu search for the JSS problem, and the most innovative element of the algorithm is the neighbourhood, later called N5 neighbourhood, which also performs transpositions of successive operations in a critical path, but significantly

decreases the size of the neighbourhood by restricting the transpositions by removing transpositions that are guaranteed to not immediately decrease the makespan. This neighbourhood consist on the states generated only by transpositions of pairs of operations on the borderline of the so-called blocks, which consists of successive operations in a critical path that share the same machine. Adams et al. [1] proposed a constructive heuristic for the JSS problem called the Shifting Bottleneck Procedure. It constructs a schedule by repeatedly inserting a bottleneck machine into the current partial schedule, and re-optimizing all previously scheduled machines. The bottleneck machine is found by solving a single machine problem with release dates and delivery times for each unscheduled machine, and selecting the machine of longest makespan. The re-optimization applies the same strategy to all scheduled machines. The single machine problem is also NP-Hard, but usually can be solved in a timely manner with the branch-and-bound algorithm developed by Carlier [13]. Balas and Vazacopoulos [4] proposed a heuristic that combines the Shifting Bottleneck Procedure with a local search. The local search uses a new neighbourhood, later called N6, which performs backward and forward shifts of operations in a block of the critical path to the edges of such block. N5 is a subset of the N6 neighbourhood. Zhang et al. [87] proposed a tabu search, and further extended the N5 and N6 neighbourhoods by allowing moves from the edge of the blocks to an internal position. Peng et al. [65] combined this tabu search with a path relinking procedure to produce one of the current state-of-the-art solvers for JSS. Akers Jr [2] presents an exact graphical method for the 2-job JSS, which transforms the problem into a shortest path problem in the plane. Gonçalves and Resende [27] extended the graphical method as an heuristic, by starting with only two jobs and inserting the remaining jobs into the solution one by one. The authors used this method in a biased random key genetic algorithm producing one of the current state-of-the-art heuristic solvers.

2.1.6 Other Partial Shop Related Models

Strusevich [78] proposes the *Super Shop Scheduling* problem. As in MSS all jobs are either OSS jobs or JSS jobs. Additionally, there may be two kinds of precedences between jobs. In the first kind, a job cannot start processing on a machine, before its predecessor has finished processing on the same machine. This is the only kind of precedence in most other shop problems. In the second kind, a job cannot start processing before its predecessor has finished processing completely. The authors proofs the NP-Hardness of the problem and some sub-cases of it, a lower bound for the solutions of the super shop, and polynomial solutions for some particular cases of the problem.

The *Multi-Component Scheduling*, proposed by Kleeman and Lamont [41], generalizes the GSS problem. It combines two other shop problems, one of which works as the outer structure, and the other as the inner structure. To create a Multi-Component Scheduling instance combining, for instance, OSS as the outer structure and JSS as the inner structure we start with a

regular OSS instance, and we substitute each operation with a JSS shop. In this case there is no order of the outer operations because it follows the OSS structure, but the inner operations in each of them will be linearly ordered following the JSS structure. The inner operations are the operations that are scheduled in the machines. When an inner operation of an outer operation is scheduled on a machine, then the outer operation is considered to be executing. Like inner operations, outer operations cannot be executed concurrently, and must be executed according to the order its structure. When we combine a JSS as the outer structure, and OSS as the inner structure of a Multi-Component Scheduling we have a GSS.

Kis [40] defines the *JSS problem with Processing Alternatives*. In this problem the routings of the jobs are defined by a subclass of directed acyclic graphs. A graph with a single operation belong to this class, and more complex graphs are constructed recursively by three possible operations which combine two or more previously defined subgraphs into a new graph. In a *sequence* of two subgraphs one subgraph precedes another, and the second one may only be executed after the first one finishes. In an *And-subgraph* of any number of subgraphs all of them can be processed in any order, but a successor of the resulting graph may only start processing after all subgraphs have finished. In an *Or-subgraph* of any number of subgraphs exactly one them must be executed before any succeeding subgraph.

2.2 Permutation Flow Shop Scheduling

The PFSS problem is a widely studied variation of the FSS problem. For more than two machines the PFSS is a NP-hard problem [39] and since the seminal work by Johnson [35] who proposed a polynomial algorithm for the case $m = 2$, it has been studied thoroughly. Instances of a size useful in actual applications are usually not exactly solvable, therefore several constructive and improvement heuristics have been proposed. In particular we have the simple and effective constructive heuristic NEH [61] which is often used to generate initial solutions for more sophisticated approaches. Taillard [79] has shown that the NEH heuristic can be implemented in time $O(n^2m)$. It is well known that tie-breaking rules are important for the performance of NEH-like heuristics. Kalczynski and Kamburowski [37] study such rules and propose the improved constructive heuristic NEHKK1. Extensions of the NEH-like heuristics, which try to overcome the fixed job order, have been studied by Farahmand et al. [20].

Framinan et al. [22] considers adaptations of the NEH heuristic when the objective is not to minimize the makespan. Other works with notable results are the ant colony algorithm of Rajendran and Ziegler [73], the CDS algorithm [11], the work from Dannenbring [16] which performs a constructive heuristic followed by an improvement phase, the iterated greedy algorithm from Ruiz and Stützle [75], and the tabu search of Nowicki and Smutnicki [63]. For more details on meta-heuristic approaches to solve the PFSS problem, we refer the reader to the excellent surveys of Gupta and Stafford [31] as well as the aforementioned shop scheduling survey by Potts and Strusevich [70].

2.3 Job Shop Scheduling Problem With Parallel Machines and Related Models

JSS-PM introduces parallel identical replicas of each machine. The JSS is NP-Hard in the strong sense Garey et al. [23] and since the JSS is a particular case of JSS-PM with $k = 1$, JSS-PM also is strongly NP-Hard. Unlike the JSS the literature on JSS-PM is scarce. Rossi and Boschi [74] provide a set of instances for the JSS-PM. The instances are generated using the well known set of instances Lawrence [43] and replicated the jobs such that the optimal solutions for the JSS are upper bounds for the optimal solutions for the parallel case. The authors propose a hybrid heuristic algorithm which combines a genetic algorithm and an ant colony optimization technique to solve the problem.

Gholami and Sotskov [25] propose an algorithm for JSP-PM that consists of several modules. The main module controls the sequencing of the machines by deciding the relative order of pairs of operations. This sequencing is delivered to the module which produces the schedule. A third module is used by the first one, and it controls the occupation of each machine as the first module decides the priority of the operations.

While Rossi and Boschi [74] utilizes the same number of replicated machines for each stage Gholami and Sotskov [25] proposes instances with a variable number of machines. Gholami and Sotskov [24] extends the model by using release times for the jobs.

2.3.1 Learning good moves

In the solver we propose for JSS-PM we use a component that learns during the search which moves produce the best solutions, and trims the neighbouhood when it is convinced that the other moves are not promising.

The literature contains extensive works that use learning to improve the effectiveness of meta-heuristics. Moll et al. [55] works with the dial a ride problem, with a neighbourhood based on the 2OPT, and the evaluation of the objective function is the bottleneck of the process. The authors propose a reinforcement learning technique to predict the value of the objective function, and were able to improve the performance of the algorithm. Prestwich [71] studies the SAT problem, and the algorithm proposed uses noise which allows the search to decrease the quality of the current solution to improve the diversification of the search. The authors show that using reinforcement learning to dynamically adjust the noise intensity improves the performance of the solver. Benlic et al. [5] proposes a hybrid breakout local search and reinforcement learning approach to the vertex separator problem, the authors use reinforcement learning to decide the number of times the perturbation will be applied, and the chance of selecting one operator over another. Mousin et al. [56] study a feature selection problem. The exploration of the neighbourhood is divided in two steps. First the quality of the solutions is predicted, and later only the most promising candidates are evaluated. The estimation of the quality is computed from the quality of solutions with the same characteristics.

2.3.2 Flexible Job Shop Scheduling

The *Flexible Job Shop Scheduling* problem is an extension of the JSS. In JSS each operation is mapped to a machine through j, whereas in Flexible JSS the assignment of an operation is not fixed, and can be processed on a subset of the machines. The operations may have a different processing time for each machine, so for operation x, p_x is not necessarily constant and may depend on the machine as well. The Flexible JSS is a NP-Hard problem since it generalizes JSS. Chaudhry and Khan [15] present a comprehensive survey about the flexible JSS.

The Flexible JSS was proposed by Brucker and Schlie [9]. The authors define the problem and work with the particular case with two jobs. An instance of such problem is classified into either a partially or totally flexible JSS instance. For the total instances the set of machines is available for all operations, while for partial instances only a proper subset of the machines is available to at least one operation.

Jensen [34] is an early early influential development on the problem, proposed a Genetic Algorithm. Pezzella et al. [67] also proposes a Genetic Algorithm, and was able to improve the state of the art for the problem. Remarkable properties of the technique are: the initial population is done by three dispatching rules where the operations of each job are scheduled one by one. The first rule is random, and the other two give priority to certain operations, one gives it to the job with most work remaining (sum of processing time), while the second gives to the job with most number of operations remaining. The encoding uses coding uses the *task sequencing list* representation proposed by Kacem et al. [36]. The decoding technique may produce invalid solutions and require a repair step. Zhang et al. [88] improved the quality of the solutions with another Genetic Algorithm. The proposed chromosome representation reduces the cost of decoding, and has no repair step.

The current state-of-the-art solver for the Flexible JSS is González et al. [28], which is a scatter search coupled with a path relinking. The algorithm uses the same neighbourhood used in the path relinking, and is inspired by Nowicki and Smutnicki [62].

2.4 Relations of the Shop Models

Most of the problems previously introduced are related to one another, and many of them differ only by the structure of the precedence relations of the operations. The General Shop is usually used as an umbrella term. The PSS problem is the particular case of the general shop where the operations of different jobs cannot precede one another, meaning the jobs are independent.

The Multi-Component Scheduling can be reduced into a PSS by constructing the overall partial order from the outer and inner structures. For instance if the outer structure follows the FSS model and the inner structure follows the OSS, the operations of a single stage will not have precedences among themselves, but will succeed all operations of the previous stages.

GSS is the case of the Multi-Component Scheduling in which the inner structure follows OSS and the outer structure follows JSS. GSS can also be seen as the special case of PSS such that each operation of a stage precedes each operation of the following stage.

In Section 1.2.1 we saw the example of the transformation of a GSS job into a PSS job illustrated in Figure 1.3. It also can be seen as a Multi-Component Scheduling that uses the JSS model for the outer structure and OSS as the inner structure. In Figure 1.3a the stages are represented by the dashed boxes, and in Figure 1.3b we see the associated PSS job. MSS is a special case of GSS where either a single stage contains all operations, or each stage contains only one operation. Formally, for a GSS problem, for each j_j we have a set of stages $S_{j1},\ldots,S_{j,k(j)}$, which partitions the set of operations J_j. The partial order associated with a job j_j is $\prec_j = S_{j1}\times S_{j2}\cup S_{j2}\times S_{j3}\cup\cdots\cup S_{j,k(j)-1}\times S_{j,k(j)}$, and the partial order for the problem is $\prec=\prec_1\cup\prec_2\cup\cdots\cup\prec_n$.

JSS and OSS are the cases of MSS in which all the jobs follow either the JSS or OSS structure. JSS is a particular case of MSS where for all jobs $\prec$ is a total order, and OSS is a particular case where for all jobs $\prec$ is empty.

FSS is well known as a particular case of JSS. PFSS is not a particular case of any of the other problems because of the restriction that only permutation schedules are allowed, which cannot be emulated by the other problems. Despite not being a particular case of any of the presented problems PFSS can be considered a restrictive problem in comparison to the others, because solutions for PFSS are always solutions for the non-permutation case.

JSS-PM is the case of the Flexible JSS where for each operation x p_x is constant on all machines it is allowed to operate, and the machines allowed for each operation in a job is disjoint, and exactly k machines are available for each operation. JSS is trivially the particular case of JSS-PM where $k = 1$.

Super shop differs from MSS only because it has the second kind of precedences. A Super shop where there are no precedences of the second kind is also a MSS instance.

The relation of the problems is shown in Figure 2.1. The main difference between the problems is highlighted in the arrows. In this figure we call JSS with Processing Alternatives "JSS PA".

The flexible JSS and JSS with Processing Alternative are similar but are neither is a particular case of the other. The first allows for alternative machines to process an operation, while the former allows for different processing routes in which operations may be selected to not be executed. This relation is shown as dashed arrows in the figure.

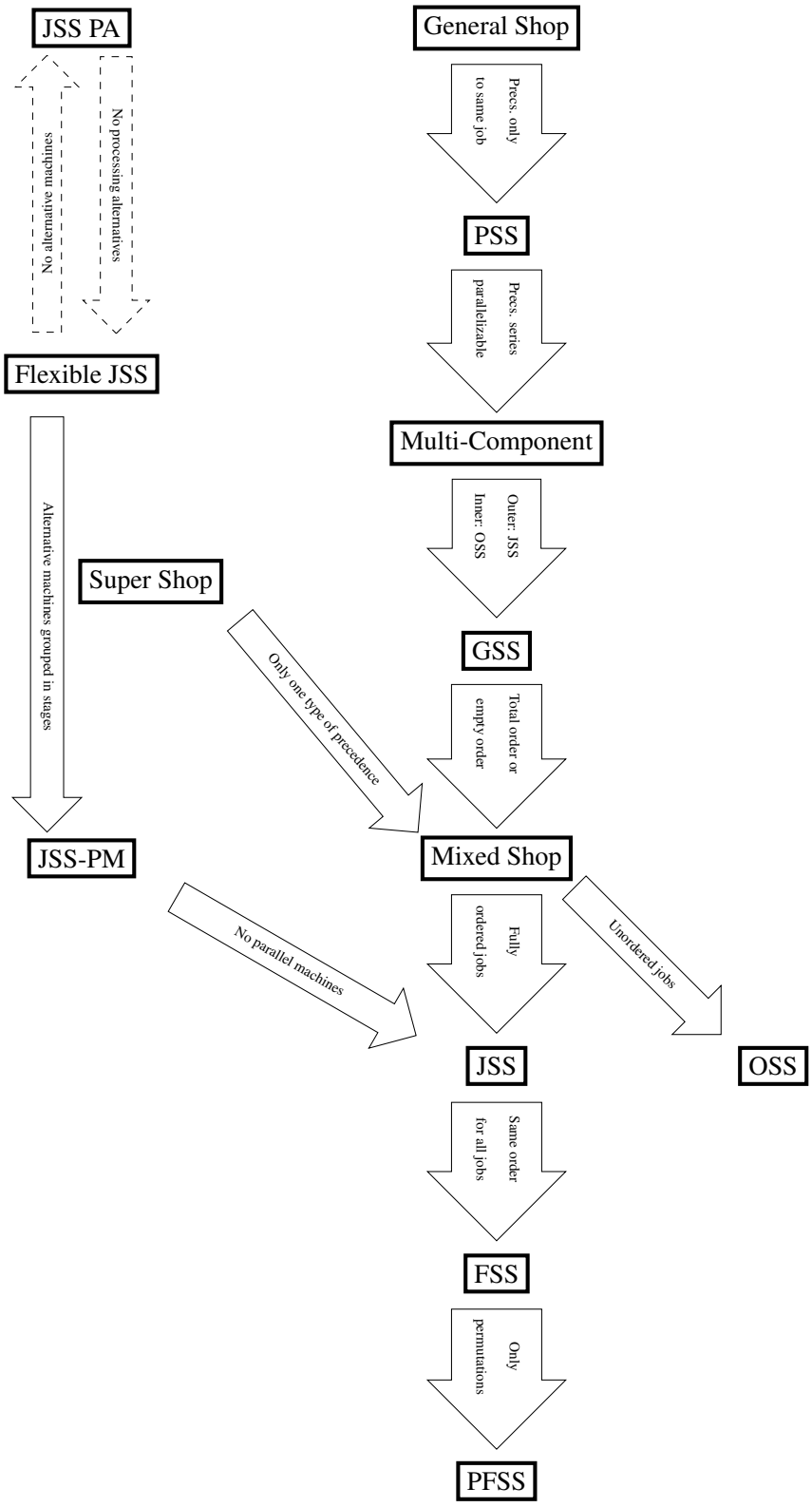


Figure 2.1: Relation of the different shop scheduling problems studied.

3 REPRESENTATION AND PROPERTIES OF THE SHOP MODELS

3.1 Problem Representation

A disjunctive graph $DG = (\mathscr{O}^\dagger, C, D)$ is a useful way to represent an instance of the PSS problem. It is defined over a set of operations $\mathscr{O}^\dagger = \mathscr{O} \cup \{0, *\}$, a set of conjunctive arcs C, and a set of disjunctive arcs D. The operations include an artificial source 0, which precedes all operations, and an artificial sink $*$, which succeeds all operations. Their processing time is defined to be 0. We consider only instances which have strictly positive processing times.

The conjunctive arcs C represent the transitive reduction of the precedence relations among the operations. The set of disjunctive arcs D contains a pair of arcs (u,v) and (v,u) for each pair of independent operations u and v of the same job, or same machine. Note that the disjunctive arcs of each machine form a clique. An orientation of the disjunctive arcs is a selection of one of each pair of these opposing arcs, and is called a *complete selection* if it is acyclic.

3.2 Properties of the Schedules Induced by the Disjunctive Graph

A complete selection defines a unique order of the operations for each machine and each job. A schedule can be derived by starting each operation as soon as its predecessors have finished. The completion time of an operation is equal to the length of the longest path from the artificial source to the operation, where the length of a path is the sum of the processing times of the operations it contains. This creates only semi-active schedules, i.e. schedules in which no operation can be started earlier without changing the order of processing on any one of the machines or jobs. The makespan of a schedule is the completion time of operation $*$.

The set of semi-active schedules corresponds one-to-one to complete selections, and at least one optimal schedule is semi-active. Therefore, there always exists a complete selection which defines an optimal schedule.

A schedule is active if it is impossible to generate any other schedule, through changes in the order of processing on the machines and jobs, with at least one operation finishing earlier and no operation finishing later. Our schedules are not necessarily active, but it is possible to construct any valid active schedule with the proper complete selection of the DG.

3.3 Heads and Tails of the Operations

Given a complete selection we denote by o_j the immediate job predecessor of operation o, and by o^j its immediate job successor. Symmetrically o_m and o^m represent the operation immediate machine predecessor and successor respectively.

For a given operation o, let L be the set of all paths from 0 up to, but not including, o. In DG all operations are reachable from 0, therefore this set is never empty. The length $q(o)$ of

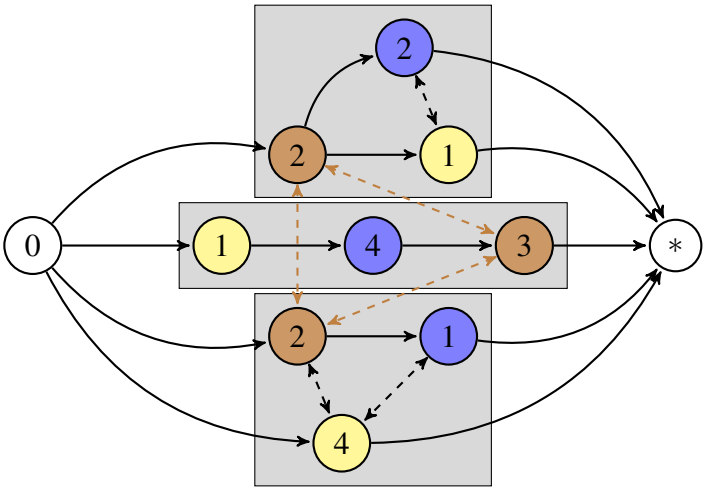


Figure 3.1: Example of a disjunctive graph representation for a PSS problem instance with three jobs (grey boxes) and three machines (yellow, blue, and brown).

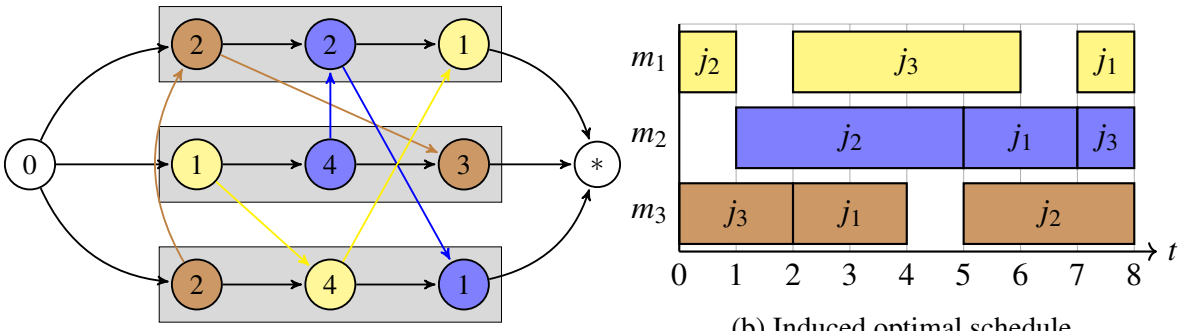
the longest path in L is the so-called head of o, defined by $q(o) = \max\{q(o_j) + p(o_j), q(o_m) + p(o_m)\}$, which is the earliest time o can be scheduled.

Let L_j be the subset of L where the paths also contain o_j. L_j is never empty since o_j is reachable from 0, and, by definition, o_j is directly connected to o in DG. We call the length of longest path in L_j the job head of o $q_j(o) = q(o_j) + p(o_j)$. This is the earliest time o could be scheduled in order to not be processed in parallel with the other operations in the same job. Let L_m be the subset of L where the paths also contain o_m. We also call the length of the longest path $q_m(o) = q(o_m) + p(o_m)$ in L_m the machine head of o, which is the earliest time o could be scheduled in order to no be executed in parallel with other operations in the same machine. The heads of o is therefore $q(o) = \max\{q_j(o), q_m(o)\}$

Symmetrically let L' be the set of paths from, but not including, o to $*$. The length $r(o)$ of the longest path in L' is the so-called tail of o, defined by $r(o) = \max\{r(o^j) + p(o^j), r(o^m) + p(o^m)\}$. Restricting L' to paths which contains o^j we define the job tail $r_j(o) = r(o^j) + p(o^j)$, and restricting L' to paths which contains o^m we define the machine tail $r_m(o) = r(o^m) + p(o^m)$. Analogously to the heads we can see the tails of an operation as the combination $r(o) = \max\{r_j(o), r_m(o)\}$.

Figure 3.1 shows an example of the disjunctive graph of a PSS problem instance with three jobs and three machines. Operations of the same machine have the same color, and operations of the same job are grouped into grey boxes. Conjunctive arcs are drawn solid, and disjunctive arcs are dashed. For clarity the figure shows only the brown arcs associated with the brown machine clique. Each operation o is labeled with its processing time p_o.

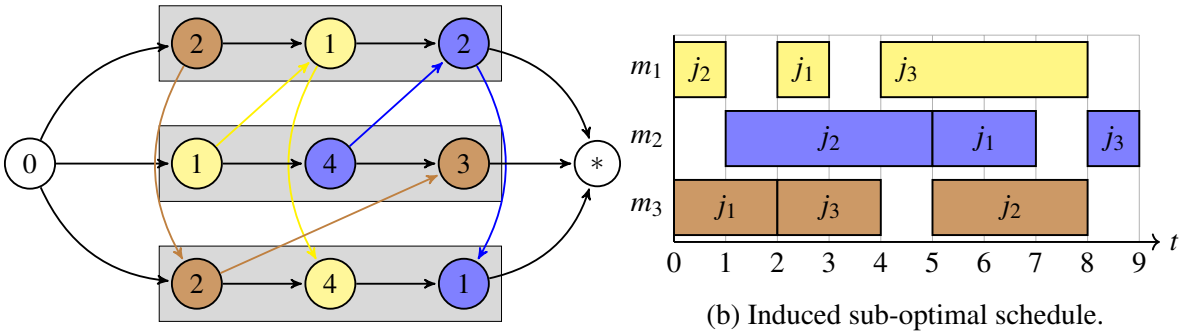
In Figure 3.2 we see a possible solution for the instance depicted in Figure 3.1. Figure 3.2a shows a valid complete selection, and Figure 3.2b shows the Gantt chart of the corresponding induced schedule, which is optimal. An alternative solution, which does not generate an optimal schedule is shown in Figure 3.3.



(a) A valid complete selection.

(b) Induced optimal schedule.

Figure 3.2: Disjunctive graph representation of an optimal solution and the Gantt chart for the associated optimal schedule.



(a) A valid complete selection.

(b) Induced sub-optimal schedule.

Figure 3.3: Disjunctive graph representation of a sub-optimal solution and the Gantt chart for the associated sub-optimal schedule.

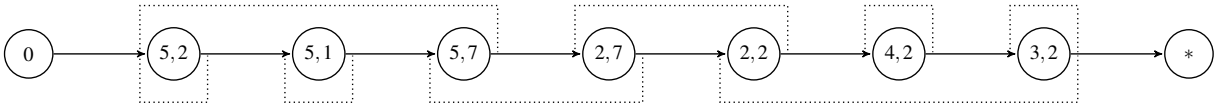


Figure 3.4: A critical path and the associated job and machine block partitions.

3.4 Critical Path and Associated Concepts

We compute the maximum distances in linear time in the number of operations of the instance. Since the processing times are positive, we can ignore transitive arcs in the disjunctive graph, because they belong to no longest path. Thus we compute the maximum distances for all operations in linear time by updating them in a topological order. However we do need the transitive closure to avoid cycles in the generation of the initial solution and repair step of the perturbation shown in the following section. We maintain the transitive closure for each job in a data structure proposed by Italiano [33], such that we can check if there is a path from one operation to another in constant time. We present more detail on how we compute the critical path in sections 4.2.2 and 4.4.3, since the way we work with the critical path is different in the cases with, and without parallel machines.

For a path $P = o_1 \dots o_k$, $o_i \in \mathscr{O}^\dagger$ in S, let $l(P) = \sum_{i\in[k]} p(o_i)$. A critical path P^{crit} is a longest path between, but not including, 0 and $*$ in the disjunctive graph, and the makespan is $l(P^{crit})$. Every longest sequence of operations in a critical path that share a machine or job is a *block* of this path: a *job block* if the job is shared, or a *machine block* otherwise. Thus, both the machine and job blocks form a partition of this path. Each pair of adjacent operations o_1 and o_2 either belongs to the same job block and a different machine block, or belongs to the same machine block and a different job block. Figure 3.4 shows an example of a critical path. The operations are labeled with the associated job and machine, and on the top we show the job block partition, while on the bottom we show the machine block partition. For a given block of two or more operations $o_1, o_2, \dots, o_{x-1}, o_x$ we refer to (o_1, o_2) and (o_{x-1}, o_x) as the *borderline* operations of the block.

Lemma 3.4.1 *$LB_{jb} = \max_{j\in[J]}(\sum_{i\in[m]} o_{ij})$ is a lower bound (the so-called* job lower bound*) for the PSS.*

Proof. The minimum processing time of job j is $\sum_{i\in[m]} o_{ij}$, since the operations cannot overlap. Thus, the maximum over these values is a lower bound for the PSS. ■

Lemma 3.4.2 *$LB_{qr} = \max_{i\in[m]}\{\min_{j\in[n]} q(o_{ij}) + \sum_{j\in[n]} p_{o_{ij}} + \min_{j\in[n]} r(o_{ij})\}$ is a lower bound for the PSS. (LB_{qr} is an improved machine lower bound and used in [60].)*

Proof. The processing on machine m_i cannot start earlier than the least head $\min_{j\in[n]} q(o_{ij})$ of any operation on this machine. Similarly, after processing all operations on m_i the schedule

cannot complete earlier than the least tail $\min_{j\in[n]} r(o_{ij})$ of any operation on m_i. Furthermore, the minimum processing time of the operations on machine i is $\sum_{j\in[n]} p_{o_{ij}}$, since they cannot overlap. Thus, the sum of these three quantities is a lower bound for the PSS with respect to machine i. Therefore the maximum over all machines is a lower bound for the PSS. ■

Proposition 3.4.1 *$LB = \max\{LB_{jb}, LB_{qr}\}$ is a lower bound the PSS.*

Proof. Follows immediately from Lemma 3.4.1 and Lemma 3.4.2. ■

4 PROPOSED ALGORITHMS

In this section we introduce the algorithm designed for both PSS and JSS-PM.

4.1 A Simple Iterated Greedy Algorithm for the Permutation Flow Shop Scheduling

In this section we present the techniques developed for PFSS, which we use on a simple iterated greedy algorithm.

4.1.1 Constructive Heuristics for the PFSS Problem

NEH and its variants are greedy constructive algorithms. For a given job order, they start with an empty schedule and insert the next job in that order into the current partial schedule at the position which maintains the makespan of the current partial schedule shortest. Thus, such an algorithm performs n insertions. When inserting the jth job, the algorithm must determine the makespan of $j-1$ candidate insertion points, which leads to an overall of $\binom{n}{2} = \Theta(n^2)$ makespan computations. Computing a makespan in $O(mn)$ yields a $O(n^3m)$ algorithm. Taillard [80] has shown that the makespan of all insertion points of a fixed job can be computed in time $O(mn)$ which reduces the time complexity to $O(n^2m)$. The basic NEH algorithm as proposed by Nawaz et al. [61] processes the jobs in order of non-increasing total processing times $\sum_{i\in[m]} p_{ij}$, for $j \in [n]$ and is shown in Algorithm 1.

Algorithm 1: NEH

input : Processing times $p_{ij}\ \forall i \in m, j \in n$.
output: A solution
1 let $P_j := p_{j1} + p_{j2} + \cdots + p_{jm}$.;
2 order the jobs such that $P_1 \geq P_2 \geq \cdots \geq P_n$.;
3 let $\sigma = ()$ be the empty sequence.;
4 **for** *all jobs* $j = 1, \ldots, n$ **do**
5 | insert j in σ in the sequence at the position where it yields the lowest makespan.;
6 **end**
7 **return** σ;

Kalczynski and Kamburowski [37] observed that the NEH heuristic can be improved by inserting the jobs in the same order as proposed by Johnson [35] for the two-machine PFSS. If we define

$$\hat{a_j} = \sum_{i\in[m]} \left(\binom{m-1}{2} + m - i \right) p_{i,j}$$

$$\hat{b_j} = \sum_{i\in[m]} \left(\binom{m-1}{2} + i - 1 \right) p_{i,j}$$

then NEHKK1 insert the jobs in order of non-increasing $\hat{c}_j = \min\{\hat{a}_j, \hat{b}_j\}$. Ties between different insertion positions are broken by choosing the first job of minimum makespan, if $\hat{c}_j = \hat{a}_j$, and the last job otherwise. NEHKK1 reduces the average percent relative deviation from the best known values in the instances proposed by Taillard [82] by about 15%. The NEHKK1 heuristic has the additional property that it solves two-machine PFSS optimally. We use the result from the NEHKK1 heuristic as the starting point for the improvement with Bubble Search, as described below.

Farahmand et al. [20] proposed several extensions to NEH-like heuristics. The variants that perform best reinsert the previously inserted jobs at positions $\max\{1, p-k\}, \ldots, \min\{p+k, n\}$ again after the insertion of the jth job at position p, for a parameter k. Their algorithm FRB3 which considers all previously inserted jobs is equivalent to FRB4$_n$. FRB4$_k$ is shown in Algorithm 2. The reinsertion of the jobs is designed to counteract the strong greediness of NEH. FRB4$_k$ has worst case time complexity $O(kn^2m)$.

Algorithm 2: FRB4$_k$

input : PFSS instance
output: A solution
Let $P_j := p_{1j} + p_{2j} + \cdots + p_{mj}$.;
Order the jobs such that $P_1 \geq P_2 \geq \cdots \geq P_n$.;
Let $\sigma = ()$ be the empty sequence.;
for *all jobs* $j = 1, \ldots, n$ **do**
    Insert j in σ in the sequence at the position where it yields the lowest makespan.;
    for *all positions* $i = \max\{1, p-k\}, \ldots, \min\{p+k, j\}$ **do**
        Reinsert the job at position i at the position which yields the lowest makespan.;
    end
end
return σ;

The last algorithm we consider is called *FRB5*, also proposed by Farahmand et al. [20], which consists in the simple idea of performing the algorithm NEH followed by a full local search after the insertion of each new job in the insertion neighborhood N_1. This neighborhood considers all possible reinsertion points of all jobs, similar to FRB3. The local search stops when a local minimum is reached.

4.1.2 Bubble Search and its Variants

Bubble Search and its variants have been proposed by Lesh and Mitzenmacher [45] as simple and flexible modifications procedures for priority-based construction algorithms. For a solution space which consists of all $n!$ permutations like in the PFSS problem, visiting all of them exhaustively will eventually find the optimal solution. The exhaustive search can be performed in any order. In particular, *exhaustive Bubble search* proposes to visit all the permutations in order of increasing Kendall-tau distance from a given permutation. Ties may be broken arbi-

trarily. Formally, for two n-permutations π and π' the Kendall-tau distance, also known as the Bubble Sort distance, is defined as

$$d(\pi,\pi') = \sum_{1\le i<j\le n} [\pi(i) < \pi(j) \text{ and } \pi'(i) > \pi'(j)].$$

It measures the number of transpositions necessary to transform π into π'. Therefore $d(\pi,\pi') \in [0, \binom{n}{2}]$.

The search can be stopped at any time and will return best permutation found so far. The performance of this truncated Bubble Search depends on a good initial order and a problem structure, that makes it more likely that good solutions are close to this initial order, such that it is more likely that we have visited solutions with better quality than random ones.

A natural variation of this proposal is the *randomized Bubble Search*, in which we choose random permutations with a probability that decreases with increasing Kendall-tau distance. Lesh and Mitzenmacher [45] suggest that we use a probability proportional to $(1-\alpha)^{-d}$ where α is a parameter to be adjusted and d the Kendall-Tau distance to the base ordering. For $\alpha = 0$ randomized Bubble Search degenerates into random sampling, and for $\alpha = 1-\varepsilon$ it approximates a sampling in a neighborhood that allows only swaps of two adjacent elements in the base order.

A further improvement of the randomized Bubble Search is the *randomized Bubble Search with replacement*. This strategy replaces the current ordering by the new ordering, whenever the new ordering is better that the current one. This turns randomized Bubble Search into a stochastic search algorithm. Lesh and Mitzenmacher [45] observed to Bubble search with replacement typically outperforms the simpler variants of Bubble Search. They also have demonstrated that Bubble Search can produce results that are competitive with similar GRASP-based algorithms.

4.1.3 An adaptive Variant of Bubble Search

Randomized Bubble Search can be implemented by a simple stochastic process. To select a new ordering, we start with an empty ordering, and process the elements of the current base ordering. With probability α we select the current element, remove it from the base ordering and append it to the new ordering. In this case we start visiting the elements from the start. Otherwise, with probability $1-\alpha$, we continue with the next element of the current base ordering, cycling as necessary. This process is repeated until the base ordering is empty and is shown in Algorithm 3.

This process guarantees that an ordering of Kendall-tau distance d is selected with probability proportional to $(1-\alpha)^d$ [45]. However, since the range of the Kendall-tau distance increases with n, the probability of selecting orderings with a fixed distance from the current ordering decreases with increasing n. We will show below that this behavior can affect the scalability of Bubble Search adversely. We therefore propose to adjust α as a function of n as follows.

Algorithm 3: Bubble Search Sampling

input : A base permutation σ^*, a probability α.
output: A new permutation

Let $\sigma = ()$ be the empty sequence.;
for $i = 1, \ldots, n$ **do**
  $j := 1$;
  while *no element selected* **do**
    With probability α: select element σ^*_j;
    Otherwise: $j := j \mod n + 1$;
  end
  Remove σ^*_j from σ^* and append it to σ;
end
return σ

Consider the probability $P[d = 0]$ that Algorithm 3 returns σ^* on input σ^*. We have

$$\begin{aligned} P[d=0] &= \alpha(1+(1-\alpha)^n+(1-\alpha)^{2n}+\ldots) \\ &\quad \times \alpha(1+(1-\alpha)^{n-1}+(1-\alpha)^{2(n-1)}+\ldots) \\ &\quad \times \cdots \times \alpha(1+(1-\alpha)^2+(1-\alpha)^4+\ldots) \times 1 \\ &= \prod_{i\in[n]} \frac{\alpha}{1-(1-\alpha)^i} = O(\alpha^n). \end{aligned}$$

Therefore, we propose to compensate this behavior by setting $\alpha = \alpha_0^{1/n}$ for some base probability α_0 which has to be calibrated. We call this approach *adaptive randomized Bubble Search*, which can be applied with or without replacement.

4.2 Iterated Tabu Search for Partial Shop Scheduling Problem

In this section we propose an Iterated Tabu Search (ITS) algorithm for the PSS problem. An ITS is similar to an iterated greedy algorithm, which is a meta-heuristic proposed by Ruiz and Stützle [76], but substitutes the local search by a tabu search. The structure of the ITS algorithm is shown in Algorithm 4. It keeps a current solution S, an incumbent (best known solution) S^*, and a trial solution S'. The initial solution is generated by a constructive heuristic. At each step the algorithm perturbs the current solutions and applies a tabu search. The perturbation destroys part of the solution and reconstructs it, making sure to generate a new valid solution. The new perturbed solution is used to initialize an elite set E. A tabu search is applied to the solutions in this elite set, and when a new promising solution is found, the algorithm adds it to the set. When the elite set is empty, the best solution found by the tabu search S' is used to update the incumbent S^*, and an acceptance criterion decides if it will be the new current solution S.

Each element of the elite set E is a triple (S, T, η), where S is a solution, T a tabu list and

Algorithm 4: Iterated Tabu Search

input : ι, δ_I, τ, and C
output: The best solution found

Generate initial solution S with constructive heuristic
while *Time limit not expired* **do**
  T is an empty tabu list
  η is the complete neighbourhood of S
  push (S, T, η) to the elite set E
  $I = \iota$
  while $E \neq \emptyset$ **do**
    $S' = \min(S', TabuSearch(S', E, \iota, I, \delta_I, \tau, C))$
  end
  Update S^* with S'
  Acceptance criterion decides if $S = S'$
  Perform Perturbation on S
end
return S^*

η a subset of the neighbourhood of S. The elite set has a maximum size of e, and the element with the worst solution, which is also the oldest element in the set, is discarded when this limit is exceeded.

The parameters ι and δ_I control the variable I that is the maximum number of iterations of the tabu search without improving S'. The tabu tenure parameter is τ, and C is used to calibrate the stagnation detection which stops the tabu search.

The search stops if the lower bound is met, or if the time limit is reached.

4.2.1 A Constructive Heuristic for the PSS problem

The initial solution is generated by a constructive heuristic. It starts with an empty solution and repeatedly inserts an operation into the current partial solution. A partial solution represented by a linear order of the included operations on each job and each machine. It can be mapped to a partial solution represented by a disjunctive graph by selecting the orientation of the arcs between included operations according to the linear orders. Operations are inserted in order of non-increasing processing time. Each new operation o_{ij} is inserted into the linear order of machine i and job j at the positions which minimize the longest path from 0 to $*$ passing through o_{ij}. Only positions that respect the precedences $\prec$ between the operations and that do not lead to cycles are considered. Note that, if the new critical path passes through o_{ij}, this procedure minimizes the makespan at each insertion. Otherwise, it gives preference to insertion positions which minimize the internal delay caused by o_{ij}.

The constructive heuristic is shown in Algorithm 5. It generalizes the constructive heuristic INSA proposed by Nowicki and Smutnicki [62] to the JSS problem. The key difference is

Algorithm 5: Initial Solution Generator

input : PSS instance
output: A valid solution

$\forall j \in [1,n], \iota_j = ()$
$\forall i \in [1,m], \mu_i = ()$
$\Sigma = \{\iota_j \mid \forall j \in [1,n]\} \cup \{\mu_i \mid \forall i \in [1,m]\}$
for *each operation* $o \in \mathcal{O}$ *in non-decreasing order of* p_o **do**
  let j be the job of operation o
  let i be the machine of operation o
  $l_{min} = \infty$
  for $x = 0, \ldots, |\iota_j|, y = 0, \ldots, |\mu_i|$ **do**
    $\iota'_j = \iota_j$
    $\mu'_i = \mu_i$
    Insert o at position x in ι'_j
    Insert o at position y in μ'_i
    Map $(\Sigma \setminus \{\iota_j, \mu_i\}) \cup \{\iota'_j, \mu'_i\}$ into a partial solution S
    if *S is a valid partial solution* **then**
      let l be the longest path from 0 to $*$ in S containing o
      if $l < l_{min}$ **then**
        $\iota^*_j = \iota'_j$; $\mu^*_i = \mu'_i$
        $l_{min} = l$
      end
    end
  end
  $\Sigma = (\Sigma / \{\iota_j, \mu_i\}) \cup \{\iota^*_j, \mu^*_i\}$
end
Map Σ into complete solution S
return S

that we need to set the position of the operation in the job as well, so each operation can, in principle, have mn possible positions instead of n. This makes the construction of a schedule more expensive.

To accelerate this procedure, for each new operation o to be inserted we compute the heads and tails of the operations which are part of the partial solution. Given a candidate position for o we can compute the maximum distance from 0 to $*$ passing through o in constant time by

$$\max\{q(o_j) + p_{o_j}, q(o_m) + p_{o_m}\} + p_o + \max\{q(o^j) + p_{o^j}, q(o^m) + p_{o^m}\}.$$

This is similar to the acceleration technique proposed by Taillard [80] for the NEH heuristic [61] for the permutation flow shop problem. We use the same optimization in the perturbation, described further.

4.2.2 Neighbourhood for the PSS Problem

In this section we introduce a new neighborhood for the PSS problems. We first explain how we compute the critical path critical path, we then explain the neighborhood, and prove some basic characteristics of it. Finally, in Section 4.2.2.1 we show how to speed up its evaluation.

Let S be a solution for a PSS instance. An arc of S from operation a to operation b is a tight arc if, and only if, $q(b) = q(a) + p(a)$. This means that in the schedule associated with S operation b will begin processing as soon as a finishes. A critical path contains the operations in a path that follows the tight arcs from 0 to $*$. Each operation on this path will start processing as soon as the preceding operation in the path finishes.

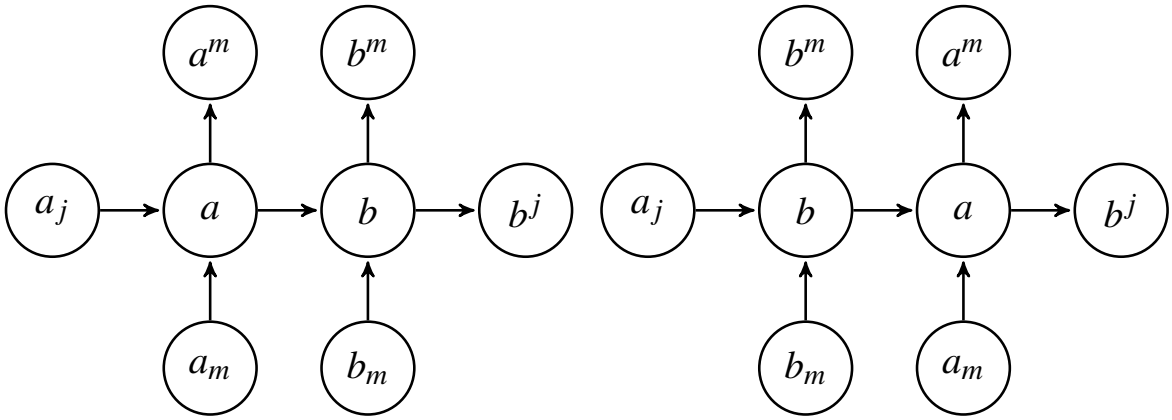
We find the critical path when we update the heads of each operation o according to $q(o) = \max\{q(o_j) + p(o_j), q(o_m) + p(o_m)\}$. This has linear time cost in the number of operations by updating $q(o)$ in topological order. The head $q(*)$ is equal to the makespan. Every time we compute the heads of a new operation o we store the predecessor that is connected to o by a tight arc, and if more than one is available we choose the job predecessor o_j. This operation is the *critical predecessor* of o. To compute the critical path we start from $*$ and follow the critical predecessor until we reach 0. We then just invert the list of operations to get the critical path. The reader can find more details on how to process the operations in topological order and find the critical path in Brucker [8].

The neighbourhood used in the tabu search step of ITS consists of the solutions generated by the transposition of pairs of operations that are adjacent on either a job or a machine in the disjunctive graph DG. Such a transposition is equivalent to changing the selected orientation of the corresponding disjunctive arc.

The pairs to be transposed in the neighbourhood are only pairs of operations in a critical path. If there is more than one critical path we choose the one with the longest job blocks. Furthermore, we only exchange the pairs of operations in the borderline of the blocks, except the first and last two operations of the critical path, if they share a block of size two or more. In the case of job blocks we perform only transpositions that respect the partial order, in order to keep the generated solutions valid.

This neighbourhood is an extension of the so-called N5 neighbourhood proposed by Nowicki and Smutnicki [62] for JSS. It has the following properties: it is disconnected, since it is identical to N5 for the JSS problem, and N5 is disconnected. Of all possible transpositions of operations, only transpositions in this neighbourhood can immediately decrease the makespan (Theorem 4.2.1 below), and the transpositions in the neighbourhood always generates valid solutions (Theorem 4.2.2 below).

In the following we use the notation shown in Figure 4.1. In the figure we assume the operations a and b being transposed are in the same job. If the pair of operations share the same machine the situation is symmetrical after exchanging the symbols j and m. We denote the head and tail of an operation o before the transposition by $q(o)$ and $r(o)$, and after the transposition



(a) Before. (b) After.

Figure 4.1: A transposition of a and b that share the same job.

by $q'(o)$ and $r'(o)$.

Theorem 4.2.1 *Of all possible transpositions of a pair of operations (a,b), only the ones in the neighbourhood can decrease the makespan.*

Proof. We can assume that (a,b) is on a critical path. Otherwise, after the transposition the same critical path would still exist and the maximum distance from 0 to $*$ would not decrease.

The length of the critical path before the transposition is $l = q(a) + p(a) + p(b) + r(b)$, and after the transposition, the longest the path through b and a has length $l' = q'(b) + p(b) + p(a) + r'(a)$.

Assume that (a,b) share the same job block, and that the transposition of a and b decreases the makespan, but is not in the neighbourhood. So the pair must be either in the interior of a block or be the first or the last pair of the critical path in a block of size three or more. Since (a,b) share a job block, $q(a) = q(a_j) + p(a_j)$, and $r(b) = r(b^j) + p(b^j)$. Note that if a is the first operation of the critical path $a_j = 0$, and $q(a_j) = q(a) = 0$, and if b is the last operation of the critical path $b^j = *$, and $r(b^j) = r(b) = 0$.

Moreover, by definition, $q'(b) = \max\{q(a_j) + p(a_j), q(b_m) + p(b_m)\} \geq q(a_j) + p(a_j) = q(a)$, and $r'(a) = \max\{r(b^j) + p(b^j), r(a^m) + p(a^m)\} \geq q(b^j) + p(b^j) = r(b)$, and therefore $l' \geq l$.

The same holds if (a,b) share a machine block, by a symmetric argument. ■

Figure 4.2 shows an example of a critical path to illustrate Theorem 4.2.1. We show the operations on the critical path labeled $x(y)$, where x is the operation and $y = p_x$ is its processing time. We group blocks of operations of size two or more by brackets, where top brackets group job blocks, and bottom brackets machine blocks. Each operation has an incoming arc on the critical path. If there exists another predecessor not on the critical path it has a second incoming arc. If this predecessor is a machine predecessor the arc enters from below, otherwise, for job predecessors, from above. The incoming arc from each such predecessor is labeled with the

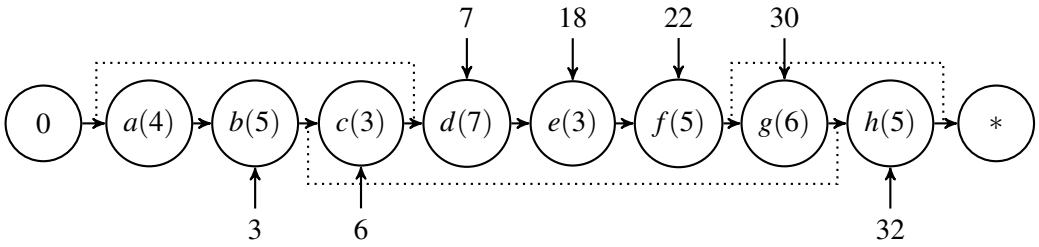


Figure 4.2: A critical path, and the longest paths from 0 to each operation outside this critical path.

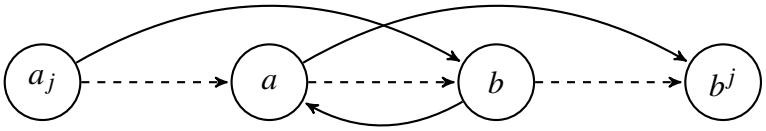


Figure 4.3: Arcs in DG of a job shared by a and b. A transposition of a and b removes the dashed arcs, and introduces the solid ones.

length of the longest path from 0 passing through it. The length of the longest path from 0 to an operation on the critical path is simply the sum of the processing times of its predecessors on the critical path. For example, operation e has $e_m = d$ on the critical path, and the longest path from 0 to e passing through d is 19, while the longest path from 0 to e passing through job predecessor e_j is 18, therefore $q(e) = 19$. The length of the critical path is 38, which is the sum of the processing times of all its operations.

For simplicity we assume this instance has no precedences between operations on this critical path. The neighbourhood contains the solutions generated by transposing the pairs (b,c), (c,d) and (f,g). If we transpose pair (e,f) the path $abcdfegh$ of length 38 still exists and we do not decrease the makespan. The same holds for every other adjacent pair not in the neighbourhood. After transposing (c,d) the path $abdcefgh$ does not exist, since b is not a predecessor of d and the makespan may decrease. In fact, the path from 0 to d through d_j then following $cefgh$ has length 36, and if no longer path was formed, it will be the new critical path and the makespan will decrease. When transposing pair (f,g), the makespan will not decrease, despite the fact that the solution is in the neighbourhood. This pair will be excluded from the neighbourhood by the lower bound presented in Section 4.2.2.1

Theorem 4.2.2 *For a valid solution S a transposition in the neighbourhood generates a valid solution S'.*

Proof. Since S is valid, the associated disjunctive graph contains no cycle, and the partial order of the jobs is respected. We exclude transpositions that do not respect the partial order, so the new solution will be valid if the disjunctive graph remains acyclic after the transposition.

Consider a transposition of (a,b). Then, by definition of the neighborhood, ab is on a critical path $P = QabR$. If the transposition creates a cycle, then it must contain some of the

arcs introduced by the transposition, since there was no cycle before. Figure 4.3 shows the change in the arcs of DG, where the dashed arcs are removed, and the solid ones are introduced.

Case 1: only arc (b,a) is part of the cycle. Thus, there exists a path P' from a to b in S'. Since arc (a,b) is not part of S', P' has length 2 or more, and since no operation has time 0, we have $l(P') > p(a)+p(b)$. But P' is also a path from a to b in S, and thus $P'' = QP'R$ is a path of length $l(P'') = l(Q)+l(P')+l(R) > l(Q)+p(a)+p(b)+l(R) = l(P)$, a contradiction with the fact path P is critical.

Case 2: The cycle contains at least one of the arcs (a_j,b), (a,b^j), and possibly arc (b,a). In this case there must exist a path in S from operation b or b^j to operation a or a_j. In other words, there is a path from an operation to another operation that precedes it in S (in Figure 4.3 operations to the left precede operations to the right in S). In this case S would contain a cycle. ■

To illustrate both cases of the proof, consider transposing operations (c,d) in the example of Figure 4.2. There can be no indirect path from a to b, since that would contradict the fact that $abcdefgh$ is a critical path, and there can be no path from e to c, since that would contradict the fact that the solution before the transposition cannot contain a cycle.

4.2.2.1 A Lower Bound for the Neighbours

The proposed neighbourhood leads to a manageable number of neighbours for each solution. It tends to have less than twice the number of neighbours that N5 has for an equivalent JSS instance. However, it is possible to decrease the number of solutions to be investigated. To do so we use a lower bound to evaluate which neighbors are more promising, and evaluate them first. We stop the neighbourhood evaluation when it is impossible to find better neighbours.

For a transposition of a pair of adjacent operations of the same job, the following theorem gives a lower bound for the new solution.

Theorem 4.2.3 *Let a and b be adjacent operations of the same job on the critical path, and define $L_1 = \max\{q(a_j),q(b_m)\}+p_b+r(b^m)$, $L_2 = q(a_m)+p_a+\max\{r(b^j),r(a^m)\}$, and $L_3 = \max\{q(a_j),q(b_m)\}+p_a+p_b+\max\{r(b^j),r(a^m)\}$. Then, $L = \max\{L_1,L_2,L_3\}$ is a lower bound for the makespan of the new solution after the transposition of a and b.*

Proof. The longest path passing through b and not through a has length L_1, the longest one passing through a and not through b has length L_2, and finally the longest one passing through a and b has length L_3, therefore L is a lower bound for the new state. ■

A similar theorem can be derived from the transposition of an adjacent pair of operations of the same machine, by exchanging the indices j and m. Both lower bounds can be computed in constant time, when the heads and tails are known.

As an example, consider the neighbourhood associated with the critical path shown in Figure 4.2. For the transposition of the pair (f,g) we $L_1 = \max\{22,30\}+6+5 = 41$, which is

larger than the current makespan of 38.

4.2.3 Tabu Search Procedure

Tabu search is a meta-heuristic proposed by Glover [26] which combines a best-improvement local search with short-term and possibly long-term memory, to escape local minima. Attributes of previously visited solutions are stored in a so-called *tabu list* and are then declared *tabu* for a number of iterations, the *tabu tenure*. Elements of the neighbourhood which are not tabu are preferred. To avoid missing good solutions often the tabu search uses *aspiration criteria* which are rules that allow the tabu neighbours to be selected even if non-tabu solutions are available. We use the most common aspiration criterion which is accepting tabu neighbours if they improve the incumbent best solution S' found so far.

Algorithm 6: Tabu Search

input : $S', E, \iota, I, \delta_I, \tau, C$
output: Best solution found.

Let (S,T,η) be the best elite element.
$i = 0$
$S = \text{NSP}(T,\eta)$
Update (S,T,η) to $(S,T,\eta \setminus \{S\})$, and remove it from the elite set if $|\eta| = 1$
while *not* $Cycle(C,R)$ *and* $i < I$ **do**
    $i = i+1$
    Update tabu list T
    if $C_{\max}(S) < C_{\max}(S')$ **then**
        $I = \iota$
        add (S,T,η) to the elite set E
        return S
    end
    $S = \text{NSP}(T, N(S))$
end
$I = I - I/\delta_I$
return S

Algorithm 6 shows the pseudo-code for the tabu search. It starts by getting the best element (S,T,η) from the elite set, which is also the newest element in the set. Then, the *neighbourhood search procedure* NSP, explained below, selects a neighbor S from the set of neighbors η. The tabu search removes S from η of the selected elite element, and if η is empty removes the element completely from the set. The algorithm updates the tabu list T according to the move selected by NSP. Each element of the tabu list is a pair of operations swapped that generates the selected neighbour. Each movement remains tabu for the number of iterations given by the tabu tenure τ.

NSP evaluates the neighbors in non-decreasing order of their lower bounds L. The procedure chooses the best neighbor if it improves S', or the best non-tabu neighbor, otherwise. If there

is no such neighbor, none is selected, and the next iteration begins. NSP stops when the lower bound of the current neighbour is larger than the makespan of an already evaluated non-tabu move.

If the current solution S is better than the best solution since the last perturbation S', it adds (S, T, η) to the elite set E and returns the solution S to the ITS.

The tabu search has a budget of I iterations that can be performed without improving S'. It stops if it is able to improve S', in which case the elite set is updated, or if stagnation is detected by either reaching the limit of I iterations without improving S', or if a cycle in the makespan values of S is detected. Each time the tabu search finishes without improving S', the budget I is decreased by I/δ_I, and each time it is able to improve S', the budget I is set to ι again. To detect cycles the algorithm checks for the values for the makespan of each newly generated solution, and if a sequence of at most C makespans repeats itself it assumes the search is cycling and stops. This algorithm is adapted from Nowicki and Smutnicki [62].

4.2.4 Perturbation

The tabu search uses the tabu list to avoid getting stuck in local minima, yet it is still possible that the search will get stuck in a region of the search space. In this case we may loose interesting solutions, so we use a perturbation to move the search, more aggressively, to another part of the search space.

The perturbation in the ITS algorithm uses the mechanism to build the initial solution. Given a complete solution the procedure removes π operations and then reinserts them, where π is a parameter. This operation is sometimes called "destroy and repair" or "deconstruction and reconstruction" in the literature. The operations are processed in a random order, and the insertion position of each operation is determined as in the constructive heuristic, so the new solution is guaranteed to be valid.

4.2.4.1 Bubble Perturbation

The bubble perturbation works by changing the relative ordering of *sb* jobs or machines, where *sb* is a parameter to be calibrated. The *bubble perturbation* procedure is similar to the perturbation in the Bubble Search proposed by Lesh and Mitzenmacher [46].

In bubble search sampling we obtain a new permutation π' from a base permutation π. Each possible new permutation can be selected with chance proportional to $(1-\alpha)^{-d}$, where α is a parameter and d is the Kendall-Tau distance to the original ordering. The procedure to select a new permutation according to the bubble search sampling is explained in Section 4.1

To perform the bubble perturbation we select *sb* random jobs or machines. For each of job or machine we extract the linear order π of its operations induced by DG. Let A be the set of operations of the job or machine we selected to perturb. We then compute π' according to a

modified bubble sampling. For each pair of operations u and v in A we make an new selection of disjunctive arc. We select (u,v) if u is earlier than v in π, and we select (v,u) if not.

Given a complete solution for the PSS problem we cannot change the ordering of a machine or job arbitrarily, because some permutations can lead to a cycle in DG, or, can break the expected partial order in case of a job and lead to invalid states. To avoid this problem we compute a partial order Γ such that, if enforced, guarantees that the new modified DG will generate a valid state. We modify the bubble search sampling, so that it generates only permutations that are linear extensions of Γ.

To compute Γ for a machine we start with an empty partial order, and for a job we start with the partial order of the job itself. In both cases we then remove the arcs of DG connecting any two operations in A. For each pair of operations $a,b \in A$, such that $a \neq b$, and such that we can reach b in DG starting from a, we add the precedence $b \prec a$ to Γ.

This procedure guarantees that the new selection does not create cycles, if DG was acyclic before the perturbation, and if the new order for the operations in A is a linear extension of Γ. If a new cycle is introduced it has to contain one of the new arcs introduced to re-order the operations in A, yet an arc (a,b) can only be selected if it is impossible for b to reach a. To find quickly which operations reach one another we iterate once through the topological order that we obtained when we computed the makespan, as explained in Section 4.2.2.

To change the permutation of operations in DG of a job or machine and generate a new perturbed state we use the modified Bubble sampling, the *Partial Bubble sampling*, shown in Algorithm 7. It follows the same structure of the Bubble sample but only operations whose predecessors in Γ were already included in π' are tested for selection. The Bubble sampling is an special case of the Partial Bubble sampling where Γ is empty.

Algorithm 7: Partial Bubble Sampling

input : A permutation π with length η, an insertion probability bp, and a partial order Γ.
output: A linear order that respects Γ.

let $\pi' = ()$ be the empty sequence.
for $x = 1,\ldots,\eta$ **do**
  $y := 1$
  while *no element selected* **do**
    if *All predecessors of* π_y *in* Γ *are in* π **then**
      with probability bp: select element π_y
      otherwise: $y := y \mod \eta + 1$
    end
  end
  remove π_y from π and append it to π'.
end
return π'
return S;

For a given a partial order Γ for a set H of size $|H| = \eta$ the *order strength* $os \in [0,1]$ is given

by Equation 4.1a. It is a measure of how many different topological orders exist for Γ. If there are more possible topological orderings os is closer to 1, and if there are fewer it is closer to 0. A total order has only one possible linear extension and has $os = 1$, while an empty partial order has $\eta!$ possible linear extensions and has $os = 0$.

$$os = \sum_{a,b \in H} \psi(a,b) \Big/ \binom{\eta}{2} \tag{4.1a}$$

$$\psi(a,b) = \begin{cases} 1, & \text{if } a \prec b \in \Gamma \\ 0, & \text{otherwise} \end{cases} \tag{4.1b}$$

We use the order strength of the job when selecting the *sb* random jobs or machines to skew the selection probabilities. We work with particular cases of PSS whose jobs partial orders range from both ends of the spectrum of the order strength, so when performing the bubble perturbation we select each job and machine with probability proportional to its *flexibility*. The flexibility of a job is $1 - os$, and the flexibility of all machines is 1.

This perturbation is an extension of the bubble search procedure we implemented for PFSS, where we include the partial order precedence into the perturbation. This perturbation is not used directly in ITS, but we use it in the component analysis.

4.2.5 Acceptance Criterion

The ITS algorithm repeatedly applies the tabu search, and each time the tabu search finishes and produces a new solution the ITS algorithm decides to either keep the new solution or to discard it. The simplest approach is to keep only solutions that improve the current best solution, but to allow better diversification we use the *Metropolis* acceptance criterion. It always accept a new better solution, and worse solutions have a chance to be accepted, such that the probability gets smaller for worse solutions. The probability of acceptance is given by

$$e^{-\Delta/T} \tag{4.2}$$

where T is a parameter, the so-called temperature. We define the temperature of an instance by

$$T = \alpha \overline{p}/10 \tag{4.3}$$

for a parameter α and an average processing time of

$$\overline{p} = \sum_{j \in n} \sum_{i \in m} \frac{p_{ji}}{nm}. \tag{4.4}$$

4.3 Mixed Integer Program for Parallel Machines Job Shop

In the DG representation the conjunctive arcs model the given precedence relations between operations, disjunctive arcs model alternatives in the processing order. We assume that the operations of job j_j are $o_{(j-1)i+1},\dots,o_{jm}$ given in order of their execution. Then for the JSS we have

$$\begin{aligned} C &= \{(O_o, O_{o+1}\} \mid o \in ((j-1)m+1, jm), j \in J\} \\ &\quad \cup \{(0, O_o) \mid o = (j-1)m+1, j \in J\} \cup \{(O_o, *) \mid o = jm, j \in J\}, \\ D &= \{(O_o, O_p\} \mid \mathfrak{m}(o) = \mathfrak{m}(p)\}. \end{aligned}$$

A solution to the JSS is described by a selection of disjunctive arcs $S \subseteq D$, such that S contains either (O_o, O_p) or (O_p, O_o) for each pair of operations with $\mathfrak{m}(O_o) = \mathfrak{m}(O_p)$, i.e. defines a total order on the machines, and such that the graph $G' = (V, C \cup S)$ is acyclic.

The disjunctive graph model leads naturally to a mathematical formulation of the JSS, where we introduce a binary variable $x_{op} \in \{0,1\}$ for each pair of operations with $\mathfrak{m}(O_o) = \mathfrak{m}(O_p)$, such that $x_{op} = 1$ when operation O_o precedes operation O_p on their common machine. Introducing further starting times $y_o \in \mathbb{R}$ for each operation $o \in O$, and an auxiliary variable $C_{\max} \in \mathbb{R}$ representing the makespan, i.e. the completion time of the artificial operation $*$, we obtain the integer linear program

$$\begin{aligned} \textbf{minimize} \quad & C_{\max} && && (4.5) \\ \textbf{subject to} \quad & C_{\max} \geq y_o + p_o, && \forall o \in O, && (4.6) \\ & y_p \geq y_o + p_o && \forall (O_o, O_p) \in C, && (4.7) \\ & y_o \geq y_p + p_p - M x_{op} && \forall (O_o, O_p) \in D, && (4.8) \\ & y_p \geq y_o + p_o - M(1 - x_{op}) && \forall (O_o, O_p) \in D, && (4.9) \\ & x_{op} \in \{0,1\}, && \forall (O_o, O_p) \in D, && (4.10) \\ & y_o, C_{\max} \in \mathbb{R} \geq 0, && \forall o \in O. && (4.11) \end{aligned}$$

Here constraint (4.6) defines the maximum completion time, constraint (4.7) forces operations with fixed precedences to start after their predecessor terminated, constraints (4.8) and (4.9) defines the order of the disjunctive operations depending on the chosen order by the x variables. M is large constant, and can be set to $M = \sum_{o \in O} p_o$, for example. Finally, (4.10) and (4.11) define the domains of the variables.

When we introduce parallel machines, $\mathfrak{m}(o)$ now only defines the stage on which the operation has to be executed. In this case we have to decide additionally to which machine an operation will be assigned. For the case of k parallel machines for each of the m stages, we can introduce additional decision variables $s_{oi} \in \{0,1\}$ such that $s_{oi} = 1$ when operation O_o is

executed on the ith parallel machine on its stage. This leads to the model

$$\textbf{minimize} \quad C_{\max} \tag{4.12}$$

$$\textbf{subject to} \quad \sum_{i\in[k]} s_{oi} = 1 \qquad \forall o \in O, \tag{4.13}$$

$$s_{oi} + s_{pi} \leq x_{op} + x_{po} + 1, \qquad \forall (O_o, O_p) \in D, \tag{4.14}$$

$$y_p \geq y_o + p_o - M(3 - s_{oi} - s_{pi} - x_{op}) \qquad \forall (O_o, O_p) \in D, \tag{4.15}$$

$$s_{oi} \in \{0,1\}, \qquad \forall o \in O, i \in [k], \tag{4.16}$$

$$\text{constraints (4.6),(4.7),(4.9),(4.10).} \tag{4.17}$$

In this model, the new constraints (4.13) make sure that each operation is assigned to a single machine on its stage, and constraint (4.14) forces precedences among operations on the same stage, when they have been assigned to the same machine. Note that, different from the model for the JSS, it is possible that $x_{op} = x_{po} = 0$ for some pairs of operations when they have been assigned to different machines. Constraint (4.15) forces non-overlapping execution of operations assigned to the same machine. Constraint (4.16) defines the domains of the new variables s_{oi}.

4.4 Tabu Search for Job Shop with Parallel Machines

In this section we describe a tabu search designed for JSS-PM. It combines a neighbourhood based on the critical path with an heuristic scheduling of the parallel machines.

4.4.1 Heuristic Scheduling for Parallel Machines

We present a novel way to produce a schedule for JSS-PM which does not include the replicated machines explicitly in the DG representation. We use the DG representation of JSS ignoring the replicated machines, but adapt the procedure that computes the schedule and makespan. Given a complete selection of the DG we derive the associated schedule by starting the execution of each operation as soon as all preceding operations, according to the DG, have been scheduled.

Our algorithm processes all operations in topological order, as defined by the DG. This can be done in time $O(mn)$. Each operation o will be assigned to one of the machines $c_{\mathfrak{m}(o),1}, \ldots, c_{\mathfrak{m}(o),k}$ and be scheduled on this machine. The assignment and starting time depends on the completion time of its job predecessor and on the occupation of the k replicated machines of its corresponding stage.

During the scheduling procedure when we process operation o all the preceding operations of the corresponding job $\mathfrak{j}(o)$ have been scheduled on stages different from $\mathfrak{m}(o)$, and all the preceding operations, according to the complete selection, at the same stage $\mathfrak{m}(o)$ have being

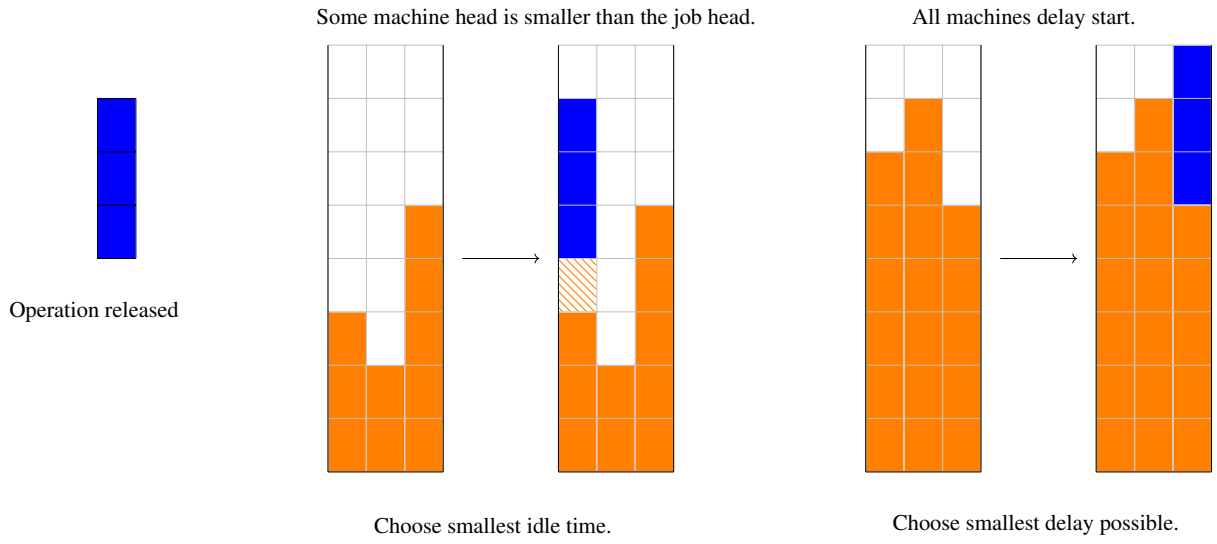


Figure 4.4: Example of the heuristic scheduling of an operation for 3 machines.

scheduled on one of the k machines of the stage.

We use the concepts of job and machine heads to perform the heuristic scheduling. The job head can be used unmodified for the non-parallel case, but the machine head has to be adapted to incorporate the parallel machines. We call the completion time of the last operation scheduled at the ith machine of stage $\mathfrak{m}(o)$ the ith machine head $q^i_{\mathfrak{m}(o)}(o)$ of operation o.

Let o be the operation to be scheduled, such that $\mathfrak{m}(o) = m$. We check if there exists a machine $m_{m,i}$ such that $q_j(o) \geq q^i_m(o)$, these are called non-delay machines. In case there are non-delay machines we can schedule operation o at time $q_j(o)$ without delay. In this way operation o will start as soon as its job predecessor finishes. Among the non-delay machines we select the one with the largest machine head. The rationale behind this decision is that a machine that is available earlier is more valuable, and therefore makes sense to select the machine that is released later, this operation will not be scheduled any earlier with any other selection. Ideally the selected non-delay machine will have the same head as the job head and no idle time will be introduced in the machine or the job schedule.

If no machine satisfies $q_j(o) \geq q^i_m(o)$ the set of non-delay machines is empty and we choose the machines i with the lowest machines head and schedule the operation at time $q^i_m(o)$. In this case the schedule will have idle time between operation o and o_j, and will start as soon as any of the available machine of its stage is free.

In Figure 4.4 we see an example of such a schedule. The operation's job head is 4. In the case on the left, the first machine's head is 3, the second is 2, and the third is 5. The set of non-delay machines contains the first two machines. Of the two we select the first machine, because it introduces the least amount of idle time. In the figure on the right, the machines have heads 6, 7 and 5. Therefore the set of non-delay machines is empty, and we select the third machine, since it delays the start of the operation the least possible.

This procedure generates only non-delay schedules, but does not guarantee to generate ac-

tive schedules. It is possible to generate any non-delay schedule with this procedure with the proper ordering of the DG and at least one of the optimal schedules is a non-delay schedule. Therefore it is always possible to represent at least one optimal solution for any instance of JSS-PM in this way.

4.4.2 Tabu Search for the Parallel Machines Job-Shop Scheduling Problem

We propose a tabu search for JSS-PM. It is based on the tabu search for JSS by Nowicki and Smutnicki [62], and is similar to the tabu search proposed for PSS.

Algorithm 8: Tabu Search

input : Instance and parameters $I_{\max}$, Δ, L, T
output: Best solution found.

```
1  S* = Constructiveheuristic()
2  I = I_max
3  add S* to elite set E
4  while E ≠ ∅ do
5  |  get the next solution S from E
6  |  i = 0
7  |  repeat
8  |  |  i = i + 1
9  |  |  S = NSP(S)
10 |  |  if C_max(S) < C_max(S*) then
11 |  |  |  I = I_max; i = 0
12 |  |  |  update the incumbent S*
13 |  |  |  insert S* into the elite list E
14 |  |  end
15 |  |  if Cycle or i > I then
16 |  |  |  I = I − Δ
17 |  |  |  break
18 |  |  end
19 |  until;
20 end
21 return S*
```

Algorithm 8 shows the pseudo-code of the proposed tabu search. The algorithm is deterministic and has four parameters (I, Δ, L, T) which will be explained below.

4.4.2.1 Constructive Heuristic

To generate the initial solution we use a constructive heuristic. We start with an empty solution, and repeatedly schedule the first unscheduled operation of the job with the highest remaining workload, where the remaining workload of a job is the total cost of its unscheduled operations.

To generate the schedule we keep track of the occupation of each parallel machine. We schedule each new operation as early as possible after its job predecessor finishes. To this end, we determine for each parallel machine the earliest starting time of the operation after the completion of its job predecessor, such that it can run to completion without interruption. Note that scheduling the operation can split an interval of idle time on a machine into two intervals. To avoid fragmenting idle times, we break ties among machines with the same earliest starting time by giving preference to those which do not split idle time intervals. Any remaining ties are broken by selecting the machine that generates the smallest idle time before the operation to be scheduled, and finally, if there is still a tie, by selecting the machine with smaller index.

4.4.2.2 Neighbourhood Search Procedure

The *neighbourhood search procedure* NSP evaluates and selects one of the neighbours at each iteration. The tabu list contains pair of operations, which cannot be shifted. Each time a move is performed, the inverse operation is added to the tabu list with a tabu tenure of T. The NSP tests all neighbours of the current solution and chooses the next solution with the following priority:

1. The best non-tabu move that improves the best solution found so far.
2. The best tabu move that improves the best solution found so far.
3. The best non-tabu move.
4. The oldest tabu move.

If we select the oldest tabu move, the algorithm artificially forwards the number of iterations performed in the tabu list until a move is available.

For intensification the tabu search uses an elite list as a long-term memory. The elite list contains the L best solutions found during the search. For each solution in the elite list, we maintain a list of neighbors that have been examined already. Each time a new best solution is found it is inserted into the elite list E along with the associated tabu list and the neighbours that were not chosen by the NSP. If the list already contains L solutions, the oldest element of the list, namely the solution, candidate and tabu list, is discarded.

The tabu search starts at the best neighbor of the initial solution. It runs with a budget of a most I iterations, which is renewed whenever the incumbent improves. Otherwise after I iterations or when a cycle has been detected, the current solution is abandoned, and the tabu search continues from the next solution from the elite list. If the elite list is empty, the search stops. The budget I is initially $I_{\max}$ and is reduced by Δ whenever a solution is abandoned. If the incumbent improves, the budget is reset to $I_{\max}$.

In order to avoid cycling, the search includes a simple cycle detector. If the makespan of the last 100 solutions is repeated then the search also abandons the current solution and continues from the next solution in the elite list.

4.4.3 Neighbourhood

Given a non-delay schedule there exists at least one permutation of a subset of the operations $\{o_1, o_2, ..., o_x\}$ such that o_1 starts at time 0, o_x finishes at the makespan of the schedule, and $f_{o_i} = s_{o_{i+1}}$ for all $i \in [1, x)$. Such a sequence of operations is the critical path. Notice that this is an alternative way to define the critical path to the one in Section 3.4. It is an equivalent definition for the case with no parallel machines, but not for the case with parallel machines due to the heuristic scheduling presented in Section 4.4.1. Any two adjacent operations (o_i, o_{i+1}) in a critical path are either executed on the same machine or belong to the same job. Moreover, in the usual DG representation for the JSS they are also directly connected by a conjunctive or disjunctive arc. With our scheduling scheme this is necessarily true only if they belong to the same job, otherwise o_i may be any of the preceding operations of o_{i+1} in the DG that share the same stage.

We adapt the technique to compute the critical path for the parallel machine case. Let o be an operation to be scheduled. In JSS-PM o may be delayed not because of an operation that immediately precedes it in DG, but by the last operation scheduled on the machine to which it was assigned. Operations adjacent on the critical path will not necessarily be adjacent on DG. When we schedule o, if there are any non-delay machines, then the job head is larger then any of the machine heads, and s_o is defined by the job heads $q_j(o)$, hence o_j will be also the predecessor on the critical path. On the other hand if there are no non-delay machines and we select machine i to schedule o, then this machine head $q^i_{\mathfrak{m}}(o)$ will define s_o and therefore the last operation scheduled on this machine will be the predecessor in the critical path. This operation is not necessarily o_m.

Let operation p be an operation on the critical path, and let b_i be the last operation scheduled on the ith parallel machine of before p was scheduled. Let $B(p)$ be the set of operations that contains all the operations b_i that delay the start of operation p on the stage. Therefore it contains the last operations scheduled on each machine of the stage when p was scheduled that finish after p was released by the predecessor in its job.

The neighborhood of a complete selection consists of all complete selections generated by a *backward shift* of an operation p on the critical path to the position immediately preceding some element of $B(p)$.

An example of the modification in DG caused by a backward shift on a machine is shown in Figure 4.5. The orange operation f is shifted right before the other orange operation c. We see the machine in the DG before the shift, the modification of the shift, and the resulting machine ordering after the shift. The black arcs remain the same, the dashed arcs are removed, and the red arcs are inserted. This may generate invalid states, because the resulting DG may contain a cycle. In cases the operations are adjacent in the DG, then the shift is equivalent to a transposition.

Figure 4.6 shows an example of a complete selection and the schedule induced by it. On

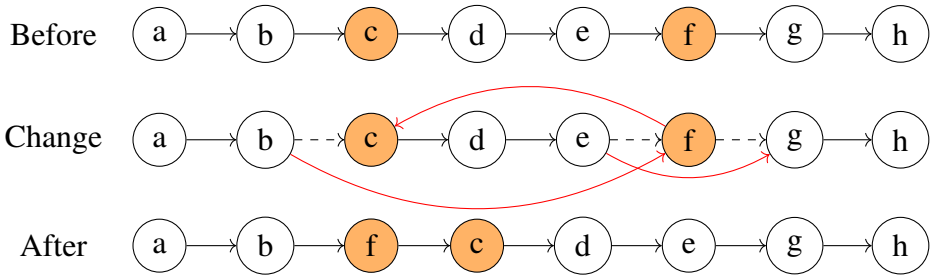


Figure 4.5: Example of a backward shift.

Table 4.1: Processing time of the operations.

	Job			
Machine	1	2	3	4
1	2	2	1	2
2	1	2	1	1

the left we see the DG representation of a complete selection for an instance with four jobs and two stages, each with two parallel machines. The processing times of the operations is shown in Table 4.1. On the right of Figure 4.6 we see the Gantt chart of the schedule induced by the heuristic scheduling of the operations on the parallel machines. The makespan of this solution is 6. The critical path is indicated on the Gantt chart by the dotted line. Figures 4.7 and 4.8 show the relation between a swap on the critical path and the associated shift on the machine. Figure 4.7 shows the critical path explicitly, and a possible swap that generates a neighbour solution, while Figure 4.8 shows the resulting shift on m_1. Figure 4.9 shows the DG and the schedule of the neighbour. The makespan was reduced to 5, which is the optimal makespan.

4.4.3.1 Extended Neighborhood

The neighbourhood may generate invalid solutions by creating cycles. We initially ignore the invalid solutions, yet it is possible that the neighbourhood contains no valid solutions. In such cases we use an extended neighbourhood.

Consider a solution in the neighbourhood of a valid solution, generated by the shift of operation f right before c as shown in Figure 4.5. If the new solution has a cycle it must contain one of the newly introduced arcs, marked in red in the figure. In particular the new cycle must contain the arc (f,c), because if (b,f) forms a cycle, then there is a path from f to b and there would be a cycle before the shift. The same argument can be made for the arc (e,g). Therefore to find the arcs that form the new cycle we find the paths from c to f.

The extended neighbourhood consists of all transpositions of adjacent operations in the cycles formed on the regular neighbourhood.

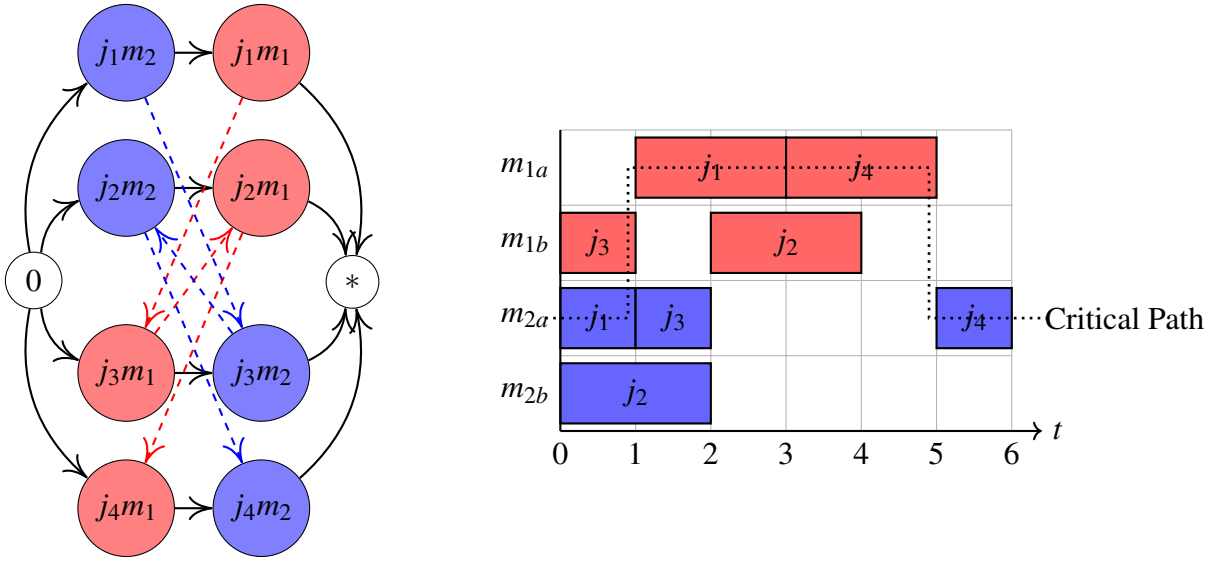


Figure 4.6: Disjunctive graph and Gantt chart before the swap.

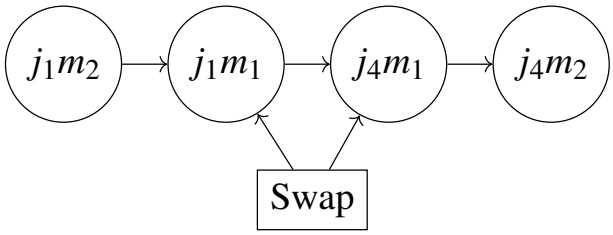


Figure 4.7: Critical path.

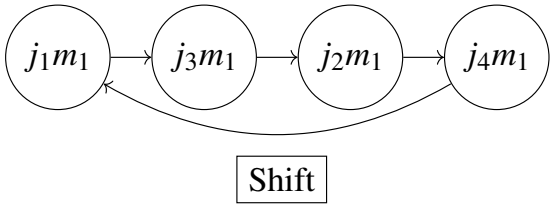


Figure 4.8: Shift on the machine.

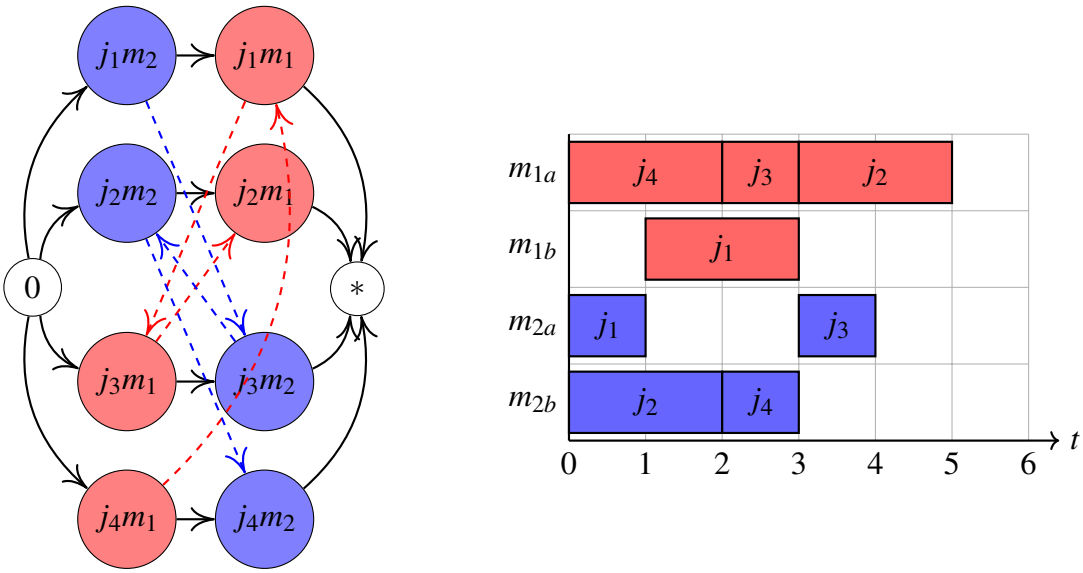


Figure 4.9: Disjunctive graph and Gantt chart after the swap.

4.4.3.2 Learning good moves

Here we introduce a learning component into the tabu search, which is able to learn and predict promising moves. It works as follows. Whenever restarting from a new elite solution the learning component is reset. During the tabu search it proceeds in two phases, a learning phase and a trimming phase. In the learning phase, the neighborhood is fully explored to acquire knowledge about good moves. In each iteration the best non-tabu move in the neighborhood is identified, and applied. The learning component maintains, for each operation $o \in O$, the frequency $f_1(o)$ of being part of the best move as the left element of the backward shift, and the frequency $f_2(o)$ of being part of the best move as the right element of the backward shift. In the trimming phase the learning component predicts the best $\beta\%$ of the moves (rounded up), for a parameter β. These moves are then evaluated, and the best non-tabu move among them is applied. The quality of a move (o, p) is predicted as the sum of the corresponding frequencies $f_1(o) + f_2(p)$. The evaluated moves are also used to update the success frequency of the operations. Note that this is robust in the sense that it yields with high probability the same result as evaluating the complete neighborhood, since the best move is very likely part of the selected moves. The search also maintains the agreement between the predicted order of the moves in the neighborhood and the order defined by the objective function. The agreement is measured by Kendall's W. The search switches from the learning to the trimming phase W exceeds some fixed threshold W_0.

We do not allow the trimming phase start before the learning component has observed at least 30 iterations, to avoid starting the trimming too early because the initial predictions were lucky. When the neighbourhood search procedure selects a tabu move the statistics for the operations are not updated in the learning component. In case we are in the trimming phase, and the neighbourhood search procedure cannot find a valid solution, the trimming is temporarily ignored, and the whole neighbourhood is evaluated. If we still cannot cannot find a valid solution, we use the previously explained extended neighbourhood.

5 INSTANCES

Next we describe the set of instances used to evaluate the proposed algorithms.

5.1 Permutation Flow Shop Scheduling

For our test use the well known and widely used set of instances proposed by Taillard [82]. The test set consists of 12 groups of instances with the number of jobs varying between 20 and 500, and the number of machines between 5 and 20. Each group contains 10 instances, with processing times drawn uniformly at random in the interval $[1,99]$. The current best known values for all 120 instances can be obtained at Taillard [84].

5.2 Partial Shop Scheduling and Special Cases

We ran computational tests not only with instances proposed specifically for the PSS problem, but also for the shop problems GSS, MSS, OSS which are particular cases of PSS. We have used one set of instances for the PSS, three sets for the GSS, one set for the MSS, and three sets for the OSS. An overview of all instance groups of the same size is given in Table 5.1, which shows for each group the names of the instances, the number of jobs n and machines m, the number of instances in the group ("#"), and the order strength ("OS"), i.e. the fraction of actual precedence relations of all possible precedences $n\binom{m}{2}$. For MSS the number of linearly ordered jobs and unordered jobs are given ("$J^j + J^o$"). We could not obtain the instances marked with $*$ from the authors, so we generated them, as described below. The instances marked with † are generated by us. In all such cases there is indeterminism in the generation procedure, so we repeated the construction 10 times for the PSS and some of the GSS instances (ata71–80) and 20 times for the MSS instances, and in the experiments report averages.

The PSS set is based on the 80 instances proposed by [60]. We could not obtain these instances, and therefore we generated them. The instances are based on the JSS instances by Taillard [81]. Each has the same number of jobs, machines and processing times as the associated JSS instance. The partial order is selected at random. Each job has a random number of precedences in the interval $[1,m]$ and each precedence is between two random operations of the job. They are denoted PSSpta01–80.

For GSS we use the instances proposed by Blum and Sampels [7] and Nasiri and Kianfar [59]. The first set contains the 40 instances GSSft10_01–10, GSSla38_01–15 and GSSabz7_01–15. n, m and the processing times are given by a JSS instance. The GSSft10_01–10 instances are associated with the JSS instance from [57], the GSSla38_01–15 with la38 [43], and the GSSabz7_01–15 instances with abz7 from [1]. The second index in the instance names is the stage size S for all jobs of that instance. The first s operations of each job belong to the first stage, the next s to the second, and so on, until the last $m \mod s$ operations which belong to the

Table 5.1: Instance sets used in the computational experiments.

Name	n	m	#	OS	Name	n	m	#	OS
PSSpta01–10*	15	15	100	0.14	MSS10*	15+15	15	10	0.50
PSSpta11–20*	20	15	100	0.14	MSS11*	15+20	10	10	0.43
PSSpta21–30*	20	20	100	0.11	MSS12*	15+20	15	10	0.43
PSSpta31–40*	30	15	100	0.13	MSS13*	20+5	10	10	0.80
PSSpta41–50*	30	20	100	0.10	MSS14*	20+10	10	10	0.67
PSSpta51–60*	50	15	100	0.13	MSS15*	20+15	10	10	0.57
PSSpta61–70*	50	20	100	0.10	MSS16*	20+20	10	10	0.50
PSSpta71–80*	100	20	100	0.10	OSStai_4 $\times$ 4	4	4	10	0.00
GSSabz7_01–10	10	10	10	0.55	OSStai_5 $\times$ 5	5	5	10	0.00
GSSft10_01–10	15	15	15	0.55	OSStai_7 $\times$ 7	7	7	10	0.00
GSSla38_01–10	20	15	15	0.55	OSStai_10 $\times$ 10	10	10	10	0.00
GSSadmu01–05	20	15	5	0.86	OSStai_15 $\times$ 15	15	15	10	0.00
GSSadmu05–10	20	20	5	0.82	OSStai_20 $\times$ 20	20	20	10	0.00
GSSadmu10–15	30	15	5	0.85	OSSj5	5	5	9	0.00
GSSadmu15–20	30	20	5	0.80	OSSj6	6	6	9	0.00
GSSadmu20–25	40	15	5	0.85	OSSj7	7	7	9	0.00
GSSadmu25–30	40	20	5	0.89	OSSj8	8	8	8	0.00
GSSadmu30–35	50	15	5	0.82	OSSgp03	3	3	10	0.00
GSSadmu35–40	50	20	5	0.80	OSSgp04	4	4	10	0.00
GSSata01–10	15	15	10	0.87	OSSgp05	5	5	10	0.00
GSSata11–20	20	15	10	0.86	OSSgp06	6	6	10	0.00
GSSata21–30	20	20	10	0.82	OSSgp07	7	7	10	0.00
GSSata31–40	30	15	10	0.85	OSSgp08	8	8	10	0.00
GSSata41–50	30	20	10	0.80	OSSgp09	9	9	10	0.00
GSSata51–60	50	15	10	0.82	OSSgp10	10	10	10	0.00
GSSata61–70	50	20	10	0.80	GSS(PSS)01–10†	15	15	100	0.57
GSSata71–80*	100	20	100	0.85	GSS(PSS)11–20†	20	15	100	0.56
MSS1*	10+5	10	10	0.67	GSS(PSS)21–30†	20	20	100	0.53
MSS2*	10+10	10	10	0.50	GSS(PSS)31–40†	30	15	100	0.55
MSS3*	10+15	10	10	0.40	GSS(PSS)41–50†	30	20	100	0.55
MSS4*	10+20	10	10	0.33	MSS(PSS)01–10†	15	15	100	1.00
MSS5*	15+5	10	10	0.75	MSS(PSS)11–20†	20	15	100	1.00
MSS6*	15+5	15	10	0.75	MSS(PSS)21–30†	20	20	100	1.00
MSS7*	15+10	10	10	0.60	MSS(PSS)31–40†	30	15	100	1.00
MSS8*	15+10	15	10	0.60	MSS(PSS)41–50†	30	20	100	1.00
MSS9*	15+15	10	10	0.50					

last stage. Therefore in this instance set all jobs of an instance are partitioned in the same way. The procedure to generate this set of GSS instances is deterministic, so our set is identical to the one proposed in [7]. The second set, proposed by Nasiri and Kianfar [59], consists of 120 instances GSSata01–80, and GSSadmu01–40. For this set the authors select a random number of partitions, and each partition has a random size. All these instances were available, except GSSata71–80, which we generated to complete the testbed.

The MSS test instances contains 16 instances, which we denote by MSS1–16, and have been proposed by Liu and Ong [49]. Again the number of jobs, machines and processing times are taken from a base JSS instance from [43]: la16, la24, la32 and la36, for MSS1-4, MSS5-8, MSS9-12 and MSS13-16 respectively. We add 5, 10, 15 and 20 jobs with no precedences, which are the OSS jobs of the instance. The new jobs have random processing times from 1 to 99. The lower number number of OSS jobs is on the MSS instance with lower index. We could not obtain this set of instances so we generated it.

For OSS we used the instance sets by Taillard [81], Brucker et al. [10], and Guéret and Prins [30]. We denote these instances by their name used in the literature with the prefix OSS.

Inst.	n	m	Inst.	n	m	Inst.	n	m
la01	10	5	la06	15	5	la11	20	5
la02	10	5	la07	15	5	la12	20	5
la03	10	5	la08	15	5	la13	20	5
la04	10	5	la09	15	5	la14	20	5
la05	10	5	la10	15	5	la15	20	5

Table 5.2: Characteristics of the instances used in the computational experiments.

To be able to compare GSS, MSS, and PSS directly we also have derived from the PSS instance set PSSpta01–50 corresponding instance sets GSS (PSS)01–50 for the GSS and MSS (PSS)01–50 for the MSS. Since each PSS instance has 10 replicates, we have 500 instances of this kind for GSS and MSS. Each GSS or MSS instance has the same processing times as the corresponding PSS instance. The order of each job of the new instances is constructed according to the corresponding partial order $\prec_{PSS}$ of PSS. For GSS, the first stage is formed by the minimal elements of the partial order, which are then removed. This is repeated until all operations have been assigned to a stage. For MSS, if $\prec_{PSS}$ contains any precedences, then the corresponding job in the MSS instance has a JSS structure, and we set the job order as a random linear extension of $\prec_{PSS}$. If $\prec_{PSS}$ is empty, then the corresponding job for the MSS instance has an OSS structure, and has no precedences. This creates GSS and MSS instances from PSS, in a way that any solution for the generated GSS or MSS instances is also a solution for the corresponding PSS.

The full set of instances can be found in the supplementary material [91].

5.3 Job Shop Scheduling with Parallel Machines

The tabu search has been evaluated on two distinct sets of instances. The first set contains 30 instances, and is derived from a set of 15 instances for the JSS proposed by Lawrence [43]. The number of jobs n and the number of machines m of these instances are shown in Table 5.2. The 30 instances for the JSS-PM have been obtained by introducing $k = 2$ and $k = 3$ parallel machines for each stage, and replicating each job by the same amount in each of the 15 instances.

These instances were proposed by Rossi and Boschi [74], and have a fixed replication factor k. Such an instance is obtained from an instance of the JSS as follows. Each job j_j from the instance is replicated k times, which the same processing times and the same machine (resp. stage) order. Each machine of the JSS is converted into a stage with k parallel machines. This set contains thirty JSP-PM instances, which are called ROla01–15.k2 and ROla01–15.k3

Note that for these instances, a solution of the corresponding JSS leads to a solution of the JSS-PM, where the k replicated jobs execute in parallel on the k replicated machines. Thus,

	Operation							
Machine	M_1		M_2		M_3		4th	
Job	ord	p	ord	p	ord	p	ord	p
J_1	3	6	4	8	2	4	1	5
J_2	2	14	4	9	1	3	3	16
J_3	3	6	4	13	2	5	1	20

Table 5.3: Example of an instance of the JSS with three jobs and four machines.

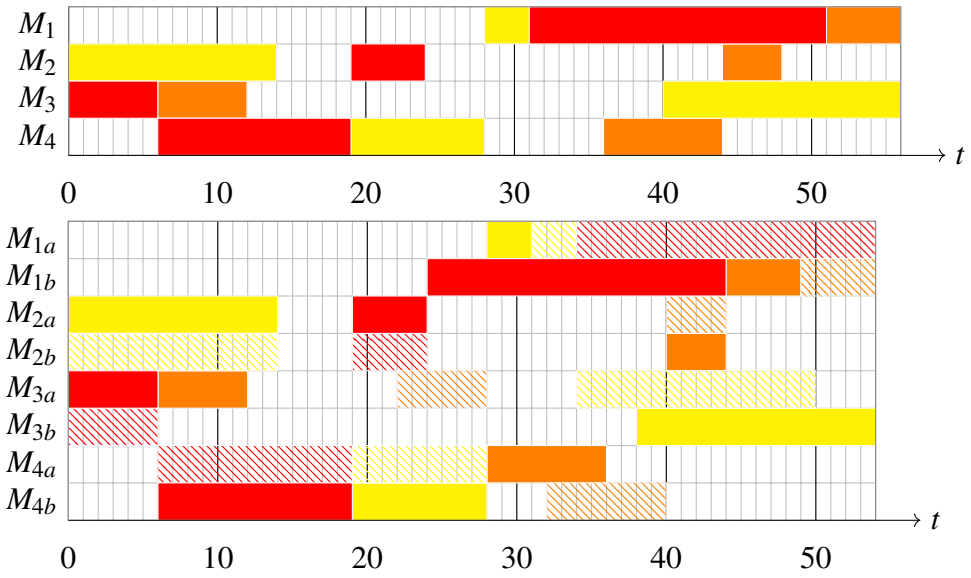


Figure 5.1: Gantt chart with the optimal solutions for a JSS and a JSS-PM instance.

the optimal solution of such a JSS-PM is at most the optimal solution of the corresponding JSS. Rossi and Boschi [74] assert that this is also an optimal solution for JSS-PM, however, the optimal solution of the JSS-PM can be less. Table 5.3 gives an example of an instance of the JSS, where the column "ord" is the order of the operation in the total order of its job. For this JSS instance the corresponding JSS-PM with $k = 2$ has a shorter makespan, as shown in the Gantt charts in Figure 5.1. The top figure shows the Gantt chart of the optimal solution with $C_{\max} = 56$ of an instance of the JSS with three jobs and four machines. On the bottom we see the Gantt chart of the optimal solution with $C_{\max} = 54$ of the corresponding instance with $k = 2$ parallel machines and the double number of jobs. The replicated jobs have the same color, but are hatched. The optimal makespan of the parallel instance is less than the optimal makespan of the base instance.

We also evaluate our solver on a second set of 22 instances proposed by Gholami and Sotskov [24]. The instances of this set are based on instances la10–lat20 for the JSP proposed by Lawrence [43] and instances mt10 and mt20 proposed by Muth and Thompson [57]. Unlike the previous set, each stage can have a variable number of parallel machines. To generate the

Table 5.4: Characteristics of the instances of the literature.

Inst.	n	m	k	Inst.	n	m	k
ROla01–05.k2	20	5	2	GHla01–05	10	5	$[1,5]$
ROla06–10.k2	30	5	2	GHla06–10	15	5	$[1,5]$
ROla11–15.k2	40	5	2	GHla11–15	20	5	$[1,5]$
ROla01–05.k3	30	5	3	GHla16–20	10	10	$[1,5]$
ROla06–10.k3	45	5	3	GHmt10	10	10	$[1,5]$
ROla11–15.k3	60	5	3	GHmt20	20	5	$[1,5]$

instances the authors replicate each machine a random number of times from one to five. The instances in this set are called GHla01–20, GHmt10, and GHmt20.

The characteristics of the instances from the literature are shown in Table 5.4. We also propose a third, new set of 300 instances for the JSP-PM. For each combination of a number of jobs in $\{5,10,15,\ldots,50\}$, a number of machines in $\{5,10,15,\ldots,30\}$, and a maximum number of replications $\bar{k}$ in $\{2,3,4,5,6\}$ we generate a JSP-PM instance. The processing time of each operation is a random number in $[1,99]$, and the number of replications for each machine is a random number in $[1,\bar{k}]$.

6 RESULTS

In this section we report and discuss the experiments for the proposed algorithms and compare it to the results found in the literature. We have implemented the algorithms in C++, and compiled it with the GNU g++ compiler version 5.3.1.

The experiments for PFSS have been done on a PC with an Intel Core i7 930 processor running at 2.80 GHz and 12 GB of main memory. For the PSS solver the experiments have been performed on a PC with an 8-core AMD FX-8150 processor and 32 GB of main memory. Finally for JSS-PM we used a PC with a 3.2 GHz Intel i5-6600 processor and 16 GB of main memory running Ubuntu Linux. In each experiment only one core was used.

6.1 Simple Bubble Search for PFSS

In this section we report on computational experiments done with several variants of Bubble Search starting from the base ordering obtained by the NEHKK1 algorithm presented in Section 4.1.1. We have implemented the NEHKK1 algorithm using the enhancements proposed to run in $O(n^2m)$ and the standard and adaptive variant of randomized Bubble Search.

6.1.1 An Analysis of the N_1 Neighborhood on the Instance Carlier5

To decide the best acceptance criterion for Bubble Search with replacement we first conducted an analysis of the N_1 neighborhood. The analysis has been done in instance 5 proposed by Carlier [12], since it has only 10 jobs and thus can be analyzed exhaustively. A classification of the $10! = 3.628.800$ solutions can be seen in Table 6.1. This table follows the classification of Hoos and Stützle [32].

We can observe that the majority of the solutions are ledges (i.e. the have better, equal, and worse neighbors), followed by slopes. All three optimal solutions of value 7720 are non-strict local minima. Based on this analysis, we opted to not only accept solutions that are strictly better in Bubble Search with replacement, but also solutions of equal value. Preliminary tests have shown that this strategy performs significantly better than accepting only strictly better solutions.

6.1.2 Parameter Adjustment

We next investigated the dependency of randomized Bubble Search with replacement on the parameter α. We selected the first instance in each group, and ran Bubble Search with a time limit of $nm/2 \times 60$ms for $\alpha \in \{0.6, 0.7, 0.8, 0.9, 0.95\}$. Each test was replicated three times. The results of these tests can been seen in Table 6.2. It reports for each value of α the average percent relative deviation from the best known value over all 12 test instances. The smallest

Table 6.1: Characterization of all solutions of instance Carlier5.

Type of solution	Number	%
Isolated	0	0
Strict local maximum	0	0
Plateau	0	0
Local maximum	6	0.00017
Strict local minimum	5	0.00014
Slope	134784	3.71
Local minimum	1743	0.048
Ledge	3492262	96.24

Table 6.2: Results for randomized Bubble Search with replacement for different values of α.

			α (%)					
n	m	NEHKK1	60	70	80	90	95	Adapt.
20	5	1.41	0.00	0.00	0.00	0.00	0.00	0.00
20	10	6.19	**0.46**	0.51	0.91	1.10	1.41	0.52
20	20	4.66	1.34	**1.00**	1.23	1.61	2.25	1.04
50	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50	10	5.95	5.02	4.51	2.75	**1.49**	2.84	2.66
50	20	5.22	4.48	3.97	3.24	**3.03**	3.10	3.14
100	5	0.38	0.00	0.00	0.00	0.00	0.00	0.00
100	10	1.32	0.88	0.63	0.18	0.20	**0.16**	0.16
100	20	5.47	5.47	5.43	5.07	3.85	**3.30**	3.41
200	10	0.97	0.68	0.33	0.13	**0.09**	0.21	0.16
200	20	3.22	3.22	3.22	3.22	2.92	**2.41**	2.24
500	20	2.71	2.71	2.71	2.71	2.57	**2.14**	1.54
Averages		3.12	2.02	1.86	1.62	**1.41**	1.49	1.24

Values are averages over three replicates.

deviations are highlighted in boldface.

The overall best value is $\alpha = 0.9$ with a relative deviation of 1.41. There is however a clear tendency to perform better with larger values of α for an increasing number of jobs. We therefore chose a base value of $\alpha_0 = 8 \times 10^{-4}$ and ran adaptive Bubble Search with $\alpha = \alpha_0^{1/n}$. The choice of α_0 fixes $\alpha = 0.7$ for $n = 20$. The results for adaptive Bubble Search can be seen in the last column (Adapt). Adaptive Bubble Search performs clearly better than any Bubble Search with a fixed alpha, obtaining an overall average percent relative deviation of 1.24. The relative deviation for each instance size is close to the best for the individual α values.

Table 6.2 also contains in column "NEHKK1" the relative deviations of the initial solutions obtained by NEHKK1 from the best known value. We can see that randomized Bubble Search with replacement has the potential to substantially improve over the values found by NEHKK1.

Table 6.3: Results for adaptive randomized Bubble Search with replacement for time scales $nm/2 \times t_0$, $t_0 \in \{0.5, 8, 60\}$ on the complete Taillard instance set.

							Bubble Search			BM
n	m	NEH	NEHKK1	FRB3	$FRB4_{12}$	FRB5	0.5	8	60	8
20	5	3.35	2.81	0.89	1.12	1.08	0.83	0.28	0.22	0.29
20	10	5.02	4.43	1.86	1.79	2.19	2.49	1.77	1.02	1.53
20	20	3.73	3.34	2.18	2.08	1.80	2.24	1.58	1.21	1.54
50	5	0.84	0.67	0.22	0.30	0.19	0.21	0.13	0.09	0.13
50	10	5.12	5.46	2.99	2.89	2.25	3.90	2.76	2.00	2.77
50	20	6.32	6.25	3.38	4.05	3.38	5.08	4.58	4.22	4.58
100	5	0.46	0.43	0.21	0.28	0.16	0.20	0.12	0.08	0.12
100	10	2.13	1.78	0.94	1.22	0.80	1.24	0.84	0.56	0.85
100	20	5.23	5.23	2.90	3.73	2.51	4.58	4.07	3.47	4.07
200	10	1.43	1.11	0.52	0.60	0.38	0.79	0.47	0.40	0.48
200	20	4.52	4.10	2.41	2.95	1.84	3.44	2.95	2.40	2.96
500	20	2.24	2.04	1.06	1.40	0.72	1.67	1.34	1.12	1.34
Averages		3.37	3.14	1.63	1.87	1.44	2.22	1.74	1.40	1.72

Values are averages over five replicates and ten instances.

6.1.3 Experiments on the Full Test Set

Based on the parameter adjustment above, we ran adaptive Bubble Search on all 120 instances. We tested the method on three time scales, namely $nm/2 \times 0.5$ms, $nm/2 \times 8$ms, and $nm/2 \times 60$ms to evaluate independently its utility for quickly improving initial solutions, as well as its limit in the long term. Each test was replicated five times.

The results of these experiments can be seen in Table 6.3. It reports the average percent relative deviation from the best known values for each group of instances for the original NEH heuristics, the improved NEHKK1, the algorithms FRB3, $FRB4_{12}$, and FRB5 proposed by Farahmand et al. [20], and for Bubble Search for the three time scales above.

The experiments confirm the potential of Bubble Search to produce good solutions and can considerably improve over the initial solution obtained by NEHKK1. As expected, the average quality of the solutions improves over time from 2.22% to 1.40%. In average over all instances, in the largest time scale, Bubble Search is able to outperform the best re-insertion heuristic FRB5, although only by a small margin. In the middle time scale its values are competitive in quality with the construction heuristics FRB3 and $FRB4_{12}$. For very short execution times, it still improves the initial values of NEHKK1 by about 1%, but produces about 0.4% to 0.6% higher deviations than other constructive heuristics.

These results have to be judged relative to the computational effort of the different methods. The result of Farahmand et al. [20] were obtained on a PC with a Pentium IV processor running at 3.2 GHz, which is comparable to but slower than our machine. The time of Bubble Search, on the other hand, is dominated by the computation of the new makespan, and is not fully

optimized, since the application of Taillard's improvements are not straightforward to apply to it. The average execution time of FRB3, FRB4$_{12}$ and FRB5 as reported by Farahmand et al. [20] is 2.43, 0.20, and 4.88 seconds, respectively, while Bubble Search needs 0.31, 4.51, and 31.60 seconds on the three time scales. Thus, Bubble Search on the small time scale is comparable to FRB4$_{12}$, on the middle time scale to FRB3 and FRB5.

Looking at the individual instance groups, we can observe that Bubble Search outperforms FRB3 in eight of the 12 groups, in particular for the instances of smaller size. The same holds for three of the smaller instance groups when comparing the short time scale with FRB4$_{12}$. This indicates that Bubble Search may have a potential advantage on short time scales and small instances. For large instances Bubble Search takes more time to obtain good results. This can be explained by the stochastic component of the method, and the need to compute the makespan of the perturbed orderings from scratch.

We finally briefly evaluated if a milder acceptance criterion would be able to improve the results for Bubble Search in the middle time scale. To this end, we used a Metropolis acceptance criterion, that for temperature T accepts solutions that are worse by Δ with probability $e^{-\Delta/T}$. We followed Ruiz and Stützle [75] and chose a fixed base temperature of $\overline{p}/10 \times 0.5$, where $\overline{p}$ is the average processing time over all machines and jobs. We adjusted this temperature such that it is about ten times lower for the largest instances. We ran Bubble Search for $750nm$ iterations, and then enabled the Metropolis acceptance criterion. The result of this test can be seen in the final column (BM) of Table 6.3. The value when using the extended acceptance criterion is better than Bubble Search in the middle time scale, but only slightly. In two of the smaller instances, it obtains better results earlier. In summary, Bubble Search seems relatively robust with respect to the acceptance criterion.

6.2 Iterated Tabu Search for Partial Shop Scheduling

In this section we evaluate the ITS developed for PSS.

6.2.1 Parameter setting

We used the R package irace [53] to calibrate the parameters of ITS. It performs an iterative F-Race [3] over the parameter space, and identifies the setting that yields the best performance. For the calibration we chose from each instance group several instances, which in preliminary experiments have shown to be most sensible to the parameter settings (namely the PSS instances PSSpta11 and PSSpta30, the GSS instances GSSabz7_10, GSSft10_2, GSSadmu14, GSSadmu27, GSSata15 and GSSata41, the MSS instances MSS1 andMSS 5, and the OSS instances OSSta31, OSSbr10, OSSbr47, OSSgu57 and OSSgu68). For the instances which were not available we chose the first one of the generated instances. The parameters and the range of values evaluated during the calibration are shown in Table 6.19. The initial ranges were chosen

Table 6.4: Parameters of ITS and the evaluated intervals.

Parameter	Range	Value
Tabu tenure τ	$[4,15]$	6
Max. no. of tabu search iterations w/o improvement ι	$[500,10000]$	2809
Reduction of the no. of iterations after backtracking δ_I	$[2,10]$	6
Size of the elite list e	$[1,20]$	4
Maximum size of detectable cycles C	$[10,200]$	95
Parameter for the metropolis acceptance criterion α	$[0.0,1.0]$	0.2169
Number of operations to perturb π	$[5,50]$	33

based on preliminary experiments. The budget of runs for the calibration is 10000 and the time limit for each run is $30s$.

The result of the calibration can be seen in column "Value" of Table 6.19.

6.2.2 Methodology

We compare the results obtained by ITS to the state-of-the-art algorithms: for PSS we compare in Section 6.2.5 to Nasiri and Kianfar [60], for GSS in Section 6.2.6 to Liu et al. [50] and Nasiri and Kianfar [59], for MSS in Section 6.2.7 to Liu and Ong [49], and for OSS in Section 6.2.8 to Sha and Hsu [77].

The running times reported in the literature have been normalized to account for differences in the processing speed of each processor by dividing by the factors shown in Table 6.7. We used the Passmark [64] CPU score, and where not available the Dhrystone/Whetstone measurements to determine relative processing speed. The factors have been conservatively rounded up because of the imprecision when comparing algorithms running on different CPUs.

In the tables we report the results of ITS and, when possible compare it to the state-of-the-art algorithms. The relative deviation $(C_{\max} - LB_{qr})/LB_{qr}$ of solution values $C_{\max}$ from the lower bound LB_{qr} is represented as D. The column "Instance" has the index of the instance. When we are comparing ITS to algorithm "A", then the adjusted time used by A is shown in the column T_A, and the deviation D obtained by A is in column D_A . D_{ITS}^{full} shows D obtained by ITS at the end of the 5 minute execution. The column bet^{full} shows how many times ITS obtained results as good or better than A at the end of the execution, while opt^{full} shows the number of times ITS obtained the lower bound, and therefore a guaranteed optimal solution. Notice that the instances may have optimal solutions above the lower bound. The same measurements are provided at an equivalent time to the one used by A: D_{ITS}^{rel} is D, bet^{rel} is the number of times ITS solution had at least the same quality as A, and opt^{rel} is the number of times ITS reached the lower bound. The time $ttb(s)$ is the average time ITS took to reach the best solution it found, and the time $\bar{t}(s)$ is the average time it took to reach the quality of the solution found by A.

When comparing to other algorithms, we run ITS with the normalized time limit reported in

Table 6.5: Factor for adjustment of processing times.

Reference	Environment	Factor
Nasiri and Kianfar [60]	Core Duo 2 T2400, 1.83 Ghz	2
Liu et al. [50]	Pentium IV, 1.8 GHz	6
Nasiri and Kianfar [59]	Core Duo 2 T2400, 1.83 Ghz	2
Liu and Ong [49]	Pentium II, 500 MHz	20
Sha and Hsu [77]	AMD Athlon 1800+	3

Table 6.6: Comparing the loss of solution quality when using GSS and MSS to model a PSS instance.

	PSS			GSS			MSS		
Group	LB_{qr}	D_{300s}	$ttb(s)$	LB_{qr}	D_{300s}	$ttb(s)$	LB_{qr}	D_{300s}	$ttb(s)$
01–10	937.10	3.01	46.04	940.91	6.36	116.17	994.01	33.82	59.61
11–20	1182.50	0.00	0.03	1183.83	0.28	5.78	1231.96	15.42	113.52
21–30	1245.50	1.26	116.51	1250.53	4.48	171.59	1325.75	30.20	114.52
31–40	1728.80	0.00	0.02	1729.41	0.09	0.61	1741.98	2.71	112.60
41–50	1752.50	0.00	0.13	1753.68	0.19	18.49	1775.58	11.13	195.02
Avg.	1369.08	0.85	32.55	1371.67	2.27	62.71	1413.86	18.64	121.20

the literature. For problem sets PSS, GSS, and MSS this is the time for the maximum number of iterations or for reaching the lower bound, for problem set OSS this is a time limit. Note that for the former three groups running our algorithm for the reported time is strongly biased against our method, since even if two algorithms have the same average running time and solution quality, running one with the average time of the other will lead to apparent worse results. We also executed ITS for a total of 5 minutes to evaluate its performance with additional time.

For the scatter plots we compare ITS with the state-of-the-art algorithms. The time used for the comparison is always the time reported for the algorithm with which we are comparing ITS.

6.2.3 Comparison of PSS, GSS and MSS

First we use ITS to compare the PSS, GSS and MSS problems. We evaluate the lower bound and quality of the solutions produced by ITS on the PSSpta01–50, GSS(PSS)01–50 and MSS(PSS)01–50 sets of instances. The GSS and MSS sets of instances are built from the PSS set, in such a way that a solution for the two former sets is always a solution to the corresponding solution of the latter set as described in Section 5.2.

Table 5.1 shows the average order strength of the instances. We can see that MSS was converted to only linear orders, with order strength 1.0. GSS is able to represent some of the flexibility of PSS, but the order strength of the instances is in average 0.55 compared to 0.12 for

the PSS instances.

In Table 6.6 we see the results of ITS for PSSpta01–50 under PSS, for GSS(PSS)01–50 under GSS and for MSS(PSS)01–50 under MSS. For each instance group, columns LB_{qr} show average lower bounds, columns D_{300s} the average deviation from the PSS lower bound of the solutions produced at 300 seconds, and columns $ttb(s)$ the average time to find the best solution. In order to make the quality of the solutions comparable, we used only the lower bound of the PSS instance to compute the relative deviations of the corresponding GSS and MSS instances.

As expected, we can see that the lower bound of the PSS instances is smaller than for GSS and MSS. The higher flexibility of PSS also leads to significantly better solutions, in average about 1.4% compared to GSS and more than 17% compare to MSS. The average time of 32.55s to find these solutions is also smaller for PSS. For GSS instances the time to find the best solution is in average about twice as long, and for MSS instances four times as long. Thus a larger search space not necessarily leads to more longer computation times. In summary, this experiment shows that the increased flexibility of PSS in modeling instances leads to better solutions.

6.2.4 Component Analysis

In this section we evaluate the influence of some of the most important components of ITS. To perform this evaluation we chose a subset of representative instances. We selected instance for which the time to best is large, which we consider as a metric to define the harder instances for ITS, and therefore will more likely show the influence of the different components. We selected a few instances from each benchmark set. For PSS the two instances with largest $ttb(s)$ is first PSSata23 and then PSSata26, but both have the same n and m, so to select a more representative set of instances we selected the third largest $ttb(s)$ which is PSSpta06. The selected instances are: PSSpta06, PSSpta23, GSSft10_02, GSSla38_4, GSSata47, GSSadmu08, MSS1, MSS13, OSSgp09-10, OSSj6-per0-0 and OSSta31. The analysis is performed at the end of the 300 seconds.

To evaluate the influence of trimming the neighbourhood shown in Section 4.2.2.1 we measured the average number of state transitions in ITS. Each state transition consists in performing the neighbourhood search procedure and selecting the new solution to be visited. The trimming accelerates this procedure by discarding a part of the neighbourhood that does not have to be evaluated. The average number of state transitions is 44381 per second without the trimming, and 91788 per second with the trimming procedure, therefore the algorithm performs about two times faster with this technique.

Table 6.7 shows the results at 300s for the different configurations of the components of ITS. The column instance show the instance name. The column ITS shows the quality of the solution produced by the ITS with the best configuration we found. It is the configuration we used to compare it to the state of the art of PSS, MSS, GSS, and OSS bellow. The column

Tabu shows the results of the tabu search in ITS before it makes the first perturbation, while the column $Tabu_{50k}$ shows it exchanging the Max number of tabu search iterations without improvement ι of 50000 instead of 2809 used in ITS. The *Restart* column shows the value with a random solution generation in place of the perturbation, while *Bubble* shows the ITS solution substituting the perturbation, by the perturbation described in Section 4.2.4.1.

Table 6.7: Results for the component modifications in ITS.

Instance	*ITS*	*Tabu*	$Tabu_{50k}$	*Restart*	*Bubble*
PSSpta06	2,57	5.17	5.06	4.72	4.27
PSSpta23	2.45	7.65	6.66	5.06	4.22
GSSft10_02	22.73	26.01	24.47	22.78	22.22
GSSla38_4	5.34	5.93	5.51	5.51	5.51
GSSata47	5.07	9,40	5.62	7.34	7.16
GSSadmu08	13.00	15.90	13.08	13.00	13.46
MSS1	3.43	5.12	5.12	5.12	2.56
MSS13	1.63	6.82	4.33	2.72	2.72
OSSgp09-10	0.25	7.73	7.73	2.61	0.54
OSSj6-per0-0	0.90	5.01	5.01	3.12	1.61
OSSta31	0.85	2.51	2.51	2.20	0.47
Average	5.29	8.84	7.74	6.74	5.89

We see that the proposed ITS outperforms the other configurations, while substituting the perturbation by the bubble perturbation is not very detrimental to the quality of the solution, increasing the deviation by only 0.6%.

6.2.5 Partial Shop

In this sub-section we evaluate the performance of ITS on the PSS instances. We compare the results with the *Hybrid Scatter Search* (HSS) of Nasiri and Kianfar [60]. Nasiri and Kianfar [60] report the results obtained for HSS for the 50 smaller instances. We show a comparison with ITS in Table 6.8.

The average relative deviation of ITS is 0.99% compared to 10.14% for HSS. In 34 instances (01, 11–20, 24–25, 27, and 31–50) ITS always obtained the lower bound LB_{qr}. ITS was able to reach the quality of the solution obtained by HSS in 0.02 s in average, never taking more than 0.2 s. The results show that ITS clearly outperforms HSS for PSS, that these instances are mostly easy to solve, and that LB_{qr} is a good lower bound for them. The small variations of the relative deviations of ITS for the instance groups show that the solution quality depends only weakly on the generated precedences. In 5 minutes the average relative deviation of ITS on instances PSSata01–50 improves to 0.85%, and the average time to find the best solution is 32.55*s*.

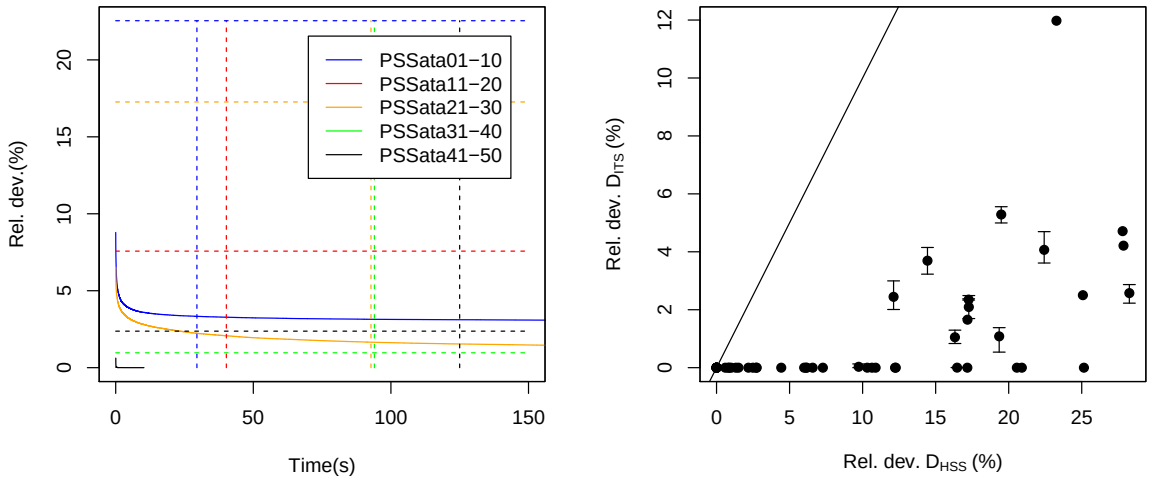
Table 6.8: Comparing ITS with the HSS of Nasiri and Kianfar [60]

Instance	T_{HSS}	D_{HSS}	D_{ITS}^{full}	bet^{full}	opt^{full}	$ttb(s)$	D_{ITS}^{rel}	bet^{rel}	opt^{rel}	$\bar{t}(s)$
PSSata01	29.75	25.15	0.00	100	100	2.14	**0.00**	100	100	0.00
PSSata02	28.85	25.08	2.50	100	0	5.24	**2.50**	100	0	0.00
PSSata03	28.90	17.26	2.33	100	0	14.37	**2.35**	100	0	0.00
PSSata04	29.50	27.80	4.71	100	0	3.20	**4.71**	100	0	0.00
PSSata05	29.35	27.86	4.21	100	0	1.91	**4.21**	100	0	0.00
PSSata06	29.95	28.26	1.00	100	3	200.99	**2.57**	100	0	0.00
PSSata07	29.30	17.27	1.63	100	0	75.45	**2.09**	100	0	0.00
PSSata08	29.80	23.27	11.98	100	0	1.75	**11.98**	100	0	0.00
PSSata09	30.15	17.18	1.66	100	0	4.29	**1.66**	100	0	0.00
PSSata10	29.55	16.32	0.11	100	62	151.01	**1.05**	100	1	0.00
PSSata11	40.55	2.71	0.00	100	100	0.02	**0.00**	100	100	0.01
PSSata12	41.45	10.32	0.00	100	100	0.01	**0.00**	100	100	0.01
PSSata13	40.00	7.28	0.00	100	100	0.01	**0.00**	100	100	0.01
PSSata14	38.90	1.50	0.00	100	100	0.02	**0.00**	100	100	0.01
PSSata15	40.60	10.64	0.00	100	100	0.03	**0.00**	100	100	0.01
PSSata16	39.80	12.22	0.00	100	100	0.15	**0.00**	100	100	0.01
PSSata17	42.75	6.58	0.00	100	100	0.02	**0.00**	100	100	0.01
PSSata18	38.30	10.89	0.00	100	100	0.03	**0.00**	100	100	0.01
PSSata19	39.80	1.32	0.00	100	100	0.01	**0.00**	100	100	0.01
PSSata20	39.90	12.27	0.00	100	100	0.02	**0.00**	100	100	0.01
PSSata21	85.00	22.43	3.20	100	0	205.50	**4.07**	100	0	0.01
PSSata22	96.20	9.73	0.00	100	99	55.31	**0.03**	100	86	0.01
PSSata23	89.55	12.12	1.51	100	0	234.93	**2.45**	100	0	0.02
PSSata24	92.75	20.90	0.00	100	100	6.86	**0.00**	100	100	0.01
PSSata25	90.45	20.56	0.00	100	100	3.04	**0.00**	100	100	0.01
PSSata26	94.25	14.44	2.76	100	0.00	233.58	**3.69**	100	0.00	0.01
PSSata27	96.95	17.17	0.00	100	100	7.53	**0.00**	100	100	0.01
PSSata28	91.95	16.47	0.00	100	100	28.94	**0.00**	100	98	0.01
PSSata29	96.95	19.35	0.44	100	19	208.61	**1.08**	100	2	0.01
PSSata30	93.50	19.49	4.68	100	0	180.80	**5.29**	100	0	0.01
PSSata31	119.60	0.79	0.00	100	100	0.02	**0.00**	100	100	0.01
PSSata32	128.65	2.75	0.00	100	100	0.01	**0.00**	100	100	0.01
PSSata33	62.65	**0.00**	0.00	100	100	0.02	**0.00**	100	100	0.02
PSSata34	113.70	0.60	0.00	100	100	0.01	**0.00**	100	100	0.01
PSSata35	112.35	0.87	0.00	100	100	0.02	**0.00**	100	100	0.01
PSSata36	47.75	**0.00**	0.00	100	100	0.02	**0.00**	100	100	0.02
PSSata37	14.45	**0.00**	0.00	100	100	0.02	**0.00**	100	100	0.02
PSSata38	99.55	**0.00**	0.00	100	100	0.02	**0.00**	100	100	0.02
PSSata39	128.40	2.49	0.00	100	100	0.02	**0.00**	100	100	0.02
PSSata40	112.90	2.18	0.00	100	100	0.03	**0.00**	100	100	0.02
PSSata41	140.75	6.00	0.00	100	100	0.05	**0.00**	100	100	0.03
PSSata42	75.70	**0.00**	0.00	100	100	0.03	**0.00**	100	100	0.03
PSSata43	151.00	1.00	0.00	100	100	0.08	**0.00**	100	100	0.05
PSSata44	111.60	**0.00**	0.00	100	100	0.04	**0.00**	100	100	0.04
PSSata45	120.95	**0.00**	0.00	100	100	0.06	**0.00**	100	100	0.06
PSSata46	113.50	**0.00**	0.00	100	100	0.03	**0.00**	100	100	0.03
PSSata47	150.15	4.43	0.00	100	100	0.07	**0.00**	100	100	0.03
PSSata48	142.45	6.12	0.00	100	100	0.04	**0.00**	100	100	0.03
PSSata49	92.85	**0.00**	0.00	100	100	0.03	**0.00**	100	100	0.03
PSSata50	150.90	6.18	0.00	100	100	0.87	**0.00**	100	100	0.03
Average	76.29	10.14	0.85	100	73.66	32.55	**0.99**	100	71.74	0.02

In the graph in the left of Figure 6.1 we show the evolution with time of the quality of the solution produced by ITS. We grouped the instances by size and associated each with a color, and each group corresponds to a set of 10 instances. The horizontal dashed lines show the quality of the solution produced by HSS, while the vertical dashed lines show the time reported for HSS. We show the time only up to 150 seconds despite the execution lasting 300 seconds. For PSSata01-10 and PSSata21-30 still improve the solution quality up to close to 300 seconds but the improvement after 150 seconds is negligible. The quality of the solution for PSSata11-

20 stagnates in a second, and for 41-50 it stagnates after 10 seconds. For PSSata31-40 ITS reaches the lower bound LB_{qr} in a second. It shows that ITS converges very fast for these PSS instances, and produces better solutions substantially better and faster than HSS. Notice that the black, green and red curves are not shown because the instances converge to the value of lower bound very fast, most in under 0.1 seconds.

The scatter plot on the right of Figure 6.1 compares for each of the 50 instances the average relative deviation D_{HSS} to the minimum, average and maximum average relative deviation D_{ITS} over the 10 generated instances.



(a) Time evolution of the quality of the solution.

(b) Scatter plot comparison.

Figure 6.1: Graphs for evaluating the ITS and the HSS on the PSS instances.

The instances PSSpta51–80 are shown in Table 6.9, and there is no direct comparison with HSS, since Nasiri and Kianfar [60] only report that the lower bound LB_{qr} could be obtained in all replications with different seeds in less than a second. The same holds for ITS, and the average time to obtain the lower bound is $0.19s$.

6.2.6 Group Shop

In this subsection we evaluate ITS on the GSS instances. We evaluate separately the instances proposed by Blum and Sampels [7] and Nasiri and Kianfar [59].

6.2.6.1 Instances by Blum and Sampels [7]

We first evaluate ITS on GSS on the instances from Blum and Sampels [7], and compare it to the state-of-the-art tabu search of Liu et al. [50].

Table 6.9: IGA results for the large instances of Nasiri and Kianfar [60]

Instance	D_{ITS}^{full}	opt^{full}	$ttb(s)$
PSSata51	0.00	100	0.05
PSSata52	0.00	100	0.05
PSSata53	0.00	100	0.06
PSSata54	0.00	100	0.05
PSSata55	0.00	100	0.05
PSSata56	0.00	100	0.05
PSSata57	0.00	100	0.05
PSSata58	0.00	100	0.05
PSSata59	0.00	100	0.05
PSSata60	0.00	100	0.06
PSSata61	0.00	100	0.11
PSSata62	0.00	100	0.10
PSSata63	0.00	100	0.09
PSSata64	0.00	100	0.09
PSSata65	0.00	100	0.11
PSSata66	0.00	100	0.10
PSSata67	0.00	100	0.10
PSSata68	0.00	100	0.09
PSSata69	0.00	100	0.09
PSSata70	0.00	100	0.10
PSSata71	0.00	100	0.43
PSSata72	0.00	100	0.44
PSSata73	0.00	100	0.42
PSSata74	0.00	100	0.42
PSSata75	0.00	100	0.42
PSSata76	0.00	100	0.44
PSSata77	0.00	100	0.43
PSSata78	0.00	100	0.41
PSSata79	0.00	100	0.46
PSSata80	0.00	100	0.45
Average	0.00	100	0.19

Table 6.10 shows the results of the tests. The average relative deviation of ITS is 6.69%, and for the tabu search 7.07%. With additional time ITS obtained a relative deviation of 6.36% in an average time of 42.82 s. For all base instances and for all stage sizes the relative deviations for ITS are lower than those of the tabu search. ITS obtained the same or a better result in 36 of the 40 instances, and strictly better results in 21.

The difference in the overall average relative deviation is 0.38%. ITS was able to reach the solution quality of the tabu search in 375 of the 400 runs, and with more time was able to reach the values reported for the tabu search in all runs. The average time ITS takes to reach the solution quality of the tabu search is 6.48 s. Thus ITS is about a factor of 4 faster compared to the average running time of 27.35 s for the tabu search. ITS was able to reach the lower bound LB_{qr} in 98 of the 400 runs, and in six instances in all 10 replications. With additional time ITS was able to reach the lower bound 125 times in the 400 runs, and in 12 instances in all 10 replications. In summary, ITS is consistently better than the tabu search, but only by a slight margin.

Figure 6.2 shows the scatter plot comparing ITS to the tabu search. It shows that the quality of the solutions of ITS and the tabu search are very similar, with a slight advantage for ITS. Which is compatible with the difference of 0.38% difference in D_{ITS}^{rel}, and D_{TS}. Moreover we can see that the difficulty to reach the lower bound for both algorithms is correlated.

Table 6.10: Comparing ITS solver with TS of Liu et al. [50] for GSS

Instance	T_{TS}	D_{TS}	D_{ITS}^{full}	bet^{full}	opt^{full}	$ttb(s)$	D_{ITS}^{rel}	bet^{rel}	opt^{rel}	$\bar{t}(s)$
GSSabz7_1	46.02	5.17	4.09	10	0	123.85	**4.50**	10	0	0.54
GSSabz7_2	45.05	**8.09**	8.09	10	0	1.28	**8.09**	10	0	1.28
GSSabz7_3	45.98	**7.18**	7.18	10	0	1.19	**7.18**	10	0	1.19
GSSabz7_4	44.67	**6.65**	6.65	10	0	5.00	**6.65**	10	0	5.00
GSSabz7_5	45.08	**14.75**	14.75	10	0	0.14	**14.75**	10	0	0.14
GSSabz7_6	40.77	**5.45**	5.45	10	0	2.24	**5.45**	10	0	2.24
GSSabz7_7	54.33	**0.00**	0.00	10	10	36.02	0.37	8	8	36.02
GSSabz7_8	45.77	**3.78**	3.78	10	0	0.13	**3.78**	10	0	0.13
GSSabz7_9	47.52	**3.78**	3.78	10	0	23.78	**3.78**	10	0	23.78
GSSabz7_10	44.98	**10.07**	10.07	10	0	6.84	**10.07**	10	0	6.84
GSSabz7_11	43.08	**7.58**	7.58	10	0	0.01	**7.58**	10	0	0.01
GSSabz7_12	43.83	4.79	4.59	10	0	0.01	**4.59**	10	0	0.01
GSSabz7_13	41.97	3.13	2.65	10	0	0.02	**2.65**	10	0	0.01
GSSabz7_14	42.47	**0.00**	0.00	10	10	0.02	**0.00**	10	10	0.02
GSSabz7_15	2.52	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
GSSft10_1	10.38	17.24	16.83	10	0	2.24	**16.83**	10	0	2.24
GSSft10_2	10.92	22.84	22.32	10	0	141.41	**22.73**	9	0	8.79
GSSft10_3	10.53	22.68	20.37	10	0	125.48	**21.28**	10	0	0.06
GSSft10_4	11.25	20.78	18.00	10	0	127.45	**18.98**	10	0	0.92
GSSft10_5	10.22	14.27	13.74	10	0	5.88	**13.77**	10	0	0.21
GSSft10_6	10.50	11.05	10.69	10	0	7.74	**10.75**	10	0	3.00
GSSft10_7	7.62	5.79	3.24	10	0	94.08	**4.76**	8	0	6.30
GSSft10_8	2.10	**0.00**	0.00	10	10	1.72	0.05	8	8	1.72
GSSft10_9	0.13	**0.00**	0.00	10	10	0.05	**0.00**	10	10	0.05
GSSft10_10	0.08	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
GSSla38_1	32.70	27.92	26.83	10	0	27.60	**26.98**	10	0	4.45
GSSla38_2	31.32	17.45	16.61	10	0	102.53	**17.23**	4	0	49.54
GSSla38_3	30.33	11.24	9.59	10	0	172.21	**11.22**	10	0	12.62
GSSla38_4	33.32	5.71	4.32	10	0	200.86	**5.34**	8	0	25.51
GSSla38_5	34.62	5.62	4.98	10	0	36.50	**5.15**	10	0	0.48
GSSla38_6	35.65	2.94	0.47	10	2	162.00	**1.80**	9	0	13.74
GSSla38_7	35.17	2.07	0.00	10	10	60.78	**0.47**	10	4	5.68
GSSla38_8	28.17	0.54	0.00	10	10	12.97	**0.03**	10	9	3.84
GSSla38_9	35.38	2.20	1.06	10	3	113.34	**1.95**	10	0	5.89
GSSla38_10	32.12	5.98	2.86	10	0	54.29	**4.17**	10	0	1.29
GSSla38_11	33.70	5.65	3.82	10	0	25.52	**4.03**	10	0	0.91
GSSla38_12	19.85	**0.31**	0.00	10	10	36.47	0.33	4	2	33.52
GSSla38_13	1.63	**0.00**	0.00	10	10	1.35	0.14	7	7	1.35
GSSla38_14	1.25	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
GSSla38_15	1.22	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
Average	27.35	7.07	6.36	10.00	3.12	42.82	**6.69**	9.38	2.45	6.48

Figure 6.3 shows the evolution of the solution quality of the ITS. The instances of Blum and Sampels [7] have been generated using as a base the JSS instances abz7, ft10, and la38. For an instance with m machines, m instances have been generated, by dividing each job into $\lceil m/s \rceil$ stages of size $s \in [1,m]$ (if s does not divide m, the last stage has size $m \mod s$). Since abz7, ft10, and la38 have 15, 10, and 15 machines, we have a total of 40 instances. In the left, on Figure 6.3a, the instances are grouped by the base JSS instance, while on the right, on Figure 6.3a, the instances are grouped by stage size.

For Figure 6.3a we show the evolution of the quality of the solution produced by ITS until 60 seconds. By that time for both the GSSab7_x instances and ft10_x instances ITS gets to 0.13% or less of the deviation at 300 seconds. For the GSSla38_x instances ITS still improve D from 5.08 at 60 seconds to 4.71 to 4.71 at 300 seconds. The graph shows that the majority of the improvement in the quality of the solution produced is done in the first 10 seconds, and the behavior of the quality of the solution produced by ITS the 3 instances group.

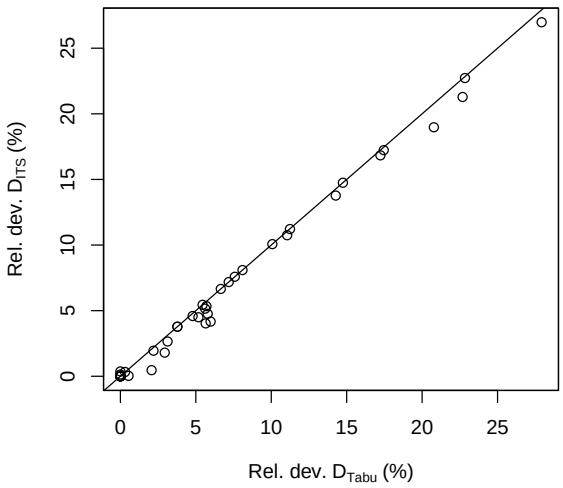
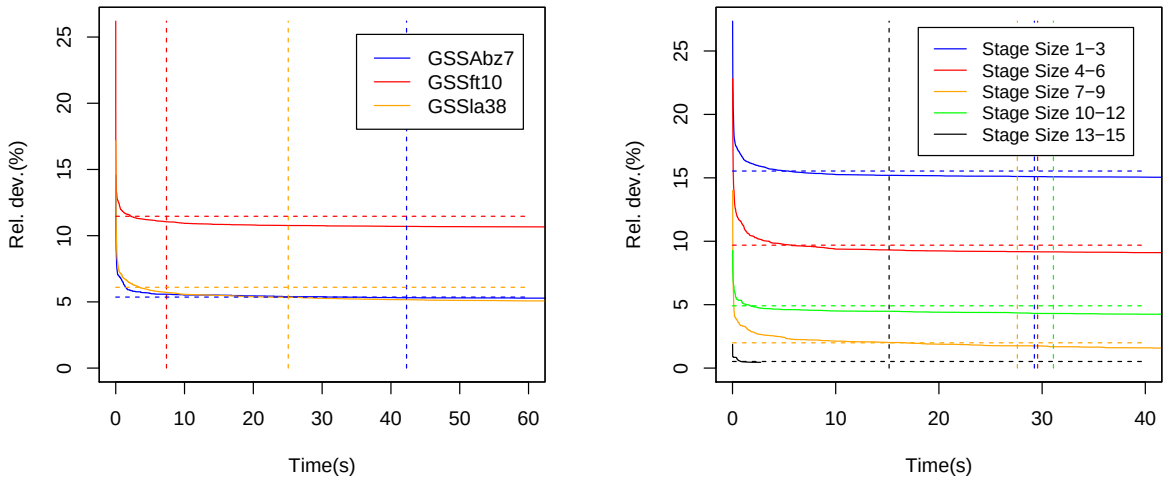


Figure 6.2: Scatter plot comparison of the ITS and the tabu search for the GSS instances by Blum and Sampels [7].



(a) Grouped by JSS instance.

(b) Grouped by stage size

Figure 6.3: Time evolution of the quality of the solution for GSS on the instances by Blum and Sampels [7].

For Figure 6.3b we also cut the time axis, to highlight the most interesting part of the evolution of the solution quality of ITS. The deviation D changes by 0.37% on the groups with stage size 3 or less, and improves less on the other groups. In particular for the group with stage sizes 13 to 15 the solution quality stagnates after 2.72s very close to 0%.

In both graphs we can see that the comparison of the time evolution with the dashed lines

Table 6.11: Comparison of quality of the solution produced by the ITS and the number of possible arrangements for the jobs.

ss	$D^{\dagger}_{ITS}$	V
1	15.46	1.00
2	12.35	2.11
3	8.38	3.89
4	5.49	4.92
5	9.87	6.24
6	2.96	6.49
7	0.00	7.40
8	1.89	8.31
9	2.42	8.42
10	6.47	8.64
11	5.70	8.98
12	2.30	9.46
13	1.33	10.10
14	0.00	10.94
15	0.00	12.12

which represent the time and quality of the solutions produced by the tabu search also show that the main advantage of ITS over the tabu search is produces good solutions much faster.

Table 6.11 shows a comparison of the average quality of the solution produced by ITS at 300 seconds, and the number of ways to arrange the operations in each job. We removed the instances GSSft10. This instance groups does not have instances with groups size larger than 10, and both ITS and the tabu search produce solutions with higher D for this instance group, therefore including them would skew the results. For the table in the left the column *ss* shows the stage size, $D^{\dagger}_{ITS}$ shows the average deviation for ITS at 300 seconds for this subset of instances, and V shows $log_{10}(x)$ where x is the number of arrangements of the operations of each job for that stage size. The figure in the right plots $D^{\dagger}_{ITS}(V)$. There is a strong inverse correlation between $D^{\dagger}_{ITS}$ and V of 0.86.

6.2.6.2 *Instances by Nasiri and Kianfar [59]*

The papers [59] and [58] work on this instances for GSS. We do not have full confidence in the results presented by the authors. Most instances are available as a supplementary material of [58], however the instances GSSata71-80 are not available.

Next we compare the performance of ITS with the ABC algorithm of Nasiri [58]. It must be noted that the results for GSSadmu06 the solution is 111 units under the sum of the costs of the operations of the first job. Which makes the reported solution impossible. The first job has an operation with processing time of 111, which may be missing from the schedule. The schedules for the solutions were not provided, and the authors did not respond to contact attempts, so we

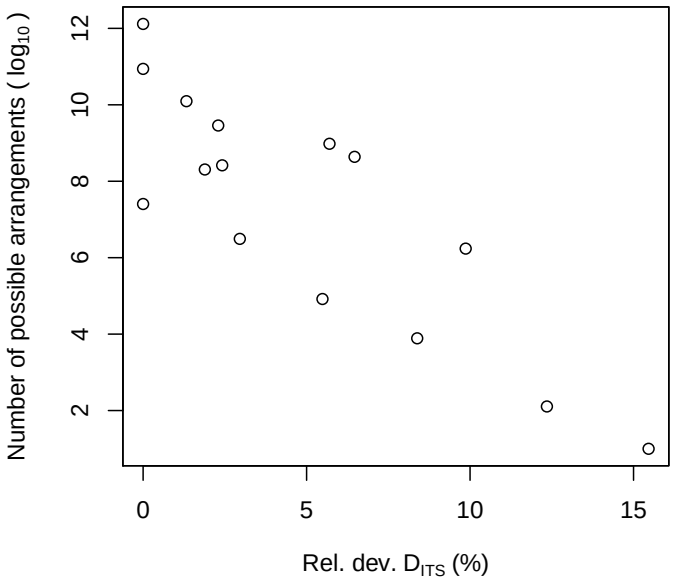


Figure 6.4: Plot showing the deviation of the solutions produced by ITS a the logarithm of the possible arrangements for each job.

cannot be sure of the error.

In Table 6.12 we compare the ABC algorithm of Nasiri [58] to ITS on the GSSadmu instances. On these instances ABC achieves an average relative deviation of 1.08%, and ITS of 2.99%. On GSSagmu06 the authors report a solution better than the lower bound. ABC ITS finds better results than GA/TS in 17 instances, in another 23 ABC is better than ITS. The difference in the average relative deviation is explained by its uneven distribution, on the instances where ITS obtains a better result, the difference is at most 0.02%. In all those cases ITS found the optimal solution, while ABC found solution slightly above this value. The exception is instance GSSadmu07 where ITS found a solution 8.46% better than ABC. In seven instances (01,02,03,13,30,37,39) GA/TS finds solutions close to the lower bound whereas ITS has relative deviations ranging from 3.1% to 11.6%. In 300 s the average relative deviation of ITS decreases to 1.64% with an average time to best of 50.91 s. ITS now finds solutions of similar quality, except for seven instances (01,02,03,05,06,10,13,18). Excluding these instances the average time to best is 7.45s, and the average time to the target solution of ABC is $\bar{t} = 10.18$ s and shows a similar behavior. In summary, both algorithms show a similar performance, except for a few outliers. There seems to be no apparent correlation of the performance of ITS on these instances with basic parameters such as size or order strength. In Figure 6.6 we see the graphs with the quality of the solution evolution with time, and the scatter plot both show a better performance by ABC.

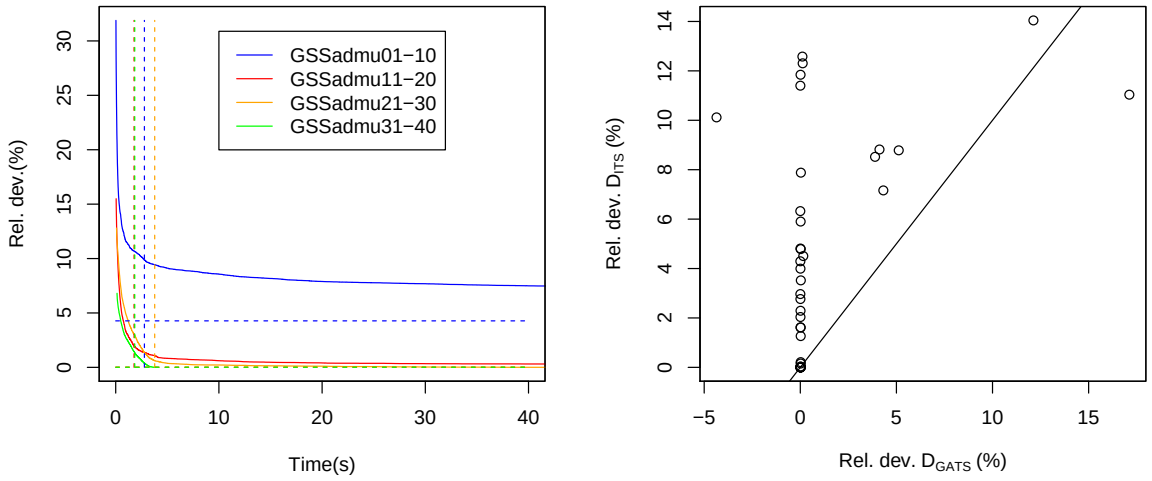
In Table 6.13 we compare the ABC algorithm of Nasiri [58] to the results of ITS on the 50

Table 6.12: Comparing ITS solver with GA/TS of Nasiri [58] for GSS

Instance	T_{abc}	D_{abc}	D_{ITS}^{full}	bet^{full}	opt^{full}	$ttb(s)$	D_{ITS}^{rel}	bet^{rel}	opt^{rel}	$\bar{t}(s)$
GSSadmu01	5.98	**0.15**	4.36	0	0	2.28	4.36	0	0	
GSSadmu02	6.76	**0.11**	6.57	0	0	229.71	11.03	0	0	
GSSadmu03	5.82	**0.12**	9.11	0	0	143.40	11.63	0	0	
GSSadmu04	9.84	**3.89**	3.98	5	0	160.94	7.38	0	0	
GSSadmu05	8.10	**4.12**	6.98	0	0	141.32	7.77	0	0	
GSSadmu06	15.86	**-4.36**	4.93	0	0	203.94	7.01	0	0	
GSSadmu07	14.64	17.12	6.52	10	0	208.03	**8.66**	10	0	0.34
GSSadmu08	15.52	**12.13**	10.23	10	0	239.59	13.00	1	0	66.22
GSSadmu09	12.46	**5.12**	4.98	7	0	202.86	6.86	0	0	
GSSadmu10	16.22	**4.32**	5.53	0	0	158.71	6.50	0	0	
GSSadmu11	2.82	0.01	0.00	10	10	0.69	**0.00**	10	10	0.69
GSSadmu12	2.42	0.02	0.00	10	10	0.16	**0.00**	10	10	0.16
GSSadmu13	2.76	**0.02**	1.97	0	0	13.29	3.10	0	0	
GSSadmu14	4.88	**0.02**	0.00	10	10	3.88	0.25	8	8	3.88
GSSadmu15	3.18	**0.00**	0.00	10	10	8.55	0.63	1	1	8.55
GSSadmu16	5.36	0.01	0.00	10	10	3.18	**0.00**	10	10	3.18
GSSadmu17	8.34	0.01	0.00	10	10	4.69	**0.00**	10	10	4.69
GSSadmu18	23.38	**0.03**	0.23	4	4	175.25	1.70	0	0	
GSSadmu19	12.18	0.02	0.00	10	10	4.62	**0.00**	10	10	4.62
GSSadmu20	5.82	**0.00**	0.00	10	10	17.98	1.20	0	0	17.98
GSSadmu21	10.48	0.03	0.00	10	10	0.29	**0.00**	10	10	0.29
GSSadmu22	0.40	**0.00**	0.00	10	10	0.57	1.85	0	0	0.57
GSSadmu23	27.56	0.01	0.00	10	10	0.61	**0.00**	10	10	0.61
GSSadmu24	16.32	0.02	0.00	10	10	0.60	**0.00**	10	10	0.60
GSSadmu25	40.42	**0.02**	0.00	10	10	28.04	0.04	7	7	28.04
GSSadmu26	10.46	**0.01**	0.00	10	10	29.18	0.76	0	0	29.18
GSSadmu27	13.60	0.01	0.00	10	10	7.23	**0.00**	10	10	7.23
GSSadmu28	6.50	0.02	0.00	10	10	3.81	**0.00**	10	10	3.81
GSSadmu29	23.84	**0.02**	0.00	10	10	20.82	0.30	7	7	20.82
GSSadmu30	1.56	**0.01**	0.00	10	10	3.69	5.78	0	0	3.69
GSSadmu31	11.84	0.01	0.00	10	10	0.19	**0.00**	10	10	0.19
GSSadmu32	0.42	0.01	0.00	10	10	0.34	**0.00**	10	10	0.34
GSSadmu33	14.34	0.01	0.00	10	10	1.23	**0.00**	10	10	1.23
GSSadmu34	0.30	**0.01**	0.00	10	10	0.68	2.35	0	0	0.68
GSSadmu35	13.34	0.01	0.00	10	10	0.85	**0.00**	10	10	0.85
GSSadmu36	16.18	0.00	0.00	10	10	3.43	**0.00**	10	10	3.43
GSSadmu37	0.96	**0.03**	0.00	10	10	2.77	4.28	0	0	2.77
GSSadmu38	15.26	0.02	0.00	10	10	3.65	**0.00**	10	10	3.63
GSSadmu39	0.38	**0.01**	0.00	10	10	2.10	5.55	0	0	2.10
GSSadmu40	0.80	**0.00**	0.00	10	10	3.20	7.70	0	0	3.20
Average	10.18	**1.08**	1.64	7.90	7.10	50.91	2.99	4.85	4.58	

GSSata instances. On these instances ABC achieves better results than ITS: the average relative deviation for all instances is 7.78% for ABC, and 10.31% for ITS, and ABC is better in 43 of the 50 instances. Different from the instance set GSSadmu there are no outliers, and the scatter plot shows that the solution quality of both algorithms is highly correlated. On the remaining instances, the average time-to-target is 37.4 s. When running with a time limit of 300 s, the average relative deviation of ITS improves to 7.41%, with an average time of 116.64 s to the best value. Figure 6.6 we see the usual graphs which show that ABC outperforms ITS.

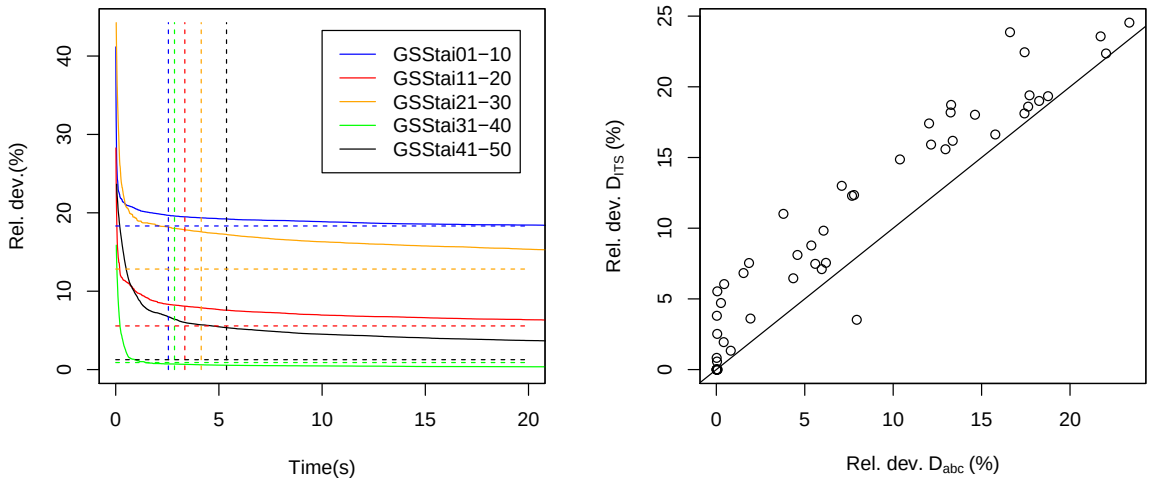
Instances GSSata51–80 are shown in the Table 6.14. Nasiri [58] report that these instances are easy to solve. ITS solves them in an average time of 7.8 s.



(a) Time evolution of the quality of the solution.

(b) Scatter plot comparison

Figure 6.5: Graphs for evaluating the ITS and the ABC algorithm on the GSS instances by Nasiri and Kianfar [59] based on the test set by Demirkol et al. [17].



(a) Time evolution of the quality of the solution.

(b) Scatter plot comparison

Figure 6.6: Graphs for evaluating the ITS and the ABC algorithm on the GSS instances by Nasiri and Kianfar [59] based on the test set by Taillard [81].

Table 6.13: Comparing ITS solver with GA/TS of Nasiri and Kianfar (2011) for GSS

Instance	T_{abc}	D_{abc}	D_{ITS}^{full}	bet^{full}	opt^{full}	$ttb(s)$	D_{ITS}^{rel}	bet^{rel}	opt^{rel}	$\bar{t}(s)$
GSSata01	2.61	**13.37**	12.98	10	0	75.22	16.18	0	0	52.78
GSSata02	2.50	**21.73**	21.27	10	0	43.42	23.57	0	0	39.65
GSSata03	2.62	**23.36**	23.00	7	0	120.67	24.54	0	0	135.05
GSSata04	2.75	**18.25**	17.68	10	0	71.75	19.00	1	0	51.69
GSSata05	2.77	**12.96**	13.51	0	0	73.36	15.59	0	0	
GSSata06	2.21	**17.71**	17.42	10	0	121.93	19.40	0	0	67.42
GSSata07	2.15	**17.42**	16.82	10	0	16.85	18.12	1	0	11.14
GSSata08	2.42	**17.63**	16.21	10	0	108.28	18.60	2	0	6.95
GSSata09	2.83	**18.76**	18.24	8	0	169.75	19.34	1	0	43.73
GSSata10	2.67	**22.04**	20.89	10	0	25.37	22.36	2	0	16.25
GSSata11	3.48	**7.67**	6.67	10	0	164.71	12.29	0	0	52.79
GSSata12	3.31	**0.42**	0	10	10	87.88	1.94	0	0	60.91
GSSata13	3.53	**5.59**	5.19	8	0	179.45	7.48	0	0	135.35
GSSata14	3.04	**5.37**	4.65	9	0	121.57	8.78	0	0	37.78
GSSata15	3.62	**6.06**	5.79	3	0	197.44	9.83	0	0	135.30
GSSata16	3.10	**5.96**	4.55	10	0	219.17	7.10	0	0	46.01
GSSata17	3.57	**6.20**	6.26	4	0	142.26	7.55	0	0	137.76
GSSata18	3.29	**12.14**	12.65	1	0	152.17	15.92	0	0	126.23
GSSata19	3.37	**1.93**	0.78	10	1	194.43	3.61	0	0	94.89
GSSata20	3.17	**4.35**	2.75	10	0	154.62	6.46	0	0	34.54
GSSata21	3.96	**16.61**	17.40	1	0	202.76	23.86	0	0	69.79
GSSata22	3.83	**15.77**	14.01	10	0	175.62	16.63	0	0	31.96
GSSata23	4.25	**13.25**	12.93	7	0	178.22	18.19	0	0	109.94
GSSata24	4.84	**14.63**	13.23	10	0	188.63	18.03	0	0	102.90
GSSata25	3.91	**10.38**	10.55	5	0	141.20	14.86	0	0	106.54
GSSata26	4.19	**17.43**	17.75	4	0	129.93	22.44	0	0	136.58
GSSata27	3.85	**7.09**	7.36	3	0	228.85	13.00	0	0	223.37
GSSata28	3.76	**7.79**	7.16	9	0	197.80	12.35	0	0	130.28
GSSata29	4.45	**13.28**	12.43	9	0	192.17	18.72	0	0	72.59
GSSata30	4.38	**12.03**	12.19	4	0	196.89	17.41	0	0	159.20
GSSata31	1.55	0.01	0.00	10	10	0.28	**0**	10	10	0.28
GSSata32	1.38	0.02	0.00	10	10	1.14	**0**	10	10	1.14
GSSata33	3.18	0.02	0.00	10	10	2.51	**0.01**	9	9	2.51
GSSata34	1.57	0.03	0.00	10	10	0.39	**0**	10	10	0.39
GSSata35	4.79	**0.82**	0.12	10	0	42.88	1.34	0	0	20.61
GSSata36	4.04	0.07	0.00	10	10	2.71	**0**	10	10	2.53
GSSata37	1.10	0.04	0.00	10	10	0.59	**0**	10	10	0.59
GSSata38	1.41	0.04	0.00	10	10	0.39	**0**	10	10	0.39
GSSata39	3.79	7.94	2.08	10	0	138.38	**3.52**	10	0	0.36
GSSata40	5.68	**0.02**	0.00	10	10	12.56	0.83	1	1	12.56
GSSata41	6.34	**0.05**	0.00	10	10	73.69	2.52	0	0	73.69
GSSata42	2.62	**0.44**	0.00	10	10	17.71	6.05	0	0	12.73
GSSata43	3.56	**0.03**	0.00	10	10	146.34	3.81	0	0	146.34
GSSata44	6.62	**0.06**	0.98	3	2	209.59	5.53	0	0	226.63
GSSata45	6.17	**1.85**	1.85	8	0	191.80	7.54	0	0	172.61
GSSata46	5.03	**0.04**	0.00	10	10	9.48	0.57	2	2	9.48
GSSata47	6.18	**1.54**	1.33	7	0	270.57	6.83	0	0	215.73
GSSata48	6.08	**4.59**	5.08	1	0	151.29	8.11	0	0	161.07
GSSata49	5.33	**0.26**	0.00	10	10	53.15	4.71	0	0	46.01
GSSata50	5.75	**3.79**	6.51	0	0	234.01	11.02	0	0	
Average	3.65	**7.78**	7.41	7.82	2.86	116.64	10.31	1.78	1.44	

6.2.7 Mixed Shop

In Table 6.15 we compare the tabu search of Liu and Ong [49] for the MSS to ITS. Since the instances were not available, we have generated them. The authors generated 20 replications and executed the tabu search once, and we performed the same experiment with ITS. The results presented by Liu and Ong [49] use the deviation from the lower bound $\max\{LB_{jb}, \max_{i\in[m]} \sum_{j\in[n]} o_{ij}\}$, which is weaker than LB_{qr}, but in all MSS instances the two lower bounds are equal.

Table 6.14: IGA results for the large instances of Nasiri & Kianfar (2011)

Instance	$ttb(s)$	Instance	$ttb(s)$	Instance	$ttb(s)$
GSSata51	0.71	GSSata61	1.52	GSSata71	19.15
GSSata52	1.33	GSSata62	4.00	GSSata72	16.99
GSSata53	0.73	GSSata63	2.02	GSSata73	27.69
GSSata54	0.65	GSSata64	1.17	GSSata74	23.71
GSSata55	1.23	GSSata65	2.23	GSSata75	28.11
GSSata56	0.39	GSSata66	1.48	GSSata76	20.12
GSSata57	1.26	GSSata67	1.54	GSSata77	13.37
GSSata58	0.48	GSSata68	2.24	GSSata78	22.30
GSSata59	0.58	GSSata69	0.77	GSSata79	14.92
GSSata60	0.54	GSSata70	1.64	GSSata80	21.25
Average $ttb(s)$			7.80		

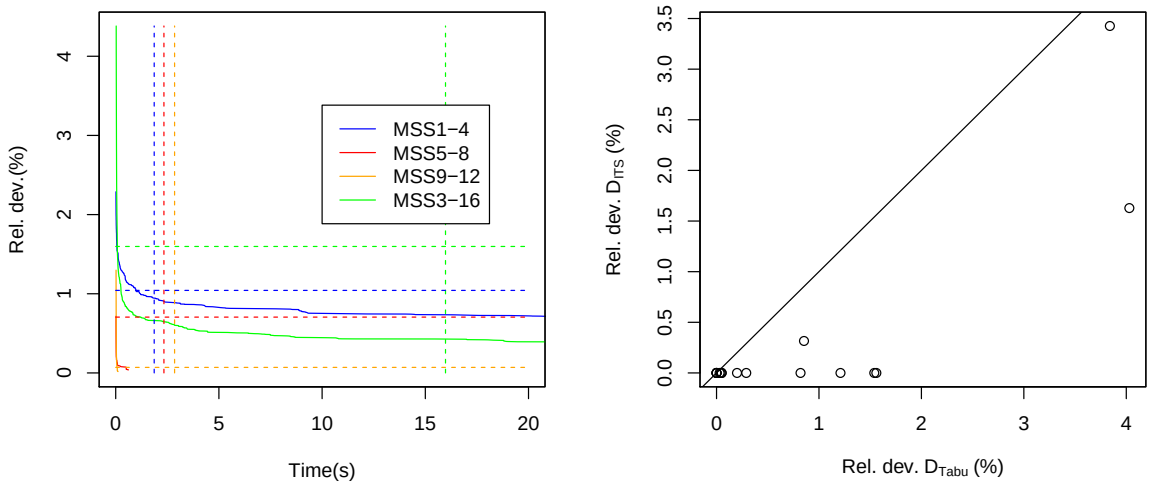
Table 6.15: Comparing ITSA solver with the Tabu Search from Liu & Ong (2004) for MSS

Instance	T_{TS}	D_{TS}	D_{ITS}^{full}	bet^{full}	opt^{full}	$ttb(s)$	D_{ITS}^{rel}	bet^{rel}	opt^{rel}	$\bar{t}(s)$
MSS1	2.11	3.84	2.46	15	6	56.87	**3.43**	10	5	1.94
MSS2	1.08	0.29	0.00	20	20	0.01	**0.00**	20	20	0.01
MSS3	1.42	0.04	0.00	20	20	0.01	**0.00**	20	20	0.01
MSS4	2.85	**0.00**	0.00	20	20	0.01	**0.00**	20	20	0.01
MSS5	1.44	1.56	0.00	20	20	0.09	**0.00**	20	20	0.04
MSS6	1.38	1.21	0.00	20	20	0.01	**0.00**	20	20	0.01
MSS7	2.47	0.05	0.00	20	20	0.01	**0.00**	20	20	0.01
MSS8	4.05	**0.00**	0.00	20	20	0.01	**0.00**	20	20	0.01
MSS9	1.04	0.20	0.00	20	20	0.04	**0.00**	20	20	0.03
MSS10	2.46	0.05	0.00	20	20	0.02	**0.00**	20	20	0.02
MSS11	3.39	0.03	0.00	20	20	0.02	**0.00**	20	20	0.02
MSS12	4.52	**0.00**	0.00	20	20	0.02	**0.00**	20	20	0.02
MSS13	18.77	4.03	1.23	19	11	50.75	**1.63**	15	10	27.52
MSS14	4.71	1.54	0.00	20	20	0.24	**0.00**	20	20	0.05
MSS15	16.43	0.82	0.00	20	20	0.04	**0.00**	20	20	0.03
MSS16	24.01	**0.00**	0.00	20	20	0.04	**0.00**	20	20	0.04
Average	5.76	0.85	0.23	19.62	18.56	6.76	**0.32**	19.06	18.44	1.86

We can see that ITS consistently produces better results, with an overall average relative deviation of 0.32%, while the tabu search has an overall average of 0.85%. ITS was able to achieve the average quality of the tabu search in most instances in $0.05s$ or under for most instances. In instance MSS1 ITS was not able to reach the average deviation of the tabu search in 5 replications of the 20 for that instance, while for MSS13 it was not able in one replication. It is worth noticing that ITS was able to achieve an optimal solution given by the lower bound in all replications of all instances except MSS1 and MSS13. For MSS1 it reached LB_{qr} in 5 of the 20 replications, and for instance MSS13 in 10.

With additional time, under 300 seconds, ITS was able to improve the average deviation to 0.23%, and found one more optimal solution for both MSS1 and MSS13.

In Figure 6.7 we see the time evolution of the solution quality of ITS and the scatter plot comparing ITS with the tabu search. The scatter plot reinforces the conclusion that the quality



(a) Time evolution of the quality of the solution.

(b) Scatter plot comparison.

Figure 6.7: Graphs for evaluating the ITS and the tabu search on the MSS instances.

of the solutions produced by ITS is better than the tabu search. The graph of the deviation value by time shows only the first 20 seconds, and most of the evolution of the quality of the solution is achieved in the first second. In Table 6.15, on the column $ttb(s)$ we can see that for all instances except MSS1 and MSS13 reaches the best value before 0.1 seconds, so the improvement seen in the graph is due to these instances, for which ITS still improves the solution until later than 50 seconds.

6.2.8 Open Shop

For completeness, we studied in an additional experiment the performance of our solver on the OSS instances proposed by Taillard [81], Brucker et al. [10], and Guéret and Prins [30]. Since OSS we have strong best known values we use them instead of LB_{qr} to compute D for OSS, and report the deviation from the best known values instead of the lower bound. We compare ITS with the state-of-the-art algorithm for OSS, the Particle Swarm Optimization with beam search of Sha and Hsu [77] (PSO).

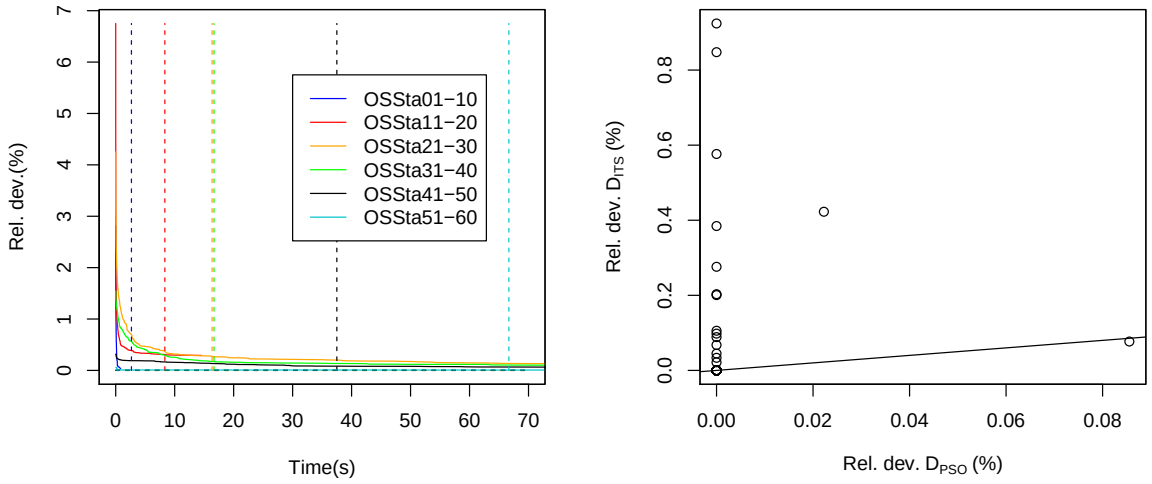
6.2.8.1 Instances by Taillard [81]

In Table 6.16 we compare ITS to PSO on the instances by Taillard [81]. The PSO, with the exception of two instances (OSSta49 and OSSta58), always finds the best known value, in an average time of 24.69 s. ITS produces in the same time an average relative deviation of 0.07% deviation from the lower bound. In 45 of the 60 instances ITS was able to reach the same quality

of the PSO, and in instance OSSta58 it produces a solution slightly better than PSO.

In 300 s the average relative deviation of ITS drops to 0.01%. With this time ITS was able to reach the optimal solution in at least 3 executions for all instances. Except for instances OSSta26, OSSta27, OSSta31, OSSta45, OSSta49 and OSSta58 ITS was able to reach the best known value in the 10 executions. For those instances it was able to reach the optimal value in 5, 9, 7, 7, 7 and 3 times respectively. For the executions where ITS reached the quality of the solution of PSO the average time to do so was 16.51 s which is faster than the time of 24.69 s for PSO.

In Figure 6.8 we see the graphics comparing PSO and ITS. On the left Figure 6.8a shows the evolution with time of the quality of the solutions produced by ITS. We can see that the solution converge fast to the best known value, but also show the slight advantage of the solution quality produced by PSO. On the right, in Figure 6.8b, we see the scatter plot, and it also shows the better solutions are produced by PSO, but it is worth noticing the low values for the deviations, and that a big amount of the points are collapsed in the origin of the graph.



(a) Time evolution of the quality of the solution.

(b) Scatter plot comparison.

Figure 6.8: Graphs for evaluating the ITS and the PSO search on the OSS instances by Taillard [81].

In summary, both PSO and ITS have a similar performance for this instance set, with an overall solution quality very close to optimal, but in a few instances PSO finds the best solutions more consistently. On the other hand ITS was faster than PSO for some instances.

Table 6.16: Comparing ITS with PSO from Sha and Hsu [77] for OSS in the instances by Taillard [81]

Instance	T_{PSO}	D_{PSO}	D_{ITS}^{full}	bet^{full}	opt^{full}	$ttb(s)$	D_{ITS}^{rel}	bet^{rel}	opt^{rel}	$\bar{t}(s)$
OSSta01	2.67	**0.00**	0.00	10	10	0.01	**0.00**	10	10	0.01
OSSta02	2.67	**0.00**	0.00	10	10	0.27	**0.00**	10	10	0.27
OSSta03	2.67	**0.00**	0.00	10	10	0.15	**0.00**	10	10	0.15
OSSta04	2.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
OSSta05	2.67	**0.00**	0.00	10	10	0.13	**0.00**	10	10	0.13
OSSta06	2.67	**0.00**	0.00	10	10	0.03	**0.00**	10	10	0.03
OSSta07	2.67	**0.00**	0.00	10	10	0.08	**0.00**	10	10	0.08
OSSta08	2.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
OSSta09	2.67	**0.00**	0.00	10	10	0.06	**0.00**	10	10	0.06
OSSta10	2.67	**0.00**	0.00	10	10	0.07	**0.00**	10	10	0.07
OSSta11	8.33	**0.00**	0.00	10	10	0.42	**0.00**	10	10	0.42
OSSta12	8.33	**0.00**	0.00	10	10	0.24	**0.00**	10	10	0.24
OSSta13	8.33	**0.00**	0.00	10	10	0.68	**0.00**	10	10	0.68
OSSta14	8.33	**0.00**	0.00	10	10	1.79	**0.00**	10	10	1.79
OSSta15	8.33	**0.00**	0.00	10	10	5.15	0.28	7	7	5.15
OSSta16	8.33	**0.00**	0.00	10	10	0.02	**0.00**	10	10	0.02
OSSta17	8.33	**0.00**	0.00	10	10	2.33	**0.00**	10	10	2.33
OSSta18	8.33	**0.00**	0.00	10	10	1.12	**0.00**	10	10	1.12
OSSta19	8.33	**0.00**	0.00	10	10	0.26	**0.00**	10	10	0.26
OSSta20	8.33	**0.00**	0.00	10	10	0.75	**0.00**	10	10	0.75
OSSta21	16.33	**0.00**	0.00	10	10	2.11	**0.00**	10	10	2.11
OSSta22	16.33	**0.00**	0.00	10	10	16.36	0.07	7	7	16.36
OSSta23	16.33	**0.00**	0.00	10	10	92.34	0.38	1	1	92.34
OSSta24	16.33	**0.00**	0.00	10	10	5.99	0.02	9	9	5.99
OSSta25	16.33	**0.00**	0.00	10	10	3.28	**0.00**	10	10	3.28
OSSta26	16.33	**0.00**	0.13	5	5	77.15	0.58	1	1	78.62
OSSta27	16.33	**0.00**	0.02	9	9	101.04	0.92	0	0	99.30
OSSta28	16.33	**0.00**	0.00	10	10	1.03	**0.00**	10	10	1.03
OSSta29	16.33	**0.00**	0.00	10	10	0.07	**0.00**	10	10	0.07
OSSta30	16.33	**0.00**	0.00	10	10	0.79	**0.00**	10	10	0.79
OSSta31	16.67	**0.00**	0.13	7	7	156.00	0.85	1	1	151.30
OSSta32	16.67	**0.00**	0.00	10	10	5.21	**0.00**	10	10	5.21
OSSta33	16.67	**0.00**	0.00	10	10	45.79	0.20	4	4	45.79
OSSta34	16.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
OSSta35	16.67	**0.00**	0.00	10	10	34.24	0.20	6	6	34.24
OSSta36	16.67	**0.00**	0.00	10	10	1.23	**0.00**	10	10	1.23
OSSta37	16.67	**0.00**	0.00	10	10	5.08	**0.00**	10	10	5.08
OSSta38	16.67	**0.00**	0.00	10	10	5.77	**0.00**	10	10	5.77
OSSta39	16.67	**0.00**	0.00	10	10	8.86	0.03	9	9	8.86
OSSta40	16.67	**0.00**	0.00	10	10	4.80	**0.00**	10	10	4.80
OSSta41	37.50	**0.00**	0.00	10	10	0.79	**0.00**	10	10	0.79
OSSta42	37.50	**0.00**	0.00	10	10	65.25	0.10	3	3	65.25
OSSta43	37.50	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
OSSta44	37.50	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
OSSta45	37.50	**0.00**	0.03	7	7	87.10	0.11	0	0	124.39
OSSta46	37.50	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
OSSta47	37.50	**0.00**	0.00	10	10	42.83	0.04	8	8	42.83
OSSta48	37.50	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
OSSta49	37.50	**0.02**	0.14	7	7	133.32	0.42	1	1	132.43
OSSta50	37.50	**0.00**	0.00	10	10	48.40	0.09	6	6	48.40
OSSta51	66.67	**0.00**	0.00	10	10	0.01	**0.00**	10	10	0.01
OSSta52	66.67	**0.00**	0.00	10	10	0.77	**0.00**	10	10	0.77
OSSta53	66.67	**0.00**	0.00	10	10	0.01	**0.00**	10	10	0.01
OSSta54	66.67	**0.00**	0.00	10	10	0.01	**0.00**	10	10	0.01
OSSta55	66.67	**0.00**	0.00	10	10	0.01	**0.00**	10	10	0.01
OSSta56	66.67	**0.00**	0.00	10	10	0.01	**0.00**	10	10	0.01
OSSta57	66.67	**0.00**	0.00	10	10	0.01	**0.00**	10	10	0.01
OSSta58	66.67	0.09	0.06	10	3	38.99	**0.08**	10	1	0.01
OSSta59	66.67	**0.00**	0.00	10	10	0.01	**0.00**	10	10	0.01
OSSta60	66.67	**0.00**	0.00	10	10	0.01	**0.00**	10	10	0.01
Average	24.69	**0.00**	0.01	9.75	9.63	16.64	0.07	8.55	8.40	16.51

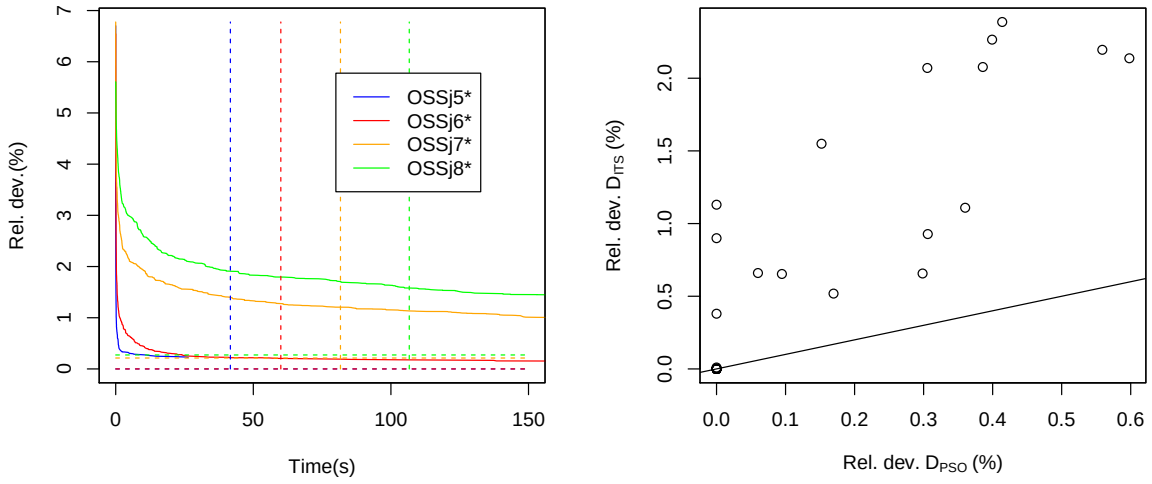
6.2.8.2 *Instances by Brucker et al. [10]*

In Table 6.17 we compare PSO to ITS on the OSS instances introduced by Brucker et al. [10]. The average relative deviation for PSO is 0.12%, for ITS 0.62%. The relative deviation of ITS is equal to that of PSO in 18 instances, mostly the smaller ones. In this case more time ITS does improve significantly: in 300 s the average relative deviation is 0.45%. In 7 of the instances ITS never reaches the best known value, in 9 it reaches in some executions, but not in all, and finally in 19 it was able to eventually reach the best known value in all executions. The average time to the reach best solution ITS finds is 68.58s which is similar to the 75.52s.

Table 6.17: Comparing ITS with PSO from Sha and Hsu [77] for OSS in the instances by Brucker et al. [10]

Instance	T_{PSO}	D_{PSO}	D_{ITS}^{full}	bet^{full}	opt^{full}	$ttb(s)$	D_{ITS}^{rel}	bet^{rel}	opt^{rel}	$\bar{t}(s)$
j5-per0-0	41.67	**0.00**	0.00	10	10	3.28	**0.00**	10	10	3.28
j5-per0-1	41.67	**0.00**	0.00	10	10	0.04	**0.00**	10	10	0.04
j5-per0-2	41.67	**0.00**	0.00	10	10	1.20	**0.00**	10	10	1.20
j5-per10-0	41.67	**0.00**	0.00	10	10	0.22	**0.00**	10	10	0.22
j5-per10-1	41.67	**0.00**	0.00	10	10	13.36	**0.00**	10	10	13.36
j5-per10-2	41.67	**0.00**	0.00	10	10	0.19	**0.00**	10	10	0.19
j5-per20-0	41.67	**0.00**	0.00	10	10	0.03	**0.00**	10	10	0.03
j5-per20-1	41.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
j5-per20-2	41.67	**0.00**	0.00	10	10	0.65	**0.00**	10	10	0.65
j6-per0-0	60.00	**0.00**	0.27	1	1	185.95	0.90	0	0	137.32
j6-per0-1	60.00	**0.00**	0.00	10	10	1.66	**0.00**	10	10	1.66
j6-per0-2	60.00	**0.00**	0.00	10	10	11.23	**0.00**	10	10	11.23
j6-per10-0	60.00	**0.00**	0.00	10	10	8.84	**0.00**	10	10	8.84
j6-per10-1	60.00	**0.00**	0.00	10	10	6.99	**0.00**	10	10	6.99
j6-per10-2	60.00	**0.00**	0.00	10	10	0.96	**0.00**	10	10	0.96
j6-per20-0	60.00	**0.00**	0.00	10	10	26.20	0.01	9	9	26.20
j6-per20-1	60.00	**0.00**	0.00	10	10	0.14	**0.00**	10	10	0.14
j6-per20-2	60.00	**0.00**	0.00	10	10	3.72	**0.00**	10	10	3.72
j7-per0-0	81.67	**0.31**	1.95	0	0	69.51	2.07	0	0	
j7-per0-1	81.67	**0.36**	0.66	3	1	144.68	1.11	1	0	141.67
j7-per0-2	81.67	**0.09**	0.42	2	2	93.79	0.65	1	1	75.08
j7-per10-0	81.67	**0.31**	0.50	3	2	161.81	0.93	1	0	172.86
j7-per10-1	81.67	**0.00**	0.63	1	1	171.10	1.13	0	0	222.12
j7-per10-2	81.67	**0.39**	1.65	0	0	148.62	2.08	0	0	
j7-per20-0	81.67	**0.00**	0.00	10	10	0.35	**0.00**	10	10	0.35
j7-per20-1	81.67	**0.30**	0.21	9	0	145.15	0.66	5	0	92.58
j7-per20-2	81.67	**0.17**	0.28	3	2	118.70	0.52	1	1	171.02
j8-per0-1	106.67	**0.41**	2.00	0	0	151.19	2.39	0	0	
j8-per0-2	106.67	**0.15**	1.28	1	0	171.19	1.55	1	0	91.08
j8-per10-0	106.67	**0.60**	1.59	0	0	183.55	2.14	0	0	
j8-per10-1	106.67	**0.56**	1.69	1	0	164.58	2.20	0	0	140.76
j8-per10-2	106.67	**0.40**	1.91	0	0	134.76	2.27	0	0	
j8-per20-0	106.67	**0.06**	0.47	0	0	137.83	0.66	0	0	
j8-per20-1	106.67	**0.00**	0.00	10	10	1.42	**0.00**	10	10	1.42
j8-per20-2	106.67	**0.00**	0.16	1	1	137.53	0.38	0	0	162.37
Average	71.52	**0.12**	0.45	6.14	5.71	68.58	0.62	5.69	5.46	

In Figure 6.9 we see the graphs comparing ITS and PSO. In Figure 6.9a we see the evolution with time of the solution quality of ITS, which also shows ITS difficulty to reach the best known value for the largest instances despite continually improving the produced solution with time. In Figure 6.9b also confirms the better quality of PSO, despite having many points collapsed in the origin.



(a) Time evolution of the quality of the solution.

(b) Scatter plot comparison.

Figure 6.9: Graphs for evaluating the ITS and the PSO search on the OSS instances by Brucker et al. [10].

In summary, on these instances PSO performs better than ITS, and ITS does not scale well for larger instances.

6.2.8.3 *Instances by Guéret and Prins [30]*

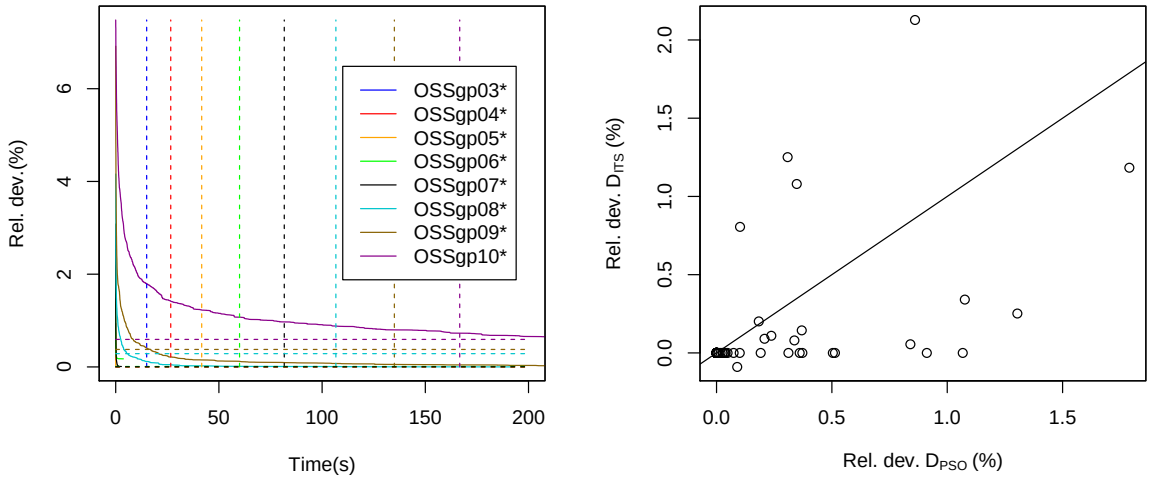
In Figure 6.18 we show the results for ITS and PSO in the instances by Guéret and Prins [30]. ITS achieved an overall average relative deviation of 0.10% from the best known value while PSO obtained 0.16%. The average relative deviation of ITS is better or equal than that of PSO in 75 of the 80 instances. To reach the same solution quality as PSO, ITS takes in average 10.38 s, which is about a factor of 7 faster than PSO. In 300 s the average relative deviation of ITS improves to 0.07% with an average time-to-best of 23.40 s. ITS was able to find a new best solution for instance OSSgp09-03 (of value 1115), and with additional time four also found new best solutions for OSSgp09-07, OSSgp09-08, and OSSgp10-04 (of values 1090, 1105, and 1078, respectively).

In Figure 6.10 we see the usual graphs comparing ITS and PSO. The plot of the solution quality evolution with time in Figure 6.10a shows the faster convergence of ITS. The exception are the instances OSSgp10*, for which PSO produces a better solution, but soon after the time limit for those instances ITS produces the same quality of solutions as PSO. The scatter plot on the right, Figure 6.10b shows that most solutions of ITS are better than PSO.

In summary, ITS performs better on this instance set, producing better solutions many times faster than PSO.

Table 6.18: Comparing ITS with PSO from Sha and Hsu [77] for OSS in the instances by Guéret and Prins [30]

Instance	T_{PSO}	D_{PSO}	D_{ITS}^{full}	bet^{full}	opt^{full}	$ttb(s)$	D_{ITS}^{rel}	bet^{rel}	opt^{rel}	$\bar{t}(s)$
gp03-01	15.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp03-02	15.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp03-03	15.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp03-04	15.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp03-05	15.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp03-06	15.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp03-07	15.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp03-08	15.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp03-09	15.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp03-10	15.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp04-01	26.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp04-02	26.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp04-03	26.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp04-04	26.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp04-05	26.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp04-06	26.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp04-07	26.67	**0.00**	0.00	10	10	0.05	**0.00**	10	10	0.05
gp04-08	26.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp04-09	26.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp04-10	26.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp05-01	41.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp05-02	41.67	**0.00**	0.00	10	10	0.01	**0.00**	10	10	0.01
gp05-03	41.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp05-04	41.67	**0.00**	0.00	10	10	0.71	**0.00**	10	10	0.71
gp05-05	41.67	**0.00**	0.00	10	10	0.18	**0.00**	10	10	0.18
gp05-06	41.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp05-07	41.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp05-08	41.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp05-09	41.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp05-10	41.67	**0.00**	0.00	10	10	0.01	**0.00**	10	10	0.01
gp06-01	60.00	**0.00**	0.00	10	10	0.55	**0.00**	10	10	0.55
gp06-02	60.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp06-03	60.00	0.05	0.00	10	10	0.00	**0.00**	10	10	0.00
gp06-04	60.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp06-05	60.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp06-06	60.00	**0.00**	0.00	10	10	0.02	**0.00**	10	10	0.02
gp06-07	60.00	**0.00**	0.00	10	10	0.15	**0.00**	10	10	0.15
gp06-08	60.00	0.04	0.00	10	10	1.67	**0.00**	10	10	1.67
gp06-09	60.00	0.01	0.00	10	10	0.00	**0.00**	10	10	0.00
gp06-10	60.00	**0.00**	0.00	10	10	0.45	**0.00**	10	10	0.45
gp07-01	81.67	0.03	0.00	10	10	0.85	**0.00**	10	10	0.85
gp07-02	81.67	**0.00**	0.00	10	10	0.09	**0.00**	10	10	0.09
gp07-03	81.67	**0.00**	0.00	10	10	0.50	**0.00**	10	10	0.50
gp07-04	81.67	**0.00**	0.00	10	10	0.58	**0.00**	10	10	0.58
gp07-05	81.67	**0.00**	0.00	10	10	0.04	**0.00**	10	10	0.04
gp07-06	81.67	0.01	0.00	10	10	0.64	**0.00**	10	10	0.64
gp07-07	81.67	**0.00**	0.00	10	10	0.07	**0.00**	10	10	0.07
gp07-08	81.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp07-09	81.67	**0.00**	0.00	10	10	0.54	**0.00**	10	10	0.54
gp07-10	81.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp08-01	106.67	0.91	0.00	10	10	24.53	**0.00**	10	10	0.00
gp08-02	106.67	0.04	0.00	10	10	9.21	**0.00**	10	10	9.21
gp08-03	106.67	0.36	0.00	10	10	15.72	**0.00**	10	10	4.05
gp08-04	106.67	0.02	0.00	10	10	6.62	**0.00**	10	10	6.62
gp08-05	106.67	0.07	0.00	10	10	0.00	**0.00**	10	10	0.00
gp08-06	106.67	1.07	0.00	10	10	23.67	**0.00**	10	10	12.85
gp08-07	106.67	0.34	0.00	10	10	93.40	**0.08**	10	6	12.58
gp08-08	106.67	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp08-09	106.67	0.03	0.00	10	10	35.40	**0.00**	10	10	35.40
gp08-10	106.67	0.03	0.00	10	10	14.80	**0.00**	10	10	14.80
gp09-01	135.00	0.37	0.00	10	10	15.44	**0.00**	10	10	13.59
gp09-02	135.00	0.37	0.14	10	1	24.87	**0.14**	10	1	7.11
gp09-03	135.00	0.09	-0.09	10	10	22.81	**-0.09**	10	10	7.66
gp09-04	135.00	0.51	0.00	10	10	18.27	**0.00**	10	10	7.53
gp09-05	135.00	**0.00**	0.00	10	10	0.00	**0.00**	10	10	0.00
gp09-06	135.00	0.10	0.00	10	10	17.65	**0.00**	10	10	8.80
gp09-07	135.00	0.50	-0.08	10	10	132.70	**0.00**	9	9	28.04
gp09-08	135.00	0.21	-0.01	10	10	74.13	**0.09**	9	1	42.21
gp09-09	135.00	0.31	0.00	10	10	14.77	**0.00**	10	10	13.51
gp09-10	135.00	1.30	0.00	10	10	167.18	**0.25**	10	2	15.54
gp10-01	166.67	**0.35**	1.06	3	2	110.35	1.08	3	2	32.98
gp10-02	166.67	0.19	0.00	10	10	21.23	**0.00**	10	10	20.48
gp10-03	166.67	**0.86**	1.87	4	0	178.96	2.13	4	0	137.04
gp10-04	166.67	0.84	-0.18	10	10	137.48	**0.06**	9	1	26.84
gp10-05	166.67	1.79	1.08	9	0	147.97	**1.18**	8	0	55.15
gp10-06	166.67	**0.31**	0.90	6	4	128.88	1.25	3	3	117.46
gp10-07	166.67	**0.10**	0.56	4	0	124.30	0.81	3	0	98.03
gp10-08	166.67	0.24	0.09	10	3	96.57	**0.11**	10	3	35.04
gp10-09	166.67	1.08	0.19	10	2	151.60	**0.34**	10	1	11.64
gp10-10	166.67	**0.18**	0.10	9	9	56.36	0.20	8	8	49.14
Average	79.17	0.16	0.07	9.69	8.53	23.40	**0.10**	9.57	8.51	10.38



(a) Time evolution of the quality of the solution.

(b) Scatter plot comparison.

Figure 6.10: Graphs for evaluating the ITS and the PSO search on the OSS instances by Guéret and Prins [30].

6.3 Tabu Search for Job Shop Scheduling with Parallel Machines

In this section we evaluate the results of our proposed tabu search for JSS-PM.

6.3.1 Lower Bound for Job Shop Scheduling with Parallel Machines

To evaluate the quality of the solutions we use the relative deviation D from the lower bound

$$L = \max\{L_1, L_2\}$$

which is the maximum of the job lower bound

$$L_1 = \max_{j\in[n]} \sum_{i\in[m]} p_{ij}$$

and the machine lower bound

$$L_2 = \max_{i\in[m]} \Big(\min_{j\in[n]} Q(i,j) + \Big\lceil \sum_{j\in[n]} p_{ij}/k(i) \Big\rceil + \min_{j\in[n]} R(i,j) \Big).$$

Here $Q(i,j)$ is the sum of the processing times of the operations preceding operation o_{ij} in job j, and $R(i,j)$ is the sum of the processing times operations succeeding operation o_{ij} in the same job.

Table 6.19: Parameters of the tabu search, initial ranges, and calibrated values.

Parameter	Range	Value
Tabu tenure T	$[4,15]$	8
Max. no. of tabu search iterations w/o improvement $I_{\max}$	$[600,5000]$	2905
Reduction of the no. of iterations after backtracking Δ	$[0,500]$	346
Size of the elite set L	$[0,8]$	6

6.3.2 Parameter setting

Like in the algorithm for ITS we calibrated the tabu for JSS-PM with the R package irace [52].

The parameters we calibrated are $I_{\max}$, Δ, L and T. We constructed 20 random instances for calibration. They were selected with a random number of jobs $n \in U[5,50]$, stages $m \in U[5,30]$, and number of parallel machines per stage $k_i \in U[1,5]$, and random processing times $p_{ij} \in U[1,99]$.

Differently from the ITS for PSS this algorithm stopping criterion is not the time, but it is derived from the parameters Δ, L and T. Since they are directly connected with the budget of time for the execution of the algorithm, to calibrate them using irace we calibrate the algorithm focusing on its anytime characteristics using the hypervolume measure as proposed by López-Ibáñez and Stützle [51]. Let $D(t)$ be the relative deviation of the quality of the best solution produced by the proposed tabu search at t seconds, the objective value to be minimized used during the calibration is $\int_{0s}^{30s} D(t)dt$. Here we ignore the time to construct the initial solution and take $D(0)$ to be the relative deviation of the initial solution value from the lower bound.

The initial ranges for the parameters to be calibrated were chosen based on preliminary experiments. These ranges, and the result of the calibration are shown in Table 6.19.

6.3.3 Experiment 1: Effectiveness of the learning component

In this section we assess the effectiveness of the learning component. We have first run experiments without the trimming phase, to determine the typical range of Kendall's W in the instances. We have observed convergence, with an average of $W = 0.47$ after the initial 30 iterations, an average value of 0.65, and a final value of 0.70. Based on these values, we have fixed $W_0 = 0.6$. This allows most of the instances to enter intro the trimming phase with a reasonably high probability of selecting one of the best moves. For a typical neighborhood size of 20, for example, the probability of a random permutation having $W \geq 0.6$ is only about 0.14, and for $\beta \geq 0.4$ the probability of selecting the best move is more than 0.5.

We next have studied the dependence on the trimming parameter β. For the limit of $\beta \to 0$, i.e. selecting always the best predicted solution, we found an average relative deviation of 4.52 % from the lower bound. This is only slightly worse than using the full neighborhood

Table 6.20: Comparison of plain TS and TS with learning on the first two instance sets.

	Plain TS		Learning TS	
Instance set	$D(\%)$	$t(s)$	$D(\%)$	$t(s)$
1	0.40	4.24	0.66	2.74
2	1.45	0.26	1.52	0.14

($\beta = 1.0$) with an average relative deviation of 4.18 %. We have finally selected $\beta = 0.2$, to make the selection more robust. For this value we expect for a typical neighborhood size of 20 a probability of about 0.3 of selecting the best move for a random permutation. The empirical probability is with 0.84 much higher, since observed permutations are not random and we often have several best moves.

Table 6.20 compares the tabu search with and without the learning component on the first two instance sets. We can see that in both instance sets we obtain a slightly worse solution quality for a speedup in solution time of about 2.

6.3.4 Experiment 2: Comparison to results from the literature

Next we compare the results of the proposed algorithm (T^l_{zr2}) with the literature including our previous work on JSP-PM (T_{zr1}) [89]. In Table 6.21 we see the comparison of T^l_{zr2} with algorithm GH for JSP-PM proposed by Gholami and Sotskov [24]. T^l_{zr2} produces better solutions than *GH* in 17 instances and the same solution in the 5 remaining instances. The average relative deviation of T^l_{zr2} is 1.52% compared to 8.21% for *GH*. We were able to reach the lower bound in 14 of the 22 instances. All instances have a relative deviation of less than 2.5 % from the lower bound, except for GHla01 and GHmt10. Gholami and Sotskov [24] do not report the execution times, but observe that all tests were concluded in under a second, which is also true for T^l_{zr2}.

We can also see that T^l_{zr2} improves the solution quality of T_{zr1} of 15.22% in average by a factor of ten. Most of the improvement comes from the instances GHla15 to GHla20, which have 10 jobs and 10 stages and in average 3 parallel machines per stage. Therefore the time a machine is occupied in a schedule will be in average 3 times less than for a job. This results in critical paths that contains mostly conjunctive arcs, which can not be changed to generate neighbours. For these instance most of the times T_{zr1} is not be able to improve the initial solutions due to not having valid neighbours. The regular neighbourhood T^l_{zr2} is already bigger, and T^l_{zr2} is able to use the extended neighbouhood to avoid getting stuck.

Rossi and Boschi [74] have performed their experiments on a PC with an AMD Athlon 2800. Based on the scores from the PassMark benchmark, we conservatively estimate that our computing environment is 5 times faster, and we adjusted the reported times of Rossi and Boschi [74] accordingly.

Table 6.21: Comparing T^l_{zr2} with the algorithm of Gholami and Sotskov [24].

		GH	T_{zr1}		T^l_{zr2}				GH	T_{zr1}		T^l_{zr2}	
Inst	LB	$D(\%)$	$D(\%)$	$t(s)$	$D(\%)$	$t(s)$	Inst	LB	$D(\%)$	$D(\%)$	$t(s)$	$D(\%)$	$t(s)$
la01	530	23.02	5.28	0.45	**5.28**	0.06	la12	701	**0.00**	3.42	0.31	**0.00**	0.34
la02	655	1.83	0.00	0.07	**0.00**	0.08	la13	384	23.44	38.80	0.02	**0.52**	0.36
la03	494	5.87	0.00	0.34	**0.00**	0.08	la14	566	22.26	0.00	0.42	**0.00**	0.41
la04	369	12.74	12.74	0.00	**0.00**	0.00	la15	1142	5.25	0.00	0.32	**0.00**	0.27
la05	463	**0.00**	0.00	0.11	**0.00**	0.01	la16	717	14.23	29.71	0.00	**1.53**	0.01
la06	815	1.60	0.00	0.41	**0.00**	0.05	la17	730	10.96	41.51	0.08	**1.23**	0.11
la07	694	**2.02**	6.05	0.15	**2.02**	0.09	la18	663	**0.00**	55.05	0.00	**0.00**	0.00
la08	863	**0.00**	0.00	0.13	**0.00**	0.04	la19	685	4.96	56.20	0.00	**1.46**	0.10
la09	769	**0.00**	0.00	0.12	**0.00**	0.01	la20	756	**0.00**	55.29	0.00	**0.00**	0.00
la10	443	0.23	9.48	0.01	**0.00**	0.00	mt10	655	30.38	18.93	0.27	**18.93**	0.18
la11	532	2.07	0.00	0.38	**0.00**	0.47	mt20	613	19.74	2.45	0.89	**2.45**	0.41
Averages									8.21	15.22	0.20	**1.52**	0.14

Table 6.22: Comparing T^l_{zr2} with the HGA algorithm of Rossi and Boschi [74].

		$k=2$						$k=3$					
		HGA		T_{zr1}		T^l_{zr2}		*HGA*		T_{zr1}		T^l_{zr2}	
Inst	LB	$D(\%)$	$t(s)$	$D(\%)$	$t(s)$	$D(\%)$	$t(s)$	$D(\%)$	$t(s)$	$D(\%)$	$t(s)$	$D(\%)$	$t(s)$
la01	666	**0.00**	36.60	0.00	0.41	**0.00**	0.17	1.65	72.20	0.45	0.78	**0.00**	1.53
la02	655	5.04	44.20	1.53	0.46	**0.61**	0.98	8.70	72.60	0.46	1.99	**3.51**	2.10
la03	588	6.46	58.00	3.06	0.65	**5.44**	0.95	14.46	49,60	7.14	0.79	**3.40**	4.11
la04	567	7.76	62.40	5.47	0.60	**1.94**	0.89	10.93	120.60	5.29	1.01	**3.35**	1.73
la05	593	**0.00**	0.00	0.00	0.28	**0.00**	0.37	**0.00**	38,60	0.00	0.39	**0.00**	1.16
la06	926	**0.00**	0.00	0.00	0.59	**0.00**	0.25	1.08	84,80	0.00	1.26	**0.00**	0.46
la07	890	0.45	22.00	0.00	1.02	**0.00**	1.29	3.60	240.40	0.00	3.08	**0.00**	2.96
la08	863	**0.00**	2,60	0.00	0.73	**0.00**	1.20	0.93	91.20	0.12	0.79	**0.00**	2.74
la09	951	**0.00**	0.00	0.00	0.66	**0.00**	1.11	0.11	121,80	0.21	1.55	**0.00**	4.21
la10	958	**0.00**	0.00	0.00	0.67	**0.00**	1.53	**0.00**	88.60	0.00	1.77	**0.00**	3.57
la11	1222	**0.00**	0.40	0.00	2.80	**0.00**	3.70	1.39	443,8	0.00	5.03	**0.00**	9.46
la12	1039	**0.00**	0.40	0.00	1.23	**0.00**	2.45	0.96	485,40	0.00	2.06	**0.00**	7.58
la13	1150	**0.00**	0.00	0.00	1.91	**0.00**	2.96	1.13	462,20	0.00	4.57	**0.00**	10.23
la14	1292	**0.00**	0.00	0.00	1.37	**0.00**	0.35	**0.00**	84.40	0.00	3.01	**0.00**	1.11
la15	1207	3.23	72.00	0.00	5.55	**0.00**	2.76	6.30	406.40	1.24	15.59	**1.41**	8.08
Avrg.		1.53	19.90	0.67	1.26	**0.53**	1.40	3.42	190.84	0.99	2.91	**0.78**	4.07
Total Average								2.48	105.34	0.83	2.09	**0.66**	2.74

In Table 6.22 we compare T^l_{zr2} with the hybrid genetic and ant colony (HGA) algorithm of Rossi and Boschi [74] on instance set 1. T^l_{zr2} produced only solutions that are equally good as or better than HGA. On the 15 instances with 2 replications T^l_{zr2} produced 5 better solutions than HGA, and on the 15 instances with 3 replications T^l_{zr2} produced 13 solutions better than HGA. The average relative deviation of the solutions produced by T^l_{zr2} is 0.66%, while for HGA it is 2.48%. T^l_{zr2} is also about 40 times faster than HGA.

T^l_{zr2} has reached the lower bound in 23 of the 30 instances, and the worst relative deviation obtained was 5.44% in instance ROla03. T^l_{zr2} has an average relative deviation of 0.66%, a slight improvement over T_{zr1}, with an average of 0.83%, but is also slightly slower.

7 CONCLUSION

For PFSS we studied the utility of the Bubble Search algorithm. We have analyzed the dependence of Bubble Search on its parameter α and proposed a new adaptive variant of Bubble Search, that scales the probability of accepting a solution of Kendall-tau distance d in accordance with the size of the instance.

Computational experiments we could demonstrate that for flow shop scheduling the adaptive variant outperforms the regular randomized Bubble Search with replacement. The experiments show also that Bubble Search is a promising technique for obtaining results comparable to the best constructive heuristics, especially on small instances.

For PSS we have proposed a general solver, which includes OSS, PSS, MSS, and GSS, and have have compared its performance to the current best algorithms in the literature for all these problem variants. The solver is overall competitive with all tested algorithms, with a clearly better performance for PSS and MSS, and a similar performance for the GSS and OSS, with very few exceptions.

The experiments show that the main components of the solver, which use only the general structural characteristics of PSS, translate well when drastically restricting the structure of the problem.

Our main conclusion is that it is possible to design a single algorithm based on a few selected, effective components for construction, perturbation and improvement of solutions that works well on all these variants of shop scheduling problems. The comparison to five state-of-the-art algorithms in significantly different time scales, also shows that a robust performance for different time limits is achievable.

For JSS-PM the tabu search clearly outperforms the state of the art, in both processing time and solution quality. On the 52 instances of the literature we produced solutions with the same quality as the state of the art in 20 instances, and better in 32, and in no instances we produced a worse solution.

On the instances proposed by Gholami and Sotskov [24] we obtained an average relative deviation of 1.52% from the lower bound, while GH obtained solutions with relative deviation of 8.21%. We reached the lower bound in 14 of the 22 instances. Gholami and Sotskov [24] report execution times of less than one second, and the same is true for our algorithm, with an average execution time of $0.14s$.

For the 30 instances proposed by Rossi and Boschi [74] we obtained an average deviation of 0.66%, in comparison to HGA that obtained 2.48%. On the instances 15 with two machine replications in 10 HGA and T^l_{zr2} produce the same quality of solution, while in 5 T^l_{zr2} produces better solutions. On the 15 with three machine replications T^l_{zr2} was able to improve the solutions on 12, and only in 3 they produced the same quality of solution. The execution time of the tabu search is considerably lower than that of the HGA in most of the instances. In average the tabu search is more than a factor 10 faster for the instances with $k = 2$ replicated machines,

and more than 40 times faster for the instances with $k = 3$ replicated machines. The better scalability in relation to the number of replicated machines of the tabu search is probably due to the strategy of not changing the DG representation with the number of replicated machines, which maintains the search space size for the tabu search independent of the number of replicated machines k.

We were able to show that the proposed tabu search can generate good results in a timely manner combining a representation that ignores the replicated machines with a greedy strategy to schedule such machines. The main contribution of this work is the scheduling strategy for the replicated machines, combined with the new neighbourhood that uses an adapted critical path. The results generated by the proposed tabu search are good, both in time and quality of the generated solutions.

7.1 Future Work

We believe that it is possible to broaden the scope of the ITS solver even more, e.g. to parallel machines, alternative routings, or job shop scheduling. Job shop scheduling in particular has been studied intensively in the literature, but since our heuristic was not designed for it more effort is necessary to include it (in preliminary tests ITS produces solutions about 1.8% above the state of the art).

7.1.1 Partial Shop

To further the study of PSS we think a new set of instances, more representative of the generality of the problem, can be of value. The current instance set has only instances with low order strength, in average of 0.12. We think an alternative set that contains instances with variable order strength and sizes can be useful to guide the further development of heuristics for PSS.

During our research of shop problems a recurrent problem has being improving on solutions with multiple critical paths, since all critical paths must be destroyed in order to improve the quality of the solution. We developed a technique to identify all operations in all critical paths in an efficient manner. This can be used in combination with all the techniques presented here, and we think it is a promising line of research. We are also able to identify which operations are in the edge of a block on any critical path, which can be used to reduce the neighbourhood in the same way it is done on the N5 for JSS, or on the PSS neighbourhood presented in this work.

We performed some preliminary work on the adaptation of the N6 neighbourhood for PSS, which we think can be a promising way of improving the effectiveness of the heuristics. In particular we think that adapting the main ideas proposed by Peng et al. [65] for JSS can generate an effective solver for PSS.

Several of the problems presented in this work use alternative routings for the jobs. These

alternative routes can be represented by a set of partial orders, and depending on the characteristics of the instances, this set may be much smaller than the number of possible alternative routings. We think that an approach that aims to identify which the alternative can be effectively represented as partial orders can generate competitive optimization techniques for these problems.

7.1.2 Parallel Machines

Like in the case of PSS it is harder to improve upon solutions with multiple critical paths in the case of JSS-PM as well. Therefore the technique to identify the operations in all critical paths can be of use in the case with parallel machines as well.

Several models use parallel machines that are not identical. For instance, in the Hybrid JSS, each operation can be processed in one of a set of possible machines, but the processing time of the operation depends on which machine it was assigned. The machine assignment can also be solved implicitly instead of representing it directly in the DG, and we think extending the heuristic parallel machine assignment presented in this work is a promising line of research.

7.1.3 Other Future Work

The learning component presented in here is simple and most likely can be improved. The learning component can be used, without much change, to improve the speed of the other heuristics proposed in this work. We think a more robust learning component can bring a considerable speed up, and we intend to study the usage of the learning component to select which neighbourhood, of a portfolio, should be used dynamically during the search. Using the learning component to trim the neighbourhood may be more important for naturally larger neuighbourhoods such as the the N6 or neighbourhoods which change multiple critical paths.

Finally we think a partial shop with parallel machines is possibly a useful problem and we think it can be interesting to study it.

REFERENCES

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401, 1988.

[2] S. B. Akers Jr. A graphical approach to production scheduling problems. *Oper. Res.*, 4(2): 244–245, 1956.

[3] P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In T. Bartz-Beielstein, M. J. Blesa, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, editors, *HM 2007*, volume 4771 of *LNCS*, pages 108–122, 2007.

[4] E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job shop scheduling. *Management Science*, 44(2):262–275, 1998.

[5] U. Benlic, M. G. Epitropakis, and E. K. Burke. A hybrid breakout local search and reinforcement learning approach to the vertex separator problem. *European Journal of Operational Research*, 261(3):803–818, 2017.

[6] C. Blum. Beam-aco hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Computers & Operations Research*, 32(6):1565–1591, 2005.

[7] C. Blum and M. Sampels. An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms*, 3(3):285–308, 2004.

[8] P. Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001. ISBN 3540415106.

[9] P. Brucker and R. Schlie. Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375, 1990.

[10] P. Brucker, J. Hurink, B. Jurisch, and B. Wöstmann. A branch & bound algorithm for the open-shop problem. *Discrete Applied Mathematics*, 76(1):43–59, 1997.

[11] H. G. Campbell, R. A. Dudek, and M. L. Smith. A heuristic for the n job, m machine sequencing problem. *Management Science 16(10)*, 1970.

[12] J. Carlier. Ordonnancements a contraintes disjonctives. *R.A.I.R.O. Recherche operationelle/Operations Research*, 12(4):333–351, 1978.

[13] J. Carlier. The one-machine sequencing problem. *European Journal of Operational Research*, 11(1):42 – 47, 1982. ISSN 0377-2217.

[14] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management science*, 35(2):164–176, 1989.

[15] I. A. Chaudhry and A. A. Khan. A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23(3):551–591, 2016.

[16] D. G. Dannenbring. An evaluation of flow shop sequencing heuristics. *Management Science July 1977 vol. 23*, 1977.

[17] E. Demirkol, S. Mehta, and R. Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.

[18] U. Dorndorf, E. Pesch, and T. Phan-Huy. Solving the open shop scheduling problem. *Journal of Scheduling*, 4(3):157–174, 2001.

[19] T. U. Eindhoven. Whizzkids '97 contest. http://www.win.tue.nl/whizzkids/1997, 1997. Accessed 2015-02-20.

[20] S. Farahmand, R. Ruiz, and N. Boroojerdian. New high performing heristics for minimizing makespan in permutation flowshops. *Omega, The International Journal of Management Science*, 2009.

[21] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling*, pages 225–251, 1963.

[22] J. M. Framinan, R. Leister, and C. Rajendran. Different initial sequences for the heuristic of nawaz, enscore and ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research, 41(1):121-148*, 2003.

[23] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.*, 1(2):117–129, 1976.

[24] O. Gholami and Y. N. Sotskov. A fast heuristic algorithm for solving parallel-machine job-shop scheduling problems. *The International Journal of Advanced Manufacturing Technology*, 70(1-4):531–546, 2014.

[25] O. Gholami and Y. N. Sotskov. Solving parallel machines job-shop scheduling problems by an adaptive algorithm. *Int. J. Prod. Res.*, 52(13):3888–3904, 2014.

[26] F. Glover. Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). *Discrete Applied Mathematics*, 49(1-3):231–255, 1994.

[27] J. F. Gonçalves and M. G. Resende. An extended akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. *International Transactions in Operational Research*, 21(2):215–246, 2014.

[28] M. A. González, C. R. Vela, and R. Varela. Scatter search with path relinking for the flexible job shop scheduling problem. *European Journal of Operational Research*, 245 (1):35–45, 2015.

[29] C. Guéret and C. Prins. Classical and new heuristics for the open-shop problem: A computational evaluation. *European Journal of Operational Research*, 107(2):306–314, 1998.

[30] C. Guéret and C. Prins. A new lower bound for the open-shop problem. *Annals of Operations Research*, 92:165–183, 1999.

[31] J. Gupta and E. Stafford. A comprehensive review and evaluation of permutation flowshop heuristics. *Eur. J. Oper. Res.*, 169(3):699–711, 2006.

[32] H. H. Hoos and T. Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[33] G. F. Italiano. Transitive closure for dynamic graphs. In *Proc. 24th Ann. Allerton Conf. on Communication, Control and Computing*, pages 29–38, 1986.

[34] M. T. Jensen. Generating robust and flexible job shop schedules using genetic algorithms. *IEEE Transactions on evolutionary computation*, 7(3):275–288, 2003.

[35] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1954.

[36] I. Kacem, S. Hammadi, and P. Borne. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 32(1):1–13, 2002.

[37] P. J. Kalczynski and J. Kamburowski. An improved neh heuristic to minimize makespan in permutation flow shops. *Computer & Operations Research 35 3001-3008*, 2008.

[38] A. Kan. *Machine Scheduling Problems: Classification, complexity and computations*. Springer US, 2012. ISBN 9781461343837. URL https://books.google.com.br/books?id=_JnhBwAAQBAJ. Accessed 2018-03-20.

[39] R. Kan. *Machine Scheduling Problems: Classification, Complexity and Computations*. Martinus Nijhoff, The Hague, The Netherlands, 1976.

[40] T. Kis. Job-shop scheduling with processing alternatives. *European Journal of Operational Research*, 151(2):307–332, 2003.

[41] M. P. Kleeman and G. B. Lamont. Scheduling of flow-shop, job-shop, and combined scheduling problems using moeas with fixed and variable length chromosomes. In *Evolutionary Scheduling*, pages 49–99. Springer, 2007.

[42] C. Koulamas and G. J. Kyparisis. The three-stage assembly flowshop scheduling problem. *Computers & Operations Research*, 28:689–704, 2001. doi: 10.1016/S0305-0548(00)00004-6.

[43] S. Lawrence. Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, 1984.

[44] C.-Y. Lee, T. C. E. Cheng, and B. M. T. Lin. inimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science*, 39(5):616–625, 1993. doi: 10.1287/mnsc.39.5.616.

[45] N. Lesh and M. Mitzenmacher. Bubblesearch: A simple heuristic for improving priority-based greedy algorithms. *Information Processing Letters*, 97(4):161 – 169, 2006. ISSN 0020-0190.

[46] N. Lesh and M. Mitzenmacher. Bubblesearch: A simple heuristic for improving priority-based greedy algorithms. *Information Processing Letters*, 97(4):161–169, 2006.

[47] C.-F. Liaw. A tabu search algorithm for the open shop scheduling problem. *Computers & Operations Research*, 26(2):109–126, 1999.

[48] C.-F. Liaw. A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Operational Research*, 124(1):28–42, 2000.

[49] S. Q. Liu and H. L. Ong. Metaheuristics for the mixed shop scheduling problem. *Asia-Pacific Journal of Operational Research*, 21(01):97–115, 2004.

[50] S. Q. Liu, H. L. Ong, and K. M. Ng. A fast tabu search algorithm for the group shop scheduling problem. *Advances in Engineering Software*, 36(8):533–539, 2005.

[51] M. López-Ibáñez and T. Stützle. Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research*, 235(3):569–582, 2014.

[52] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université libre de Bruxelles, 2011.

[53] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated race for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. doi: 10.1016/j.orp.2016.09.002.

[54] T. Masuda, H. Ishii, and T. Nishida. The mixed shop scheduling problem. *Discrete Applied Mathematics*, 11(2):175 – 186, 1985.

[55] R. Moll, A. Barto, T. Perkins, and R. Sutton. Reinforcement learning and local search: A case study. 1997.

[56] L. Mousin, L. Jourdan, M.-E. K. Marmion, and C. Dhaenens. Feature selection using tabu search with learning memory: learning tabu search. In *International Conference on Learning and Intelligent Optimization*, pages 141–156. Springer, 2016.

[57] J. F. Muth and G. L. Thompson. *Industrial scheduling*. Prentice-Hall, 1963.

[58] M. M. Nasiri. A modified abc algorithm for the stage shop scheduling problem. *Applied Soft Computing*, 28:81–89, 2015.

[59] M. M. Nasiri and F. Kianfar. A ga/ts algorithm for the stage shop scheduling problem. *Computers & Industrial Engineering*, 61(1):161 – 170, 2011. ISSN 0360-8352.

[60] M. M. Nasiri and F. Kianfar. A hybrid scatter search for the partial job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 52(9-12): 1031–1038, 2011. ISSN 0268-3768.

[61] M. Nawaz, E. E. Enscore Jr, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[62] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[63] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *Eur. J. Oper. Res.*, 91(1):160–175, 1996.

[64] Passmark. Passmark cpu benchmarks, 2016. URL https://www.cpubenchmark.net/cpu_list.php. Accessed 2016-10-22.

[65] B. Peng, Z. Lü, and T. Cheng. A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, 53:154–164, 2015.

[66] S. K. Peter Brucker. Complexity results for scheduling problems. http://www.informatik.uni-osnabrueck.de/knust/class, october 2012. Accessed 2016-05-2.

[67] F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212, 2008.

[68] M. Pinedo. *Scheduling Algorithms*. Springer, 2006.

[69] M. Pinedo. *Scheduling*. Springer, 2012.

[70] C. N. Potts and V. A. Strusevich. Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society*, 60(1):S41–S68, 2009.

[71] S. Prestwich. Tuning local search by average-reward reinforcement learning. In *International Conference on Learning and Intelligent Optimization*, pages 192–205. Springer, 2007.

[72] C. Prins. Competitive genetic algorithms for the open-shop scheduling problem. *Mathematical methods of operations research*, 52(3):389–411, 2000.

[73] C. Rajendran and H. Ziegler. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155(2):426 – 438, 2004. ISSN 0377-2217. Financial Risk in Open Economies.

[74] A. Rossi and E. Boschi. A hybrid heuristic to solve the parallel machines job-shop scheduling problem. *Advances in Engineering Software*, 40(2):118–127, 2009.

[75] R. Ruiz and T. Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3): 2033 – 2049, 2007. ISSN 0377-2217.

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

[77] D. Sha and C.-Y. Hsu. A new particle swarm optimization for the open shop scheduling problem. *Computers & Operations Research*, 35(10):3243–3261, 2008.

[78] V. Strusevich. Shop scheduling problems under precedence constraints. *Annals of operations research*, 69:351–377, 1997.

[79] E. Taillard. Some efficient heuristic methods for the flow shop scheduling problem. *European Journal of Operations Research 47 3001-3008*, 1990.

[80] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research*, 47(1):65–74, 1990.

[81] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278 – 285, 1993.

[82] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operations Research 64 278-85*, 1993.

[83] E. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA journal on Computing*, 6(2):108–117, 1994.

[84] E. Taillard. Flow shop instances. http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html, 2013. Accessed 2018-03-20.

[85] P. J. Van Laarhoven, E. H. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations research*, 40(1):113–125, 1992.

[86] S. Voß and A. Witt. Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: A real-world application. *International journal of production economics*, 105(2):445–458, 2007.

[87] C. Zhang, P. Li, Z. Guan, and Y. Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34 (11):3229–3242, 2007.

[88] G. Zhang, L. Gao, and Y. Shi. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4):3563–3573, 2011.

[89] T. Zubaran and M. Ritt. A tabu search for the parallel machines job-shop scheduling problem. 2016.

[90] T. K. Zubaran and M. Ritt. An iterated greedy algorithm for the partial job shop scheduling problem. *VIII ALIO/EURO Workshop on Applied Combinatorial Optimization*, 2014.

[91] T. K. Zubaran and M. Ritt. Online supplement to "An effective heuristic algorithm for the partial shop scheduling problem", 2017. URL http://inf.ufrgs.br/algopt/pssp. Accessed 2018-03-20.