USING ITERATED LOCAL SEARCH FOR SOLVING THE FLOW-SHOP PROBLEM: PARALLELIZATION, PARAMETRIZATION, AND RANDOMIZATION ISSUES

Angel A. Juan[1], Helena R. Lourenço[2], Manuel Mateo[3], Rachel Luo[1], Quim Castella[1]

(1) {ajuanp, rachel.s.luo, quim.castella}@gmail.com – Computer Science Dept., IN3-Open University of Catalonia, Spain
(2) helena.ramalhinho@upf.edu – Dept. of Economics and Business, Universitat Pompeu Fabra, Spain
(3) manel.mateo@upc.edu – Dept. of Management, Universitat Politecnica Catalunya, Spain

ABSTRACT

Iterated Local Search (ILS) is a powerful framework for developing efficient algorithms for the Permutation Flow Shop Problem (PFSP). These algorithms are relatively simple to implement and use very few parameters, which facilitates the associated fine-tuning process. Therefore, they constitute an attractive solution for real-life applications. In this paper, we discuss some parallelization, parametrization, and randomization issues related to ILS-based algorithms for solving the PFSP. In particular, the following research questions are analyzed: (a) is it possible to simplify even more the parameter setting in an ILS framework without affecting performance? (b) how do parallelized versions of these algorithms behave as we simultaneously vary the number of different runs and the computation time? (c) for a parallelized version of these algorithms, is it worthwhile to randomize the initial solution so that different starting points are considered?; and (d) are these algorithms affected by the use of a 'good-quality' pseudo-random number generator? In this paper, we introduce the new ILS-ESP algorithm which is specifically designed to take advantage of parallel computing, allowing us to obtain competitive results in 'real-time' for all tested instances. The ILS-ESP also uses 'natural' parameters, which simplifies the calibration process. An extensive set of computational experiments has been carried out in order to answer the aforementioned research questions.

Keywords: Flow-Shop Problem, Scheduling, Iterated Local Search, Parallelizable algorithms, Biased-randomized heuristics, Metaheuristics, Parameters setting.

1. INTRODUCTION

The Permutation Flowshop Sequencing Problem (PFSP) is a well-known scheduling problem that can be described as follows: a set J of k independent jobs has to be processed on a set M of m independent machines. Each job $j \in J$ requires a given fixed processing time $p_{ij} \geq 0$ on each machine $i \in M$. Each machine can execute at most one job at a time, and all jobs are processed by the machines in the same order. The classical goal is to find a single sequence for processing the jobs in the shop so that a given criterion is optimized. The criterion most commonly used is the minimization of the maximum completion time, or makespan, denoted by C_{max}. **Figure 1** illustrates this problem for the simple case of $k = 3$ jobs and $m = 3$ machines.

The described problem is usually denoted as $Fm|prmu|C_{max}$, and it is a combinatorial problem with $k!$ possible sequences. As is the case with other combinatorial problems, a large number of different approaches have been developed to deal with the PFSP. These approaches range from the use of exact optimization methods, such as mixed integer programming or branch and bound algorithms for solving small-sized problems, to heuristics and metaheuristics that provide near-optimal solutions for medium and large-sized problems **(Ruiz and Maroto, 2005)**. Most of these methods focus on minimizing makespan. Some of them have reached outstanding efficiency levels, often using several parameters that require a fine-tuning process. This fine-tuning process is important, as the proper selection of these parameters has a significant impact on the performance of the algorithms, i.e.: the efficiency of these methods tends to be quite sensitive to the values assigned to each parameter **(Gendreau and Potvin,**

2005; Matsui and Yamada, 2007; Zobolas et al., 2009; Zheng and Yamashiro, 2010; Engin et al., 2011; Alabas and Dengiz, 2011; Cooren et al., 2011).

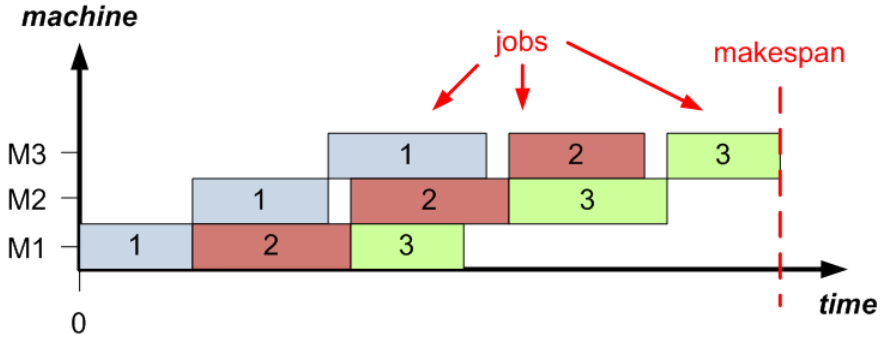

Figure 1: Flowshop Sequencing Problem

In this article, we discuss several issues regarding the use of an Iterated Local Search (ILS) framework **(Lourenço et al., 2010)** to develop efficient and parallelizable algorithms for the PFSP. Typically, these algorithms also use a reduced number of parameters with few setting requirements. In addition, ILS-based algorithms are easy to understand and to implement in a computer, which make them quite suitable for real-life applications and commercial software. Apart from considering parallelization and randomization issues in ILS-based approaches for the PFSP, in this paper we also propose a new Efficient, Simple, and Parallelizable (ESP) algorithm which moreover does not require advanced fine-tuning techniques – e.g. Design of Experiments (DOE) – which are also time consuming. The ILS-ESP algorithm employs basic 'common sense' rules for the local search, perturbation, and acceptance criterion stages of the ILS metaheuristic. In particular, a new operator which combines a swap (interchange) movement with a classical 'shift-to-left' movement is introduced for the perturbation process, thus avoiding any complex operators. Also, instead of using a traditional Simulated Annealing type acceptance criterion – which introduces a real-valued parameter that needs to be set –, a simpler acceptance-criterion rule is defined for this stage. A distinctive contribution of our approach is the introduction of a biased (non-uniform) randomization process during the construction of the initial solution. This process employs a skewed probability distribution to randomly generate different alternative initial solutions based on the well-known heuristic from Nawaz, Enscore and Ham (NEH) **(Nawaz et al., 1983)**. Thus, diversification of the local search starting point is attained in a simple, fast, and efficient manner. This diversification stage aims at avoiding poorly designed starting points and can provide benefits when applying parallel and distributed computing techniques. This randomization approach has similarities with the one proposed in Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristics **(Resende and Ribeiro, 2005)**. In fact, the algorithm presented here can be seen as a hybrid ILS-GRASP approach. However, while GRASP algorithms typically employs a uniform distribution and a restricted list of candidates during the construction process of a new solution, our approach proposes the use of a skewed distribution and considers the entire list of candidates.

The paper is organized as follows: Section 2 offers a basic literature review on the flowshop problem. Section 3 briefly describes the main ideas characterizing the GRASP and ILS metaheuristics, since our approach can be considered a hybridization of both types of algorithms. Section 4 describes the ILS-ESP algorithm. Sections 5 and 6 discuss three extensive computational experiments, which have been carried out to test the efficiency of the proposed algorithm, compare it against state-of-the-art approaches, and test the effects of increasing the computation time and number of parallel agents used in the ILS-ESP. Finally, Section 7 contains the conclusions of the paper.

2. LITERATURE REVIEW ON THE PFSP

A large number of heuristics and metaheuristics have been proposed to solve the PFSP, since it is very difficult to solve medium or large instances of the problem with exact methods. Most existing approaches focus on minimizing the makespan. **Johnson (1954)** proposed a simple procedure to obtain optimal sequences for the PFSP with two machines and three machines. **Campbell et al. (1970)** developed the CDS heuristic for solving the PFSP with more than two machines. **Dannenbring (1977)** also proposed several constructive heuristics for the general problem. **Nawaz et al. (1983)** introduced the NEH heuristic, which is commonly considered the best performing constructive heuristic for the PFSP. Basically, the NEH heuristic proposes calculating the total processing time required for each job –i.e. the total time each job requires to be processed by the set of machines– and then creating an 'efficiency list' of jobs sorted in descending order according to this total processing time. At each step, the job at the top of the efficiency list is selected and used to construct the solution. That is: the 'common sense' rule is to select first those jobs with the highest total processing time. Once selected, the job is inserted into the sorted set of jobs at a position that will minimize the makespan of this ongoing solution by using a 'shift-to-left' movement. **Taillard (1990)** introduced a data structure that reduces the NEH complexity. Some other interesting heuristics include those from **Suliman (2000)** or **Framinan and Leisten (2003)**, which consider several extensions of the NEH heuristic when facing objectives other than makespan. It should be noticed that NEH is the most commonly accepted method for the PFSP under the makespan minimization criterion, and it has been frequently used to provide an initial upper bound for the best branch-and-bound algorithms **(Ladhari and Haouari, 2005; Companys and Mateo, 2007)**.
Several metaheuristic approaches have also been proposed for the PFSP. **Osman and Potts (1989)** used Simulated Annealing (SA). **Widmer and Hertz (1989)** proposed a Tabu Search (TS) algorithm known as SPIRIT. Other Tabu Search algorithms also make use of the NEH heuristic **(Reeves, 1993; Moccellin, 1995)**. Also, Genetic Algorithms (GA) based on the NEH heuristic have been proposed for solving the PFSP **(Chen et al., 1995; Reeves, 1995; Aldowaisan and Allahvedi, 2003)**. Other metaheuristics, such as Ant Colony Optimization (ACO), have been used to obtain competitive or near-optimal solutions as well **(Chandrasekharan and Ziegler, 2004)**. **Ravetti et al. (2006)** propose hybrid heuristics that combine elements from the Greedy Randomized Adaptive Search Procedure (GRASP), Iterated Local Search (ILS), Path Relinking (PR) and Memetic Algorithm (MA). Their results are quite competitive when compared with existing algorithms. The efficiency of these procedures can be checked by comparing them against the best known solutions for the Taillard's benchmark instances **(Taillard, 1993)**.

What all of the aforementioned works have in common is that the algorithms proposed are relatively easy to code, and therefore the results can be reproduced without too much difficulty. In addition, many of the above algorithms can be adapted to other more realistic flowshop environments **(Ruiz and Maroto, 2005)**. There are other highly elaborate hybrid techniques for solving the PFSP. However, as **Ruiz and Stützle (2007)** point out, "...they are very sophisticated and an arduous coding task is necessary for their implementation." In other words, it is unlikely that they can be used for solving realistic scenarios without direct support from the researchers that developed them. For a complete description of heuristics and metaheuristics for the PFSP, we refer to the reader to more specialized references **(Framinan et al., 2004; Hejazi and Saghafian, 2005; Ruiz and Maroto, 2005)**.

In this paper, we will mainly focus on ILS-related approaches. The ILS metaheuristic has been applied successfully to a variety of combinatorial optimization problems. In certain cases, ILS algorithms achieve extremely high performance and even constitute the current state-of-the-art metaheuristics for some optimization problems. According to **Burke et al. (2010)**, who compared ILS against several hyper-heuristics, "...the implementation of Iterated Local Search produced the best overall performance. Interestingly, this is one of the most conceptually simple competing algorithms, its advantage as a robust algorithm is due to two factors: (i) the simple yet powerful exploration/exploitation balance achieved by systematically combining a perturbation followed by local search; and (ii) its parameter-less nature." For a detailed review

of existing ILS applications we refer to **Lourenço et al. (2010)**. Several algorithms based on the ILS framework have been applied to solve the PFSP. In particular, **Stützle (1998)** proposed a simple yet efficient ILS approach for this problem. Some years later, **Ruiz and Stützle (2007)** developed the Iterated Greedy (IG) method, which can be seen as an improved version of the Stützle's ILS metaheuristic. IG provides outstanding (state-of-the-art) results in terms of accuracy and speed and, for that reason, it deserves special attention. Despite its relative simplicity, it is one of the most efficient algorithms developed so far for the PFSP. In their work, **Ruiz and Stützle (2007)** tested IG against 11 different approaches –including various GA, TS, and SA. The experimental results showed that IG was the best-performing approach. Some other recent works relating the ILS method to the PFSP can be found in **Ravetti et al (2006), Pan et al. (2008)**, **Burke et al. (2010)**, and **Ribas et al. (2010)**. However, to the best of our knowledge, IG has performed better than any other algorithm (ILS-based or not) in all PFSP articles using the Taillard's benchmarks. During the last years, IG has become the method of reference in the PFSP field. Thus, for instance, in **Zobolas et al. (2009)** the authors show that their hybrid GA algorithm, NEGA-VNS, is able to compete with HGA-RMA, another hybrid GA algorithm previously developed by **Ruiz et al. (2006)**. However, as **Ruiz and Stützle (2007)** show in their work, IG is simpler and far superior to HGA-RMA. Also, **Ribas et al. (2010)** proposed several SA-based algorithms to compete with IG, but their results show that IG uses fewer parameters and performs better than their algorithms in all classical benchmarks. Finally, **Nagano et al. (2008)** tested their GA approach against a number of GA-based approaches. Some of the results in their paper are directly comparable to the results in **Ruiz and Stützle (2007)** (shared authors, same CPU, same maximum computing times, etc.). A comparison of both results shows that IG outperforms all considered metaheuristics.

As a relatively simple and yet extremely efficient algorithm, IG has inspired other approaches for different combinatorial optimization problems. For instance, **Kahraman et al. (2010)** developed a parallel greedy algorithm for the hybrid flow shop scheduling problem which uses the destruction-construction operator proposed in IG. Another example is the IG-based algorithm of **Pan et al. (2008)** for the no-wait flow shop scheduling problem.

3. OVERVIEW OF THE GRASP AND ILS METAHEURISTICS

Since the algorithm presented in this paper is inspired both in the GRASP and ILS metaheuristics –and to some extent it can be seen as a hybridization of both– these two methods are briefly described next. For a recent review of the GRASP and ILS methodologies the reader is referred to **Festa and Resende (2009a, 2009b)** and **Lourenço et al. (2010)**, respectively.

GRASP is a multi-start method designed to solve hard combinatorial optimization problems **(Feo and Resende, 1995)**. The basic methodology consists of two phases: (a) a constructive phase that builds a good but not necessarily locally optimal solution, and (b) a second phase which consists of a local search procedure. These two phases are repeated until a stopping criterion is reached, all the while keeping track of the best solution found overall in the search. The constructive phase builds step by step by adding an element to a partial solution following a greedy function. The selection of the element to be added, in each iteration, is not deterministic, but rather subject to a randomization process. That way, the repetition of both phases leads to different solutions. The randomization process is usually controlled by a parameter that in the simplest versions of GRASP is fixed along the execution of the algorithm. A particularly interesting GRASP is the so-called Reactive GRASP **(Prais and Ribeiro, 2000)**. In this version of the methodology, the parameter is not fixed along the running of the algorithm, but instead selected randomly from a set of discrete values. Initially, all values have the same probability of being chosen. After each iteration, we keep the value of the solutions which were obtained for each value of the parameter. After a certain number of iterations, the probabilities are modified. Those corresponding to values of the parameter which have produced good solutions are increased and, conversely, those corresponding to values producing low quality solutions are decreased.

The essential idea of Iterated Local Search (ILS) lies in focusing the search not on the full space of solutions but on a smaller subspace defined by the solutions that are locally optimal for a given optimization engine. **Figure 2** shows the general framework of the ILS procedure. To apply an ILS algorithm to an optimization problem, the four main components of the method must be specified in detail. These four components or processes are: *generate initial solution*, *local search*, *perturbation*, and *acceptance criterion*. For many applications, it is straightforward to first develop a basic version of ILS, since many of the components are common to other metaheuristics. For example, (i) one can start with a random solution; (ii) for most problems a local search algorithm is readily available; (iii) for the perturbation stage, a random move in a neighborhood of higher order than the one used by the local search algorithm can be effective; and (iv) a first-improvement acceptance criterion can be used. However, a state-of-the-art implementation requires the definition of more advanced components and operators. Also, the interaction among these components must be taken into account to improve the quality of the method.

```
procedure IteratedLocalSearch

01  initSol = generateInitialSolution  // initial solution generation process
02  baseSol = localSearch(initSol)  // local search process
03  bestSol = baseSol

04  while stopping condition not met do
05    aSol = perturbation(baseSol, history)  // perturbation process
06    aSol = localSearch(aSol)  // local search process
07    bestSol = updateBestSol(aSol)
08    baseSol = acceptanceCriterion(baseSol, aSol, history) // acceptance
process
09  end while

10  return bestSol

end
```

Figure 2: IteratedLocalSearch general framework

In the next sections, we propose, and test the ILS-ESP algorithm for solving the PFSP. With this algorithm, we show how it is possible to use only parameters with 'natural' values –which do not require calibration analyses– in an ILS-based framework without loosing performance with respect to a state-of-the-art algorithm such as IG. Our method, which will be described later in detail, introduces some GRASP principles inside the initial solution generation process of the ILS framework. One motivation that guided the design of the ILS-ESP method was to achieve many of the desirable features of a metaheuristic as described by **Cordeau et al. (2002)**: *accuracy*, *speed*, *simplicity* and *flexibility*. Most of the metaheuristics in the literature are measured against accuracy –the degree of departure of the obtained solution value from the optimal value–, and against speed –the computation time. However, there are two other important attributes to be considered in any metaheuristic: simplicity and flexibility. The simplicity is related to the number of parameters to be set and the facility of implementation. This is an important feature since the method can be applied to instances other than the ones tested without losing quality or performance and without the need of performing a long test run. Finally, flexibility is related to the possibility of accommodating new side constraints and also to the adaptation to other similar problems. Another fundamental motivation for developing the ILS-ESP was to design an easily parallelizable, yet easy-to-implement, method. That way, it could benefit from current multi-core processors and multi-thread programming techniques. Despite parallelization issues are critical in every modern method, as stated in **Ravetti et al. (2012)** these issues have been rarely considered in the existing PFSP literature. In fact, these authors propose a parallel hybrid search approach –combining a Memetic Algorithm and several IG algorithms– to efficiently solve the PFSP in a multi-threaded environment. Using an

optimized computer implementation of this hybrid approach, they run a set of experiments to obtain excellent results for most Taillard's instances. Their approach is complementary to ours in different ways, in particular: (a) while they analyze the effects of parallelization in long runs using a moderated number of threads (6 and 18), we focus on the effects of parallelization in short runs –i.e. 'real-time' solutions– using a larger number of threads (up to 50); (b) while we focus on developing a simpler-as-possible approach, their approach proposes a collaboration strategy between different algorithms; and (c) while our algorithm uses just a few parameters which can be easily calibrated, the Memetic Algorithm and the proposed parallelization architecture in their approach introduce many parameters which require from advanced fine-tuning processes. All in all, these two approaches are similar in the sense that they both show the importance of hybridization and parallelization issues in designing modern algorithms for the PFSP.

As will be discussed in the next section, the ILS-ESP method can also be related to Monte Carlo simulation or, simply, random sampling from a non-uniform distribution. In some previous works, **Juan et al. (2010, 2011b)** described the application of biased-randomized heuristics to solve the Capacitated Vehicle Routing Problem (CVRP). In particular, they employed a geometric distribution in order to introduce a skewed (biased) random behavior in a classical routing heuristic. Using biased randomization they were able to transform a deterministic (greedy) heuristic into a probabilistic algorithm without losing the logic behind the heuristic. Accordingly, a new heuristic-based solution is generated each time the randomized algorithm is run. Notice that this methodology could be seen as a kind of biased GRASP, where the uniform distribution is substituted by non-uniform (skewed) distributions in order to maintain most of the heuristic criteria. The aforementioned authors commented on the convenience of using similar approaches for combinatorial problems other than the CVRP: "it is convenient to highlight that the introduced methodology can be used beyond the CVRP scenario: similar hybrid algorithms based on the combination of Monte Carlo simulation with already existing heuristics can be developed for other routing problems and, in general, for other combinatorial optimization problems."

4. THE ILS-ESP ALGORITHM

The ILS-ESP algorithm uses the ILS metaheuristic as a framework, and combines it with a GRASP-like procedure. Therefore we will define the four components of any ILS-based algorithm (generate initial solution, local search, perturbation, and acceptance criterion) with emphasis on three original points **(Figure 3)** that make our algorithm significantly different from previous ILS-based algorithms such as those described in **Stützle (1998)** and **Ruiz and Stützle (2007)**.

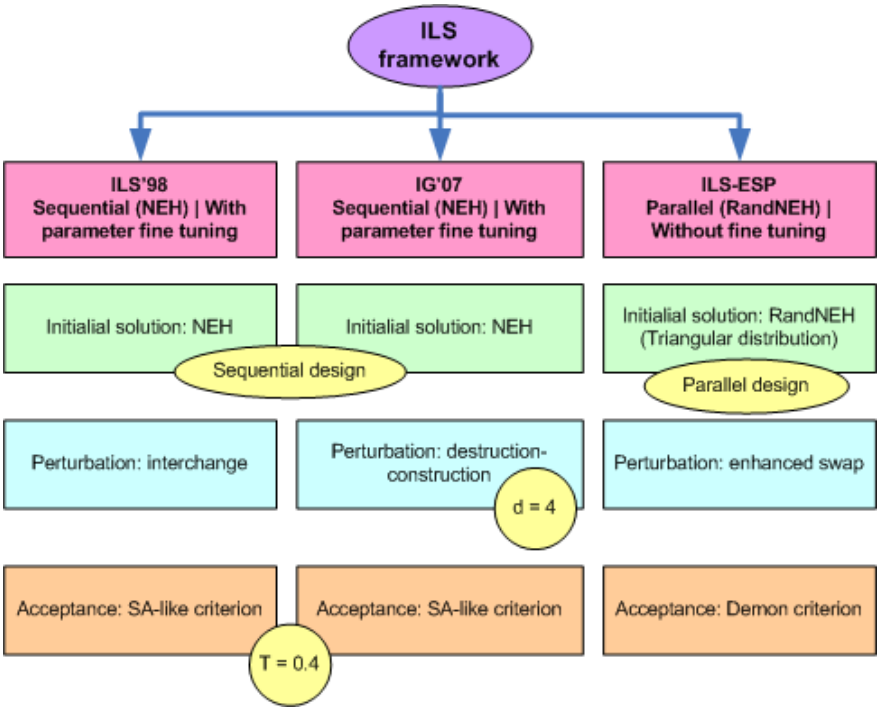

Figure 3: Scheme on the differences among different ILS-based approaches

The first of these three critical points is related to the *perturbation* component. During the perturbation process, the so-called 'enhanced-swap' operator is used. This is a very simple, fast, and efficient operator which basically does the following: (a) randomly selects (using a uniform distribution) two different jobs from the current solution; (b) interchanges both jobs, that is, interchanges their positions in the permutation; and (c) applies a classical 'shift-to-left movement' (like the one proposed in the NEH heuristic) to each of these jobs following a left-to-right order. The idea here is that we first consider a subset of the sequence of jobs by looking at the left-most swapped job and all elements to its left. Then we shift the right-most job of this subset and tentatively insert it into all possible positions of this sequence of jobs. Next, we select the sequence that results in the minimum makespan. Afterwards, we take this sequence and reinsert it into the full set of jobs. We then apply this idea again for the other swapped job. This 'shift-to-left movement' takes advantage of Taillard's accelerations **(Taillard, 1990)** to quickly determine the best position for each job when only the partial solution up to its position is considered. **Figure 4** shows the pseudo-code associated with this perturbation operator. Notice that the proposed operator is really simple and does not require any further parameter setting.

```
procedure enhancedSwap(aSol)

01  posA = selectRandomPosition(aSol) // selects a random job position in aSol
02  posB = selectRandomPosition(aSol)

03  aSol = swapJobs(aSol, posA, posB) // interchanges jobs at given positions

04  aSol = shiftToLeft(aSol, posA) // applies the NEH shiftToLeft operator
05  aSol = shiftToLeft(aSol, posB)

06  return aSol

end
```

Figure 4: enhancedSwap procedure to perform the ILS-ESP perturbation stage

A second critical difference of our algorithm is the *acceptance criterion*. The algorithm does not use a SA-based process like most other ILS-based algorithms, but instead uses a Demon-like process. As stated in **Talbi (2009, pp. 138)**, a Demon-like acceptance criterion is computationally simpler than a SA-like one since the former does not use pseudo-random numbers or real-valued parameters –which require specific calibration. This Demon-like acceptance criterion, together with the perturbation process, is designed to help avoid local minima during the algorithm execution. In order to do so, the criterion simply states the following basic principles: (a) any time a newly generated solution, *aSol*, improves the current base solution, *baseSol*, the base solution is updated (improved) to this new solution –likewise, this new solution is compared against the best-known solution, *bestSol*, to see if it must also be updated; and (b) even if a newly generated solution is worse than the base solution, the base solution will be updated (deteriorated) to this new solution as long as no consecutive deteriorations take place and the degradation does not exceed the last improvement. Notice that by allowing the base solution to degrade up to a certain level, the probability that the algorithm gets trapped at a local minimum is greatly reduced. **Figure 5** shows the pseudo-code associated with this acceptance criterion process.

```
01  delta = cost(currentSol) – cost(baseSol)

02  if delta < 0 then  // Case A: Improvement
03    credit = - delta
04    baseSol = currentSol
05    if cost(baseSol) < cost(bestSol) then bestSol = baseSol end if
06  end if

07  if 0 < delta <= credit then  // Case B: Deterioration
08    credit = 0
09    baseSol = currentSol
10  end if
```

Figure 5: pseudo-code for the ILS-ESP acceptance criterion stage

A third critical point of our approach –which contributes to make the algorithm parallelizable– is related to the starting solution used inside the ILS framework, *generate initial solution*. Usually, this starting solution is the one provided by the NEH heuristic, which generally produces a relatively good initial solution. Using the NEH solution instead of a randomly generated solution is typically considered good practice in order to accelerate the convergence of algorithms. However, it seems reasonable to think that when multiple runs of the same instance are executed –either in sequential or in parallel mode– using always the same starting point can be a severe drawback for fast convergence in those cases in which the NEH solution provides relatively 'poor' solutions. In this context, the term 'poor' does not necessarily refer to the makespan value of the solution, but rather to the number of movements or transformations that must be applied to the initial solution in order to arrive at a competitive or near-optimal solution. Since we are especially interested in running multiple iterations of any given instance, which can be seen as a form of biased (skewed) GRASP, we designed a way to generate different randomized NEH solutions with similar properties. As described before, the NEH heuristic is an iterative algorithm which uses a list of jobs sorted by their total completion time on all the machines to construct a solution for the PFSP. At each step of this iterative process, the NEH removes the job at the top of that list (with maximum completion time) and adds to it a new list at the position that results in the best partial solution with respect to makespan. As a result, the NEH provides a 'common sense' deterministic solution, by trying to schedule the most demanding jobs first. Our method instead assigns a probability to selecting each job in the list. According to our design, this probability should be coherent with the total time that each job needs to be processed by all the machines, i.e. jobs with higher total times will be more likely to be selected from the list before those with lower total times (biased distribution of probabilities). **Figure 6** shows the main pseudo-code associated with this process.

```
procedure RandNEH

01  nehJobsList = sortJobsUsingNehCriterion
02  nehSol = nehAlgorithm(nehJobsList) // NEH solution
03  baseSol = nehSol
04  nIter = 0

05  while cost(baseSol) >= cost(nehSol) and nIter < nJobs do
06    nIter = nIter + 1
07    newJobsList = biasedRandomization(nehJobList, triangular)
08    newSol = nehAlgorithm(newJobsList)
09    if getCost(newSol) < getCost(baseSol) then baseSol = newSol end if
10  end while

11  return baseSol

end
```

Figure 6: RandNEH procedure to perform the NEH biased randomization

To satisfy all of the aforementioned requirements, we employ a discretized version of the decreasing triangular distribution during the solution-construction process: each time a new job has to be selected from the list, a triangular distribution that assigns linearly diminishing probabilities to each eligible job according to its corresponding total-processing-time value is employed. Other skewed probability distributions –like the geometric one– have been successfully employed in routing problems by **Juan et. al (2010)** to generate multiple alternative solutions by inducing a similar biased-randomizaton process into a classical heuristic. In the case of the PFSP, the decreasing triangular probability distribution was chosen since it contains no parameters to be set and provides satisfactory results. That way, jobs with higher processing times are always more likely to be selected from the list first, but the assigned probabilities are variable and they depend upon the number of eligible jobs at each step. By iterating this procedure, a biased random search process is started. As a consequence, in most cases it is possible to obtain in just a few iterations (milliseconds for most tested instances) a randomized solution whose makespan is almost equal or even better than the original NEH solution **(see Figure 7)**. Notice that similar biased randomization processes can be developed for generating alternative initial solutions in other ILS-based metaheuristics, either in the context of the PFSP or in other combinatorial optimization problems. As the experimental section will show, this might be especially interesting when parallelization approaches are used to simultaneously run multiple instances of the algorithm. As a matter of fact, we consider this hybridization of ILS with GRASP-like metaheuristics as one of the major contributions of this paper, and one that should be explored in other combinatorial optimization problems.

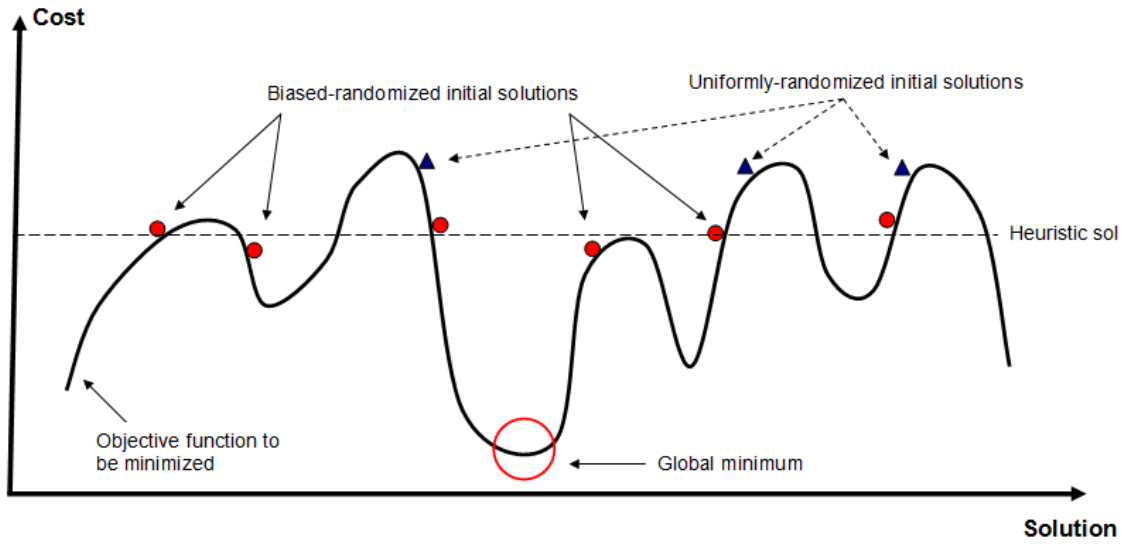

Figure 7: Diversification of initial solutions throughout biased randomization

Figure 8 shows the final pseudo-code of the ILS-EPS algorithm, which integrates the aforementioned perturbation operator, acceptance criterion, and randomization process into an ILS framework. The *local search* process that our algorithm uses is the traditional local search process used by most other authors, e.g. **Ruiz and Stützle (2007)**.

```
Procedure ILS-ESP

01  baseSol = RandNEH  // DIVERSIFICATION (NEH biased randomization)
02  baseSol = localSearch(baseSol)  // CLASSICAL LOCAL SEARCH
03  bestSol = baseSol

04  while stopping condition not met do  // ITERATED LOCAL SEARCH

05    currentSol = enhancedSwap(baseSol)  // PERTURBATION
06    currentSol = localSearch(currentSol)  // CLASSICAL LOCAL SEARCH

07    delta = cost(currentSol) – cost(baseSol)  // ACCEPTANCE CRITERION
08    if delta < 0 then  // Case A: Improvement
09      credit = - delta
10      baseSol = currentSol
11      if cost(baseSol) < cost(bestSol) then bestSol = baseSol end if
12    end if
13    if 0 < delta <= credit then  // Case B: Deterioration
14      credit = 0
15      baseSol = currentSol
16    end if

17  end while

18  return bestSol

end
```

Figure 8: ILS-ESP general procedure

The computational complexity of the ILS-ESP algorithm is discussed next. First of all, notice that the complexity of the RandNEH procedure is the same as the complexity of the NEH heuristic, which is employed in most modern meta-heuristics to generate an initial solution. Using Taillard's accelerations, its relative speed is $O(n^2m)$ **(Taillard, 1990)**. As described

before, the local search procedure is the same classical process employed in other similar ILS-based approaches. As discussed in **Ruiz and Stützle (2007)**, using Taillard's accelerations the complexity of evaluating the whole neighborhood of one solution is $O(n^2m)$. Although this evaluation is iteratively applied during the local search until the current solution cannot be improved any further, the number of iterations tends to be relatively low: each iteration is associated with a new improvement and that is something unusual, especially as we get closer to an optimal solution. In our opinion, however, the number of iterations is not a big issue since it could be limited to *n-1* without affecting the performance of the algorithm in a significant way. Finally, the perturbation procedure, which is also performed using Taillard's accelerations, is $O(n^2m)$. Since the stopping criterion we use in practice limits the computation time to some factor of $n \cdot m$, it can be considered that the complexity of the ILS-ESP algorithm is $O(n^3m^2)$, i.e. polynomial, as in other ILS-based algorithms.

Finally, notice that it seems reasonable to question whether or not using a 'high-quality' (pseudo-) Random Number Generator (RNG) can somewhat enhance the performance of ILS-based algorithms. In order to answer this natural question, we have run some tests using different RNGs as suggested by **L'Ecuyer (2001)**. These results will be discussed in Section 6.

5. TESTING THE ILS-ESP USING THE BEST AND AVERAGE METRICS

The *ILS-ESP* algorithm described in this paper was implemented as a Java application. Java was chosen for several reasons. First, it generates portable code which can run, without modifications, over different operating systems. This can be a significant advantage when executing a randomized algorithm in a parallel or distributed environment. Secondly, as one of the simplest object-oriented languages, it facilitates the rapid development of prototypes. Thirdly, according to **Luke (2009, pp. 196)** it is easier to guarantee duplicability of results in Java than in other languages such as C/C++. The expected counterpart is that, since Java code runs over a virtual machine, a Java version of an algorithm will execute somewhat slower than the corresponding C/C++ version.

An Intel Xeon at 2.0 GHz and 4 GB RAM was used to perform all tests, which were run directly on the Netbeans IDE platform for Java over Windows 7. In order to compare our ILS-ESP algorithm with other state-of-the-art ILS-based approaches, the following parameterized algorithms (with parameters D and T) were also coded in Java by the same programmers:

- The *ILS98-T04*, which is the algorithm proposed in **Stützle (1998)** using $T = 0.4$. According to the algorithm's author, this parameter value is the one offering the best performance, and it was obtained after a fine-tuning process.
- The *IG-D4T04* and *IG-D2T03*, which represent two different parameterizations ($D = 4$, $T = 0.4$ and $D = 2$, $T = 0.3$) of the well-known IG algorithm proposed in **Ruiz and Stützle (2007)**. The first set of parameters was obtained by the authors of IG after completing a DOE fine-tuning process. The second set of parameters was selected by us in order to show how performance of IG can be affected if other parameter values are selected instead of the 'optimal' ones. At this point, it is worthwhile to remember that in multiple experiments carried out by different authors **(Ruiz and Stützle, 2007; Zobolas et al., 2009; Ribas et al., 2010)**, the IG-D4T04 algorithm has outperformed any other algorithm so far, including GA and TS, which have more parameters than IG. Thus, the IG algorithm is a highly cited reference in the PFSP literature and, to the best of our knowledge, it is one of the most efficient algorithms in this field.
- The *RandIG-D4T04*, which is our proposal for a randomized version of the IG-D4T04, i.e. we have incorporated the GRASP-inspired *biased randomization* process developed for the ILS-ESP algorithm into the IG algorithm.

It is worthy to summarize at this point the main differences between our approach, ILS-ESP, and other existing ILS-based approaches can be summarized as follows:

- The GRASP-inspired biased randomization process, which is able to generate alternative initial solutions of 'good' quality. As will be shown in the experimental

section, this can be especially useful when multiple instances of the algorithm are run in parallel.

- The new perturbation operator, which according to our experiments is able to compete with the extremely efficient destruction-construction operator proposed in the IG algorithm. The latter operator, however, contains a fine-tuned parameter ($D = 4$) which determines how many jobs must be extracted during the destruction phase.
- A simple acceptance criterion component, which is based on a Demon-like process. This 'white-box' approach replaces the 'black-box' SA-like approach employed in other ILS-based algorithms, which also uses a fine-tuned parameter ($T = 0.4$).

For each one of the aforementioned algorithms, we designed and performed extensive tests – using the same machine, same programming language, same execution time, and same program developer – on the 120 Taillard's benchmark instances **(Taillard, 1993)**. These instances, which are available from http://mistic.heigvd.ch/taillard/default.htm, are grouped in 12 sets of 10 instances each according to the number of jobs and the number of machines, i.e.: set *20x5*, set *20x10*, set *20x20*, set *50x5*, set *50x10*, set *50x20*, set *100x5*, set *100x10*, set *100x20*, set *200x10*, set *200x20*, and set *500x20*. Thus, for each ILS implementation and for each tested instance, 10 independent iterations (replicas) were run. Each replica was run for a maximum time $t_{max} = 0.01s \times k \times m$, where k is the *number of jobs*, and m is the *number of machines*. Then, for each set of replicas, the best experimental solution found (BEST10) as well as the average value of the different replicas (AVG10) were registered. Also, the best-known solution (BKS) associated with each instance was obtained either from the aforementioned website or from **Zobolas et al. (2009)**. Notice that the t_{max} we are employing is really a small value in terms of computational times. Thus, for the smallest instances $t_{max} = 1s$, while for the largest ones $t_{max} = 100s$. While analyzing the results it is important to keep in mind these short computational times and the real difficulty of the selected benchmarks. As stated by **Zobolas et al. (2009)**: "It should be mentioned that the best known solutions in difficult instances are usually found with branch and bound techniques or other exact methods in powerful workstations run for extended time periods, and thus are not directly comparable to metaheuristic methods designed or intended to run on single processor PCs and provide high-quality solutions in short computational times."

In the following subsections, the results associated with the BEST10 values and those associated with the AVG10 values are respectively analyzed and discussed.

5.1. A COMPARISON USING THE BEST10 METRIC

Table 1 shows, for each tested algorithm and set of instances, a summary of the experimental results when considering the gap between the BKS and the best-found solution in 10 runs.

From the averages in the last row of the table, we can see that all tested ILS-based algorithms perform quite well on the average –taking into account the maximum time each instance is executed. However, it seems that our randomized version of the IG algorithm with optimal parameter settings, the RandIG-D4T04, is the one showing the best performance (average gap = 0.33%). Just a slightly worse than this randomized version, both the optimally parameterized IG-D4T04 and our ILS-ESP seem to perform equally well (average gap = 0.36%). Then, we have the parameterized ILS98-T04 (average gap = 0.37%). Finally, far from the rest, we find the non-optimally parameterized IG-D2T03 (average gap = 0.44%). Notice that this result seems to imply that the IG algorithm offers some degree of sensitivity with respect to its parameters, i.e. its performance can be greatly reduced when non-optimal values are assigned to its parameters. **Figure 9** shows a multiple-boxplot which allows for a visual comparison of the performance of the algorithms. This figure reinforces the idea that results from the first four approaches are quite equivalent, although maybe the RandIG-D4T04 performs slightly better than the rest. It also seems clear from this figure that the non-optimized version of the IG algorithm performs slightly worse than the rest.

Table 1: Gaps between Best Known Solution and BEST10. Java code running on an Intel Xeon at 2.0 GHz.

Taillard set	RandIG-D4T04	IG-D4T04	ILS-ESP	ILS98-T04	IG-D2T03	Max. Time (s)
20_5	0.00%	0.04%	0.00%	0.04%	0.00%	1
20_10	0.00%	0.00%	0.00%	0.00%	0.04%	2
20_20	0.00%	0.01%	0.00%	0.00%	0.03%	4
50_5	0.00%	0.00%	0.00%	0.00%	0.00%	2.5
50_10	0.38%	0.48%	0.47%	0.45%	0.53%	5
50_20	0.62%	0.66%	0.71%	0.74%	1.00%	10
100_5	0.00%	0.01%	0.00%	0.01%	0.00%	5
100_10	0.10%	0.07%	0.10%	0.07%	0.10%	10
100_20	0.94%	1.04%	1.13%	1.07%	1.25%	20
200_10	0.08%	0.08%	0.09%	0.11%	0.14%	20
200_20	1.18%	1.29%	1.24%	1.32%	1.48%	40
500_20	0.62%	0.63%	0.63%	0.67%	0.71%	100
Averages	*0.33%*	*0.36%*	*0.36%*	*0.37%*	*0.44%*	--

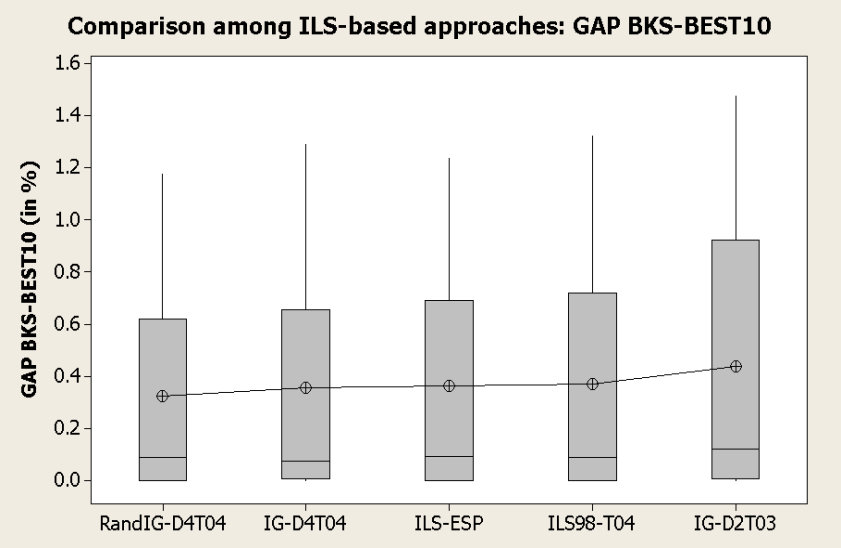

Figure 9: Multiple box-plot for the BEST10 metric

An ANOVA test for comparing the average performance of each algorithm using the BEST10 metric was also performed. According to the test results (p-value = 0.984) and the overlapping 95% confidence intervals, no statistically significant difference has been found among the various algorithms' means. In this case, however, the normality assumption has not been met and, therefore, we also completed a Kruskal-Wallis non-parametric test. The test results (p-value = 0.949) confirmed the absence of statistically significant differences among the average performance of the considered algorithms.

It is worthwhile to note that, in our opinion, the BEST10 metric is of great relevance since it is strongly related to the use of multi-process capabilities of current (and future) computers. In fact, most current workstations include multi-core processors. Therefore, different instances of randomized algorithms –like the ones compared in this paper– can be run in parallel in a single machine without increasing the total clock time employed after conveniently setting the initial seeds for the RNG. Of course, by using a computer cluster instead of a single machine, even

more instances of a randomized algorithm can be run for a given clock time. Traditionally, however, the metric against which to compare sequential algorithms in the PFSP literature is the one referred to as the average of several iterations. This average value helps to reduce somewhat the 'randomness effect', due to which a different solution is likely to be obtained each time a randomized algorithm is run. The next subsection analyzes the results of our tests using the average metric.

5.2. A COMPARISON USING THE AVG10 METRIC

Table 2 shows, for each tested algorithm and set of instances, a summary of the experimental results when considering the gap between the BKS and the average solution in 10 runs.

Table 2: Gaps between Best Known Solution and AVG10. Java code running on an Intel Xeon at 2.0 GHz.

Taillard set	**RandIG-D4T04**	**IG-D4T04**	**ILS-ESP**	**ILS98-T04**	**IG-D2T03**	**Max. Time (s)**
20_5	0.04%	0.04%	0.05%	0.05%	0.07%	1
20_10	0.05%	0.05%	0.06%	0.05%	0.17%	2
20_20	0.04%	0.05%	0.06%	0.05%	0.17%	4
50_5	0.00%	0.01%	0.01%	0.01%	0.02%	2.5
50_10	0.71%	0.69%	0.78%	0.69%	0.84%	5
50_20	1.02%	1.04%	1.08%	1.11%	1.48%	10
100_5	0.02%	0.01%	0.02%	0.04%	0.03%	5
100_10	0.28%	0.27%	0.32%	0.31%	0.37%	10
100_20	1.46%	1.47%	1.55%	1.46%	1.68%	20
200_10	0.23%	0.23%	0.26%	0.26%	0.28%	20
200_20	1.57%	1.52%	1.57%	1.57%	1.73%	40
500_20	0.78%	0.80%	0.79%	0.83%	0.84%	100
Averages	*0.52%*	*0.52%*	*0.55%*	*0.54%*	*0.64%*	--

From the last row in the table, it can be seen that both RandIG-D4T04 and IG-D4T04 perform equally well for this metric (average gap = 0.52%) while our ILS-ESP and the ILS98-T04 are just slightly behind (average gaps = 0.55% and 0.54% respectively). Lower results are obtained for the non-optimally parameterized IG-D2T03 (average gap = 0.64%). Again, this result seems to imply that performance of the IG algorithm greatly depends on the proper selection of its parameters. **Figure 10** shows a multiple-boxplot comparing the performance of the considered algorithms. The visual comparison reinforces the idea that results from the first four approaches are quite equivalent, although maybe the RandIG-D4T04 and the IG-D4T04 perform slightly better for the AVG10 metric than the ILS-ESP and the ILS98-T04. It seems also clear from the figure that the non-optimized version of the IG algorithm performs slightly worse than the rest.

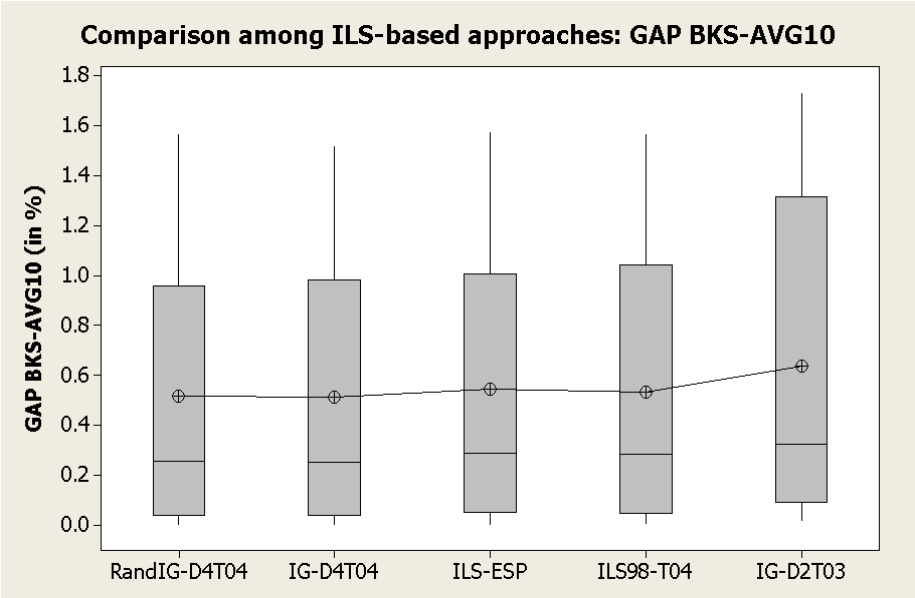

Figure 10: Multiple box-plot for the AVG10 metric

An ANOVA test for comparing the average performance of each algorithm, using the AVG10 metric, was also completed. According to the test results (p-value = 0.985) and the overlapping 95% confidence intervals, no statistically significant difference was found among the various algorithms' means. Again, the normality assumption was not met, and a Kruskal-Wallis test was also carried out (p-value = 0.895) to support the lack of significant differences among the algorithms' average performance.

To conclude this section, it is important to notice that despite its simplicity and lack of non-trivial fine-tuning processes, the proposed ILS-ESP algorithm shows itself to be quite efficient, both with respect to the BEST10 and the AVG10 metrics. This is quite interesting in our opinion since, as noted before, some of the most efficient metaheuristics are not used in practice because of the difficulties they present when trying to implement them and because of the fact that they can sometimes seem like "black-boxes" to non-experts. On the contrary, simple approaches like the one introduced here tend to be more flexible and transparent which, in turn, makes them seem more credible and therefore, as some authors suggest, more likely to be used in real-life scenarios **(Robinson, 1997)**. Finally, with regard to the BEST10 metric, notice that RandIG-D4T04, our randomized version of IG, seems to slightly outperform the original IG-D4T04 version. To the best of our knowledge, it is the first time the IG algorithm has been outperformed. The result also suggests the convenience of introducing GRASP-like approaches into ILS metaheuristics to diversify the generation of initial solutions.

6. ANALYZING THE EFFECT OF RANDOMIZATION AND PARALLELIZATION

In order to test whether or not the selection of a specific random number generator (RNG) has a significant effect on the performance of the ILS-based algorithms, the experiment described next was carried out. We considered three different RNGs to generate the continuous random numbers employed in the SA-like acceptance criterion of both IG and ILS98. The three RNGs selected were the two Java native generators (`java.util.Random` and `Math.random()`), and the LFSR113 generator proposed by **L'Ecuyer (2001)**. Then, for each of these RNGs, all 120 Taillard's instances were run using a maximum time $t_{max} = 0.01s \times k \times m$ (where k is the *number of jobs*, and m is the *number of machines*). Finally, an ANOVA test was used to compare the three sets of outputs (one per RNG). The ANOVA test showed no significant differences (p-value = 0.926) among the results obtained with each of these RNGs. Therefore, no statistical

evidence was found to support the idea that selecting one RNG or another could make a significant difference in the performance of the algorithm.

As previously noted, the ILS-ESP algorithm proposed here can easily be parallelized by splitting the random-number-generation sequence into different streams and using each stream in different threads or CPUs. This can be an interesting field to explore, given the current trend in multi-core processors and parallel computing. Offering competitive solutions to complex problems in real time and without adjustments beforehand still presents a challenge. However, it has been empirically observed that it is possible to significantly reduce the execution time that randomized algorithms need to obtain 'good' solutions, depending on the seed that is chosen for the pseudo-random number generator **(Juan et al., 2011a)**. There are new processor design paradigms based on gaining computation capacity through the parallel execution of multiple processes and threads (multi-core). All in all, the idea is to execute multiple instances or replicas of the algorithm in parallel, each replica using a different seed for the RNG. Each of these instances can be considered an individual agent that is searching the solution space. In other words, the idea is that each of these multiple agents will start to search in a different region of the solution space by using different seeds. Our hypothesis here is that, as a randomized approach with diversified initial solutions, a parallel version of the algorithm can provide very competitive results in 'real time' for most medium-size PFSP problems.

To test how parallelization could improve our approach and, in particular, how increasing the number of replicas can potentially enhance the ILS-ESP results, we have carried out two extensive experiments which are described in the next subsections.

6.1. COMPARING DIFFERENT ALGORITHMS WHEN PARALLEL RUNS ARE EXECUTED

For the first experiment on parallelization, we considered the Taillard's *50x20* set of instances. These instances have been selected because they are among the ones that most of the algorithms with better performance find difficult to solve –and, therefore, we expect performance differences to be more significant in this set of instances than in other sets. As with previous experiments, the termination condition of each execution or replica is given by an instance-size dependent maximum time $t_{max} = 0.01s \times k \times m$, i.e., we are considering a time factor of 0.01 seconds. Once more, we will use the BEST and AVG metrics defined before and we will compare the performance of the different ILS-based approaches. The difference now with regards to previous experiments is that we will analyze how results evolve as the number of replicas is increased from $n = 10$ to $n = 80$. **Table 3** and **Figure 11** show the results obtained for the BEST(n) metric.

Table 3: Gaps between Best Known Solution and BEST(n) for Taillard 50x20.

Number of replicas	RandIG-D4T04	IG-D4T04	ILS-ESP	ILS98-T04	IG-D2T03	Max. Time (s)
10	0.62%	0.66%	0.66%	0.74%	1.00%	10
20	0.59%	0.62%	0.62%	0.64%	0.88%	10
30	0.53%	0.59%	0.54%	0.63%	0.83%	10
40	0.52%	0.58%	0.52%	0.55%	0.80%	10
50	0.48%	0.58%	0.47%	0.52%	0.75%	10
60	0.46%	0.56%	0.47%	0.52%	0.75%	10
70	0.46%	0.55%	0.45%	0.51%	0.73%	10
80	0.46%	0.51%	0.45%	0.51%	0.71%	10

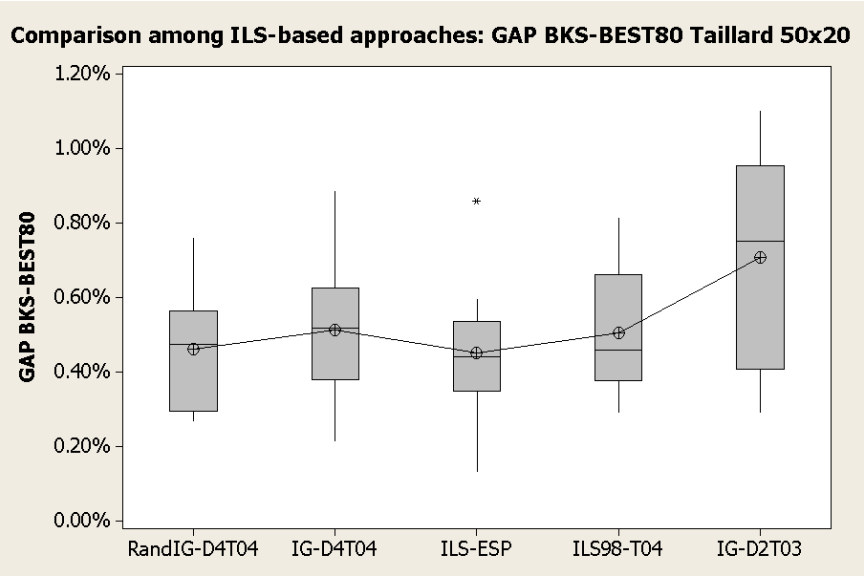

Figure 11: Multiple box-plot for the BEST80 metric

Notice that there is a visible difference between the non-optimized version of the IG algorithm (the IG-D2T03) and the rest of algorithms. The resulting ANOVA test is quite close to showing statistically significant differences among the algorithms (p-value = 0.051). Also, notice that as the number of replicas is increased, both the ILS-ESP and the RandIG-T4T04 seem to benefit from our randomized NEH process which diversifies the starting solution in the ILS framework. Thus, for n = 80 the ILS-ESP provides a 0.45% gap, which is slightly lower than the rest of the ILS-based approaches. In our opinion this is an interesting result, especially when considering the simplicity of the ILS-ESP algorithm and the fact that it contains no specific-value parameters. Finally, **Table 4** and **Figure 12** show the results obtained for the AVG(n) metric.

Table 4: Gaps between Best Known Solution and AVG(n) for Taillard 50x20.

Number of replicas	RandIG-D4T04	IG-D4T04	ILS-ESP	ILS98-T04	IG-D2T03	Max. Time (s)
10	1.06%	1.10%	1.20%	1.12%	1.48%	10
20	1.05%	1.07%	1.16%	1.13%	1.46%	10
30	1.04%	1.05%	1.18%	1.12%	1.46%	10
40	1.03%	1.06%	1.16%	1.12%	1.46%	10
50	1.03%	1.07%	1.16%	1.11%	1.47%	10
60	1.03%	1.07%	1.17%	1.12%	1.48%	10
70	1.03%	1.07%	1.16%	1.12%	1.48%	10
80	1.03%	1.07%	1.16%	1.12%	1.47%	10

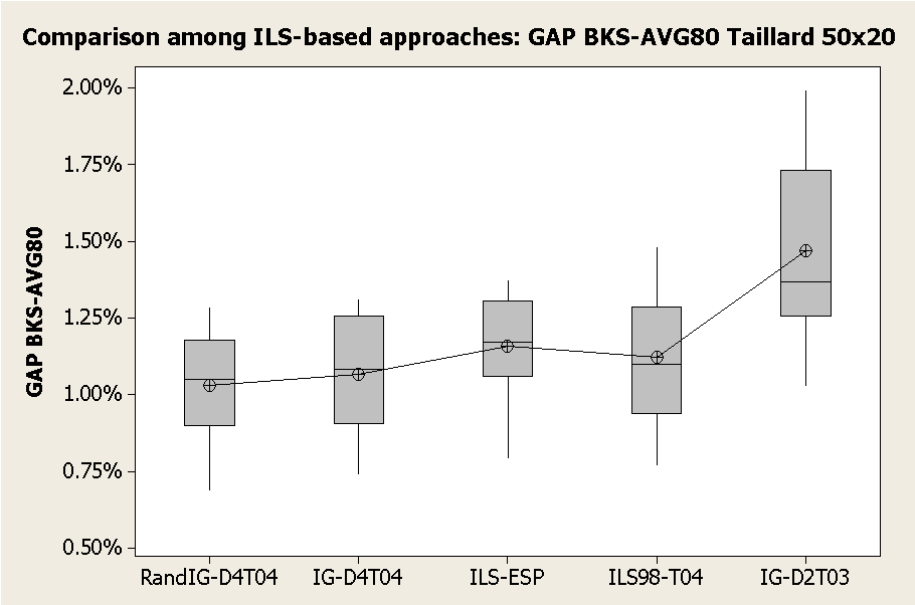

Figure 12: Multiple box-plot for the AVG80 metric

Once more, there is a noticeable difference between the non-optimized version of the IG algorithm and the rest of the approaches. Under this metric, the best performance for $n = 80$ is attained by the RandIG algorithm, that is, our randomized version of the IG algorithm. The resulting ANOVA test shows the existence of significant differences among the different algorithms (p-value = 0.001).

6.2. COMPARING RESULTS AT DIFFERENT FACTOR TIMES AND NUMBER OF RUNS

As a final experiment, we decided to analyze how the quality of the results generated by the ILS-ESP will vary when considering different levels of computation time and number of parallel agents –i.e. independent runs of the algorithm. These tests were carried out on the Taillard's *50x20* instances as well as the Taillard's *100x20* and the Taillard's *200x20* instances. For each instance in these sets, 50 runs of our algorithm were executed with different randomized starting seeds. The best found solution at each of the following times was then registered: $t_{max} = t_{factor} \times k \times m$, where the time factor, t_{factor}, took on the values of *0.01*, *0.02*, *0.03*, *0.04*, and *0.05* seconds. For $t_{factor} = 0$, we used the solution provided by the NEH heuristic. Then, for each combination of time factor and number of parallel runs, the average gap for the set of instances was computed as the average percentage difference between our best found solution (OBS) and the best known solution for each instance in the set. **Figure 13**, **Figure 14**, and **Figure 15** display the results of our experiments for the *50x20*, *100x20* and *200x20* Taillard's sets, respectively. Several items are worthy of note in these graphs. First of all, there is an enormous improvement from the NEH solution in very little time. With a time factor as small as *0.01*, we can see a more than 5% improvement, even with only one single agent or run. However, after this initial jump, the solution improves slowly with time. Between a time factor of 0.01 and 0.05, which is a difference of 40 seconds in t_{max} for the *50x20* cases, the solution improves by less than 0.5%. Another interesting result is that with as few as ten parallel agents, we can obtain very good results in a very short amount of time. Notice that with a time factor of *0.01* for ten parallel agents, the average gap is almost nonexistent. Significantly more time would be needed to achieve this result using only one sequential run of the algorithm. In other words, parallelization seems to offer clear benefits to these types of algorithms. Even more noticeably, through parallelization, it is possible to attain competitive solutions in 'real-time' – just a few seconds for problems of the considered sizes.

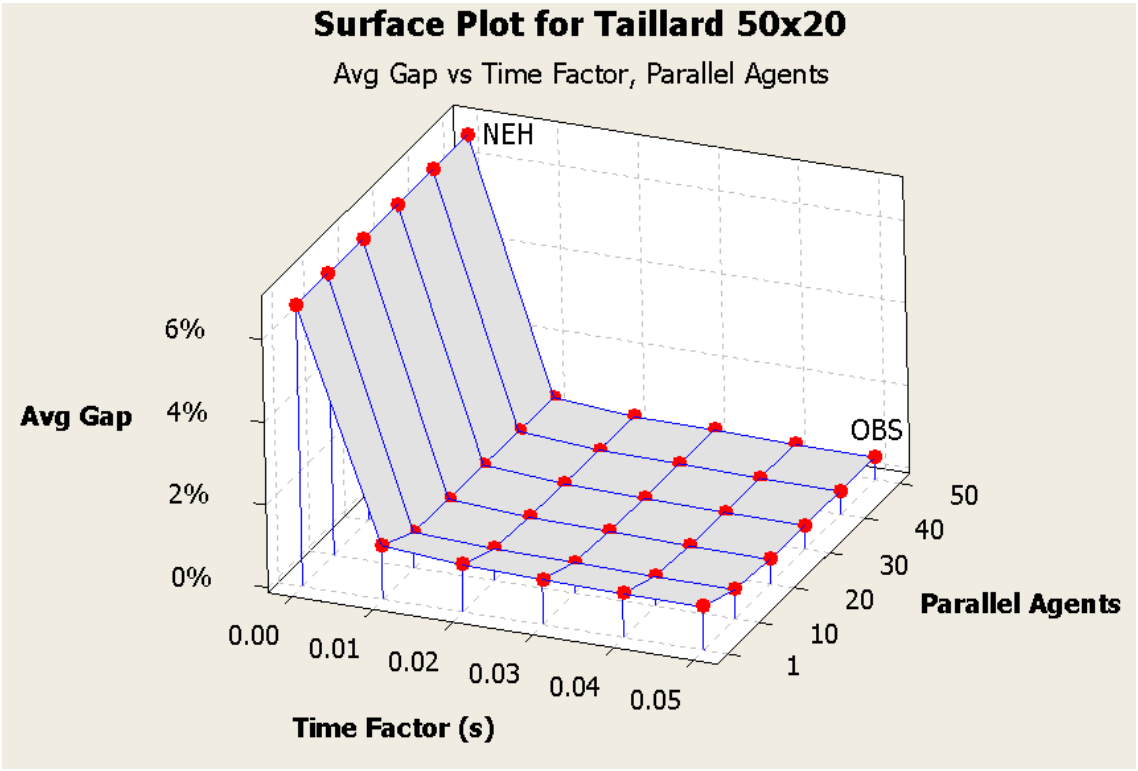

Figure 13: Surface plot for Taillard's 50x20 set.

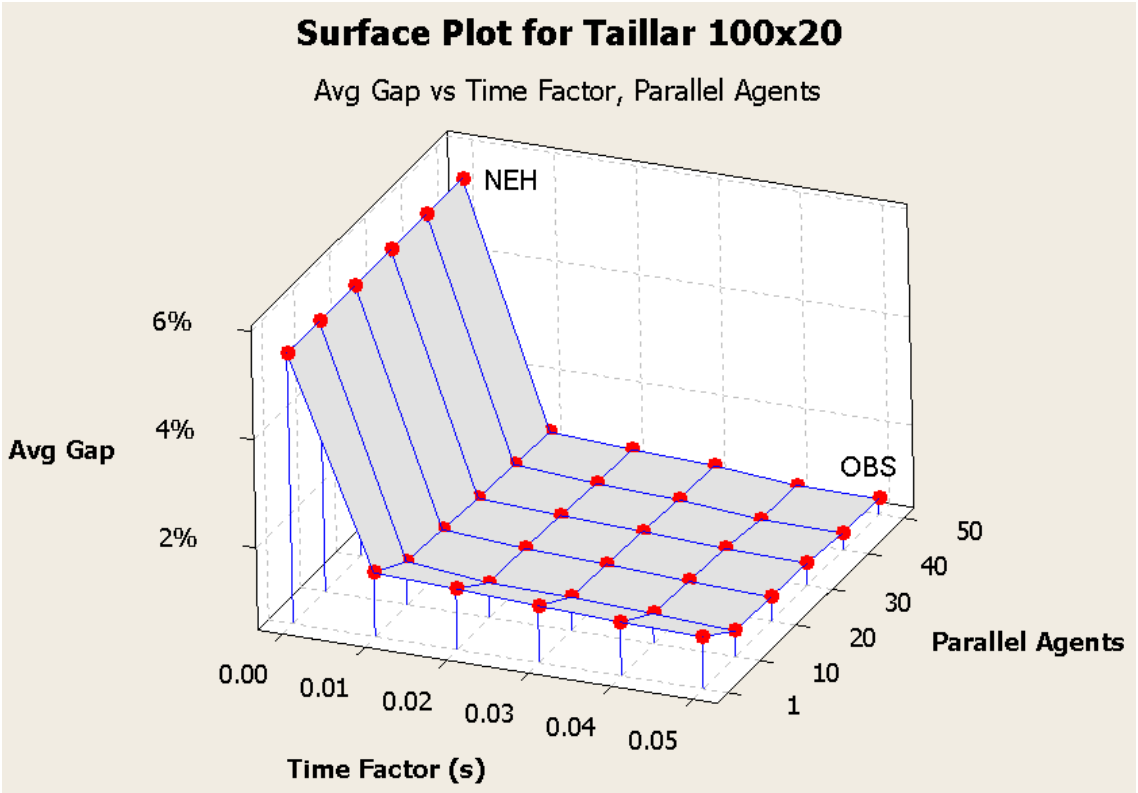

Figure 14: Surface plot for Taillard's 100x20 set.

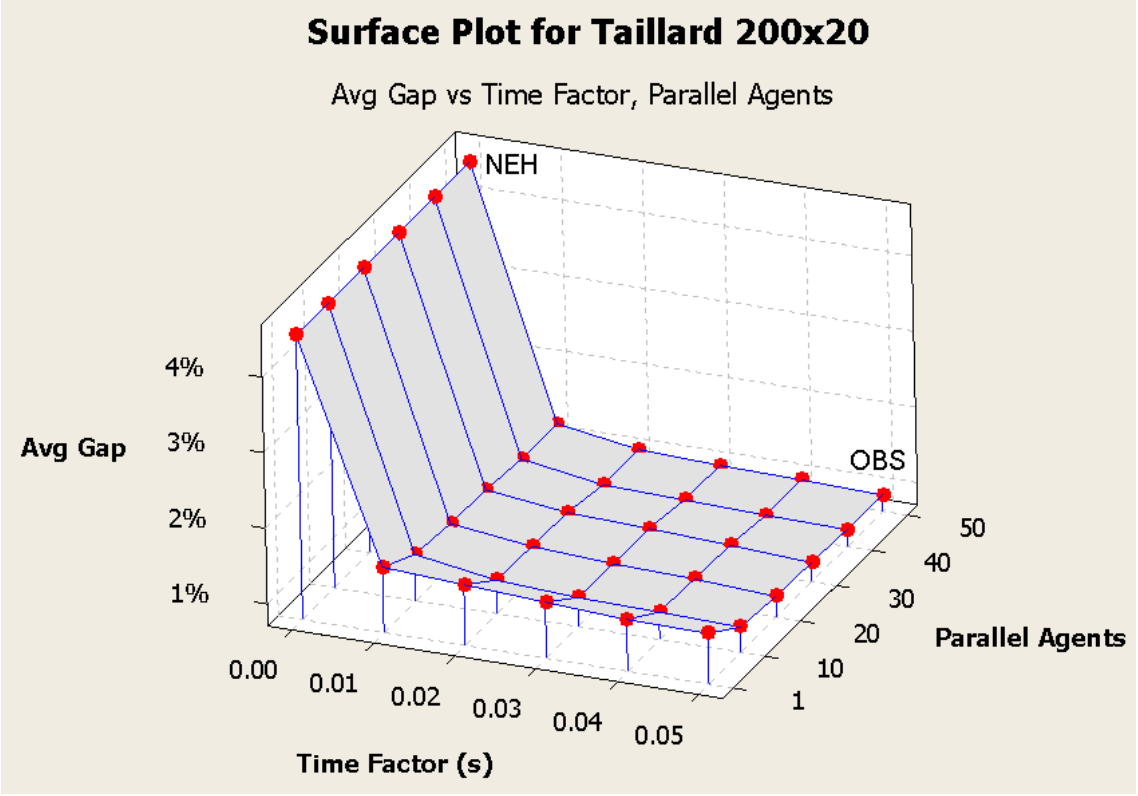

Figure 15: Surface plot for Taillard's 200x20 set.

7. CONCLUSION

In this paper, we have reviewed several algorithms based on the Iterated Local Search (ILS) framework for solving the Permutation Flow Shop Problem (PFSP). The paper offers a literature review on the PFSP and discusses some of the benefits ILS-based algorithms offer to solve this problem, including: (a) their efficiency; and (b) the fact that they can be easily implemented without requiring time-consuming fine-tuning processes. The paper also introduces our ILS-ESP algorithm, which shows how it is possible to develop an efficient and parallelizable ILS-based algorithm without requiring a parameter-calibration analysis. We explain how the ILS-ESP algorithm employs a diversification strategy to benefit from parallelization techniques. This diversification in the initial solution is attained by employing a biased randomized version of the well-known NEH heuristic for the PFSP. In particular, we propose the use of a discretized triangular distribution, which allows for the generation of alternative initial solutions without losing the 'common-sense' ideas behind the NEH heuristic. Several extensive tests show that parallelization strategies can greatly contribute to improving the performance of ILS-based algorithms. In fact, by using parallelization, we have been able to obtain competitive or near-optimal solutions in 'real-time' (a few seconds) even for the most challenging Taillard's instances. Finally, we have not observed significant differences when employing alternative random number generators to run ILS-based algorithms –although this remains an open research topic to be explored in other metaheuristics and optimization problems.

ACKNOWLEDGMENTS

This work has been partially supported by the Spanish Ministry of Science and Innovation (grants TRA2010-21644-C03, ECO2009-11307, and DPI2007-61371), and by the CYTED-HAROSA Network (http://dpcs.uoc.edu).

REFERENCES

Alabas-Uslu, C., Dengiz, B., 2011. A self-adaptive local search algorithm for the classical vehicle routing problem. *Expert Systems and Applications* 38, 8990-8998.

Aldowaisan, T., Allahvedi, A., 2003. New heuristics for no-wait flowshops to minimize makespan. *Computers and Operations Research* 30(8), 1219-1231.

Burke, E., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vazquez-Rodriguez, J.A., Gendreau, M., 2010. Iterated Local Search vs. Hyper-heuristics: Towards General-Purpose Search Algorithms. In: *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 1-8.

Campbell, H.G., Dudek, R.A., Smith, M.L., 1970. A heuristic algorithm for the n job, m machine sequencing problem. *Management Science* 16, B630-B637.

Chandrasekharan, R., Ziegler, H., 2004. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research* 155(2), 426-438.

Chen, C.L., Vempati, V.S., Aljaber, N., 1995. An application of genetic algorithms for flow shop problems. *European Journal of Operational Research* 80(2), 389-396.

Companys, R., Mateo, M., 2007. Different behaviour of a double branch-and-bound algorithm on $Fm|prmu|C_{max}$ and $Fm|block|C_{max}$ problems. *Computers & Operations Research* 34, 938-953.

Cooren, Y., Clerc, M., Siarry, P., 2011. MO-TRIBES: an adaptive multiobjective particle swarm optimization algorithm. *Computational Optimization and Applications* 49(2), 379-400.

Cordeau, J.F., Gendreau, M., Laporte, G., Potvin, J.Y., Semet, F., 2002. A guide to vehicle routing heuristics. *Journal of the Operational Research Society* 53, 512-522.

Dannenbring, D.G., 1977. An evaluation of flowshop sequence heuristics. *Management Science* 23, 1174-1182.

Engin, O., Ceran, G., Yilmaz, M.K., 2011. An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems. *Applied Soft Computing* 11, 3056-3065.

Feo, T.A., Resende, M.G.C., 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6, 109-133.

Festa, P., Resende, M.G.C., 2009a. An annotated bibliography of GRASP Part I: algorithms. *International Transactions in Operational Research* 16, 1-24.

Festa, P., Resende, M.G.C., 2009b. An annotated bibliography of GRASP Part II: applications. *International Transactions in Operational Research* 16, 131-172.

Framinan, J.M., Gupta, J.N.D., Leisten, R., 2004. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society* 55, 1243-1255.

Framinan, J.M., Leisten, R., 2003. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. *OMEGA* 31, 311-317.

Gendreau, M., Potvin, J.Y., 2005. Metaheuristics in combinatorial optimization. *Annals of Operations Research* 140(1), 189-213.

Hejazi, S.R., Saghafian, S., 2005. Flowshop-scheduling with makespan criterion: a review. *International Journal of Production Research* 43, 2895-2929.

Johnson, S.M., 1954. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly* 1, 61-68.

Juan, A., Faulin, J., Jorba, J., Caceres, J., Marques, J., 2011a. Using Parallel & Distributed Computing for Solving Real-time Vehicle Routing Problems with Stochastic Demands. *Annals of Operations Research*, 1-22., doi 10.1007/s10479-011-0918-z.

Juan, A., Faulin, J., Jorba, J., Riera, D., Masip, D., Barrios, B., 2011b. On the Use of Monte Carlo Simulation, Cache and Splitting Techniques to Improve the Clarke and Wright Savings Heuristics. *Journal of the Operational Research Society* 62, 1085-1097.

Juan, A., Faulin, J., Ruiz, R., Barrios, B., Caballe, S., 2010. The SR-GCWS hybrid algorithm for solving the capacitated vehicle routing problem. *Applied Soft Computing* 10(1), 215-224.

Kahraman, C., Engin, O., Kaya, I., Ozturk, R.E., 2010. Multiprocessor task scheduling in multistage hybrid flow-shops: A parallel greedy algorithm approach. *Applied Soft Computing* 10, 1293-1300.

L'Ecuyer, P., 2001. Software for uniform random number generation: Distinguishing the good and the bad. In: *Proceedings of the 2001 Winter Simulation Conference*, pp. 95-105.

Ladhari, T., Haouari, M., 2005. A computational study of the permutation flow shop problem based on a tight lower bound. *Computers & Operations Research* 32, 1831-1847.

Lourenço, H.R., Martin, O., Stützle, T., 2010. Iterated Local Search: Framework and Applications. In: *Handbook of Metaheuristics*, Kluwer Academic Publishers, International Series in Operations Research & Management Science vol. 146, pp. 363-397.

Luke, S., 2009. *Essentials of Metaheuristics*. Lulu.

Matsui, S., Yamada, S., 2007. An Empirical Performance Evaluation of a Parameter-free Genetic Algorithm for Job-Shop Scheduling Problem. In: *Proceedings of the 2007 IEEE Congress on Evolutionary Computation*, pp. 3796-3803.

Moccellin, J.V., 1995. A new heuristic method for the permutation flow-shop scheduling problem. *Journal of the Operational Research Society* 46, 883-886.

Nagano, M.S., Ruiz, R., Nogueira, L.A., 2008. A Constructive Genetic Algorithm for permutation flowshop scheduling. *Computers & Industrial Engineering* 55, 195-207.

Nawaz, M., Enscore, E., Ham, I., 1983. A heuristic algorithm for the *m*-machine, *n*-job flowshop sequencing problem. *OMEGA* 11, 91-95.

Osman, L., Potts, C., 1989. Simulated annealing for permutation flow-shop scheduling. *OMEGA* 17(6), 551-557.

Pan, Q.K., Wang, L., Zhao, B.H., 2008. An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. *International Journal of Advanced Manufacturing Technology* 38(7-8), 778-786.

Prais, M., Ribeiro, C.C., 2000. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing* 12, 164-176.

Ravetti, M.G., Nakamura, F.G., Meneses, C.N., Resende, M.G.C., Mateus, G.R., Pardalos, P-M-2006. Hybrid heuristics for the permutation flow shop problem, *AT&T Labs Research Technical Report* TD-6V9MEV, Shannon Laboratory, Florham Park, NJ 07932 USA.

Ravetti, M.G., Riveros, C., Mendes, A., Resende, M.G.C., Pardalos, P. 2012. Parallel hybrid heuristics for the permutation flow shop problem. *Annals of Operations Research* 199, 269-284.

Reeves, C.R., 1993. Improving the efficiency of tabu search for machine scheduling problems. *Journal of the Operational Research Society* 44(4), 375-382.

Reeves, C.R., 1995. A genetic algorithm for flowshop sequencing. *Computers and Operations Research* 22(1), 5-13.

Resende, M.G.C., Ribeiro, C.C., 2005. GRASP: Greedy Randomized Adaptive Search Procedures. In: *Search Methodologies*, Springer.

Ribas, I., Companys, R., Tort-Martorell, X., 2010. Comparing three-step heuristics for the permutation flow shop problem. *Computers & Operations Research* 37-12, 2062-2070.

Robinson, S., 1997. Simulation model verification and validation: increasing the user's confidence. In: *Proceedings of the 1997 Winter Simulation Conference*, pp. 53-59.

Ruiz, R., Maroto, C., 2005. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research* 165, 479-494.

Ruiz, R., Maroto, C., Alcaraz, J., 2006. Two new robust genetic algorithms for the flowshop scheduling problem. *Omega-International Journal of Management Science* 34, 461-476.

Ruiz, R., Stützle, T, 2007. A simple and effective iterated greedy algorithm for the permutation flow-shop scheduling problem. *European Journal of Operational Research* 177, 2033-2049.

Stützle, T., 1998. Applying Iterated Local Search to the Permutation Flow Shop Problem. Available at: http://iridia.ulb.ac.be/~stuetzle/publications/AIDA-98-04.pdf

Suliman, S., 2000. A two-phase heuristic approach to the permutation flow-shop scheduling problem. *International Journal of Production Economics* 65(1-3), 143-152.

Taillard, E., 1990. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research* 47, 65-74.

Taillard, E., 1993. Benchmarks for basic scheduling problems. *European Journal of Operations Research* 64, 278-285.

Talbi, E., 2009. *Metaheuristics: From Design to Implementation*. Wiley.

Widmer, M., Hertz, A., 1989. A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research* 41(2), 186-193.

Zheng, T., Yamashiro, M., 2010. Solving flow shop scheduling problems by quantum differential evolutionary algorithm. *International Journal of Advanced Manufacturing Technology* 49, 643-662.

Zobolas, C., Tarantilis, C., Ioannou, G., 2009. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research* 36, 1249-1267.