ILS-ESP: AN EFFICIENT, SIMPLE, AND PARAMETER-FREE ALGORITHM FOR SOLVING THE PERMUTATION FLOW-SHOP PROBLEM

Angel A. Juan[a,*], Helena R. Lourenço[b], Manuel Mateo[c], Quim Castellà[a], Barry B. Barrios[a]

[a] *IN3 - Computer Science Dep., Open University of Catalonia, Rambla Poblenou, 156, 08018 Barcelona, Spain*
[b] *Dep. of Economics and Business, Universitat Pompeu Fabra, R. Trias Fargas, 25-27 08005 Barcelona, Spain*
[c] *Dep. de Organización de Empresas, Universidad Politécnica de Cataluña, Av. Diagonal, 647 08028 Barcelona, Spain*

Abstract

From a managerial point of view, the more efficient, simple, and parameter-free (ESP) an algorithm is, the more likely it will be used in practice for solving real-life problems. Following this principle, an ESP algorithm for solving the Permutation Flowshop Sequencing Problem (PFSP) is proposed in this article. Using an Iterated Local Search (ILS) framework, the so-called ILS-ESP algorithm is able to compete in performance with other well-known ILS-based approaches, which are considered among the most efficient algorithms for the PFSP. However, while other similar approaches still employ several parameters that can affect their performance if not properly chosen, our algorithm does not require any particular fine-tuning process since it uses basic 'common sense' rules for the local search, perturbation, and acceptance criterion stages of the ILS metaheuristic. Our approach defines a new operator for the ILS perturbation process, a new acceptance criterion based on extremely simple and transparent rules, and a biased randomization process of the initial solution to randomly generate different alternative initial solutions of similar quality -which is attained by applying a biased randomization to a classical PFSP heuristic. This diversification of the initial solution aims at avoiding poorly designed starting points and, thus, allows the methodology to take advantage of current trends in parallel and distributed computing. A set of extensive tests, based on literature benchmarks, has been carried out in order to validate our algorithm and compare it against other approaches. These tests show that our parameter-free algorithm is able to compete with state-of-the-art metaheuristics for the PFSP. Also, the experiments show that, when using parallel computing, it is possible to improve the top ILS-based metaheuristic by just incorporating to it our biased randomization process with a high-quality pseudo-random number generator.

Keywords: Flow-Shop Problem, Scheduling, Randomized Algorithms, Iterated Local Search, Metaheuristics, GRASP

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. Every job $j \in J$ requires a given fixed processing time $p_{ij} \geq 0$ on every machine $i \in M$. Each machine can execute at most one job at a time, and it is assumed that all the 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 a 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 which, in general, is NP-complete [38]. Similar to what has happened 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 the use of heuristics and metaheuristics that provide near-optimal solutions for medium and large-sized problems [40]. Most of these methods focus on minimizing makespan. Some of them have reached outstanding efficiency levels, but they are still using several parameters that usually require non-trivial and time-costly fine-tuning processes. As some authors recognize (e.g. [49, 11, 48, 27, 1, 17, 8]), the proper selection of these parameters and their specific values have a significant impact on the algorithm's performance, i.e.: the efficiency of these methods tend to be quite sensitive to the values assigned to each parameter. For instance, while describing a new Genetic Algorithm (GA) for solving the hybrid flow shop problem, Engin et al. [11] stated: "It is well known that the GAs' efficiency depends to a high degree upon the selection of the control parameters. The determination of the suitable settings for the control parameters of any GA is a very difficult task. The GA's search process is controlled with multiple factors (control parameters) whose effects will possibly interact with each other". Thus, statisti-

*Corresponding author
Email addresses: ajuanp@gmail.com (Angel A. Juan), helena.ramalhinho@upf.edu (Helena R. Lourenço), manel.mateo@upc.edu (Manuel Mateo), quim.castella@gmail.com (Quim Castellà), barry.brian.barrios@gmail.com (Barry B. Barrios)

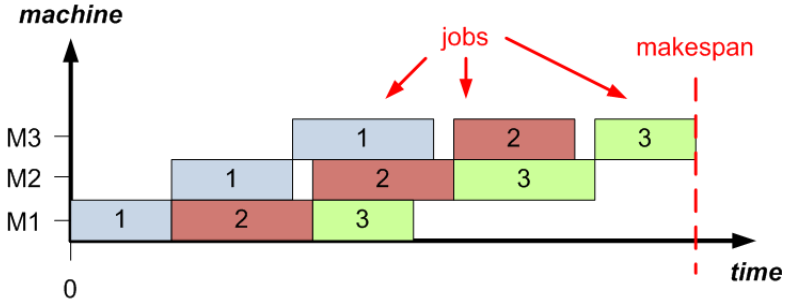

Figure 1: Flowshop Sequencing Problem.

cal techniques such as the design of experiments (DOE) need to be carried out before obtaining an efficient algorithm. Moreover, even when the parameter configuration process has been successful, it is frequent to find 'magic numbers' inside the final algorithm. Those numbers are obtained by experimentation, and they correspond to the specific values of the parameters that provide the best possible algorithm performance. However, for decision-makers, who are not usually experts in metaheuristics, it is difficult to understand the real meaning of such particular values and, therefore, the algorithm just becomes a black-box to them. This lack of transparency reduces the method's credibility and, consequently, makes the algorithm less interesting for real-life applications [39]. This relationship between simplicity and applicability has also been noticed by other authors. For instance, Zobolas et al. [49] refer to their algorithm in the following terms: "...it requires very few user-defined parameters, rendering it applicable to real-life flow shop scheduling problems".

For that reason, the main contribution of this article is the design of an efficient, simple, and parameter-free (ESP) algorithm for solving the PFSP. The interest in developing parameter-free algorithms for dealing with different combinatorial optimization problems is not new. For example, Matsui and Yamada [27] developed a parameter-free GA for solving the job-shop scheduling problem. As we will discuss in more detail, our algorithm is based on an Iterated Local Search (ILS) framework [26], and it does not require any particular fine-tuning process since it 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 'magic number' or complex operators. Also, instead of using a traditional Simulated Annealing-type acceptance criterion, which performs very well but implies some kind of indirect behavior, a new and more transparent rule is defined for this stage. Another important characteristic of our approach is the biased randomization process of the initial solution, which employs a decreasing triangular distribution to randomly generate different alternative initial solutions based on the well-known heuristic from Nawaz, Enscore and Ham (NEH) [30]. 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 is similar to the one proposed in Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristics [36].

The paper is organized as follows: Section 2 offers a basic literature review on the flow-shop problem. Sections 3 briefly describes the main ideas characterizing GRASP and ILS metaheuristics, since our approach could be considered a hybridization of both types of algorithms. Section 4 describes our method in detail. Sections 5 and 6 discuss two extensive computational experiments, which have been carried out to test the efficiency of our algorithm and compare it against state-of-the-art approaches. 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, mainly because of the difficulty encountered by exact methods to solve medium or large instances. Most existing approaches focus on the objective of minimizing makespan. Johnson [19] proposed a simple procedure to obtain optimal sequences for the PFSP with two machines and three machines. Campbell et al. [4] developed the so-called CDS heuristic for solving the PFSP with more than two machines. Dannenbring [10] also proposed several constructive and improvement heuristics for the general problem. Nawaz et al. [30] introduced the NEH heuristic, which is commonly considered as the best performing heuristic for the PFSP. Basically, what the NEH heuristic proposes is to calculate 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 to create an 'efficiency list' of jobs sorted in a 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, a job is inserted in the sorted set of jobs that are configuring the ongoing solution. The exact position that the selected job will occupy in that ongoing solution is given by the minimum makespan criterion following a 'shift-to-left' movement. Taillard [45] introduced

a data structure that reduces the NEH complexity. Other interesting heuristics are those from Suliman [44] or Framinan and Leisten [16], which also consider several extensions of NEH when facing other objectives than makespan. The NEH heuristic is the commonly accepted method for the permutation flowshop problem under the makespan minimization criterion, and it has been frequently used to provide an initial upper bound for the best branch-and-bound algorithms [25, 7].

Different metaheuristic approaches have also been proposed for the PFSP. Osman and Potts [31] used Simulated Annealing (SA). Widmer and Hertz [47] proposed a Tabu Search (TS) algorithm known as SPIRIT. Other Tabu Search algorithms also make use of the NEH heuristic [45, 34, 28]. Also, Genetic Algorithms based on the NEH heuristic have been proposed for solving the PFSP [6, 35, 2]. Other metaheuristics, such as Ant Colony Optimization, have been used to obtain optimal solutions [5]. The efficiency of the previous procedures can be checked looking at the best known solutions for the Taillard's benchmark instances [46].

What all the aforementioned work has 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 [40]. Additionally, there are other highly elaborated hybrid techniques for solving the PFSP. However, as Ruiz and Stützle [42] 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 [15, 18, 40].

In this paper, however, we will mainly focus on ILS-related approaches. The ILS metaheuristic has been applied successfully to a variety of combinatorial optimization problems. In some 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. [3], 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 probably due to two factors: (i) the simple yet powerful exploration/explotation 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 [26]. Several versions of the ILS metaheuristic have been applied to solve the PFSP. In particular, Stützle [43] proposed a simple yet efficient ILS approach for this problem. Some years later, Ruiz and Stützle [42] 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 probably the best algorithm developed so far for the PFSP. In [42], Ruiz and Stützle 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 [32, 3, 37]. However, to the best of our knowledge, IG has been performing 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 PSPF field and, therefore, in this paper we compare our approach against both the ILS and IG algorithms. Thus, for instance, in [49] 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. [41]. However, as Ruiz and Stützle show in [42], IG is simpler and far superior to HGA-RMA. Also, Ribas et al. [37] recently proposed several SA-based algorithms to compete with IG, but their results show that IG uses less parameters and performs better than their algorithms in all classical benchmarks. Finally, Nagano et al. [29] tested their GA approach against a number of GA-based approaches. Table 3 in their paper is directly comparable to Table 3 in [42] (shared authors, same CPU, same maximum computing times, etc.). A comparison of both tables shows that IG outperforms all considered metaheuristics.

Being relatively simple and yet extremely efficient, IG has inspired other approaches for different combinatorial optimization problems. For instance, Kahraman et al. [23] recently developed a parallel greedy algorithm for the hybrid flow shop scheduling problem which uses the IG's destruction-construction operator. Another example is that of Pan et al. [32], who developed an IG-based algorithm for the no-wait flow shop scheduling problem.

3. OVERVIEW OF THE GRASP AND ILS METAHEURISTICS

Since the approach presented in this paper has some similarities with GRASP and ILS metaheuristics -and to some extend it can be seen as a hybridization of both-, these two families of algorithms are briefly described next. For a recent review of the GRASP and ILS methodologies the reader is referred to [13, 14, 36] and [26] , respectively.

GRASP is a multi-start method designed to solve hard combinatorial optimization problems [12]. 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. The two phases are repeated until a stopping criterion is reached, keeping track of the best solution found overall the search. The constructive phase builds step by step 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 subject to a randomization process. In 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 [33]. In this version of the methodology, the parameter is not fixed along the running of the algorithm, but it

```
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 stage
06    aSol = localSearch(aSol)  // local search stage
07    bestSol = updateBestSol(aSol)
08    baseSol = acceptanceCriterion(baseSol, aSol, history) // accept. stage
09  end while

10  return bestSol

end
```

Figure 2: ILS general framework

is 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 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.

In this work, we propose an ILS-based algorithm with innovative components and original combinations among them. Moreover, our method, which will be described later in detail, introduces some GRASP principles inside the initial solution generation process of the ILS framework.

The main motivation to develop an ILS-based approach for the PFSP is because this method has many of the desirable features of a metaheuristic as described by Cordeau et al. [9]: *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 to be implemented. This is an important feature since the method can be applied to different instances 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 with the adaptation to other similar problems.

In the next section, our ILS-based parameter-free method is presented. By combining ILS with GRASP we try to balance, in an efficient way, the four attributes described above, i.e.: *accuracy*, *speed*, *simplicity* and *flexibility*. Regarding the GRASP part, it can also be related to Monte Carlo simulation or, simply, random sampling from a non-uniform distribution. In some recent works, Juan et al. [22, 21] described the application of simulation-based techniques to solve Vehicle Routing Problems (VRP). They 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 that make the algorithm significantly different from previous ILS-based algorithms as those described in [43, 42].

```
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 3: enhancedSwap procedure to perform the ILS-ESP perturbation stage.

```
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 4: pseudocode for the ILS-ESP acceptance criterion stage.

The first of these three critical points is related with 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 do the following: (a) randomly selects (using a uniform distribution) two different jobs from the current solution; (b) interchanges both jobs, that is, interchange 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 those 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 to all elements to its left. Then we shift the right-most job of this subset and tentatively insert it into all possible positions of the sequence of jobs in this subset. Next, we select the one that results in the minimum makespan. Afterwards, we take this subset and reinsert the other sequences that were taken out. We then apply this idea again for the other swapped job. This 'shift-to-left' movement takes advantage of Taillard accelerations [45] to quickly determine which is the best position for each job when only the partial solution up to its position is considered. Figure 3 shows the pseudo-code associated with this perturbation operator. Notice that the proposed operator is really simple and it does not use any specific-value parameter that needs any complex fine-tuning process.

A second critical point of our algorithm is the *acceptation criterion*. The algorithm does not use a SA-based process -like most other ILS-based algorithms do-, but instead it uses a simplified version of a Demon-like process. This criterion is designed to contribute, together with the perturbation process, to avoid local minima during the algorithm execution. In order to do so, the criterion simply states the following basic principles: (a) anytime 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 also against the best-known solution, *bestSol*, to see if it has to be updated too; and (b) even in the case that a newly generated solution is worse than the base solution, the base solution will be updated (deteriorated) to this new solution as far 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 probabilities that the algorithm gets trapped into a local minimum are highly reduced. Figure 4 shows the pseudo-code associated with this acceptance criterion process. Again, note that this is attained with a very simple set of basic rules and without any specific-value parameter.

A third critical point of our approach, and probably the most innovative one, is related to the starting solution employed inside the ILS framework, *generate initial solution*. Usually, this starting solution is the one provided by the NEH heuristic,

```
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 5: RandNEH procedure to perform the NEH biased randomization.

which produces a relatively good initial solution in most cases. Using the NEH solution instead of a randomly generated solution is typically considered good practice in order to accelerate the algorithm's convergence. 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 mostly to the number of movements or transformations that must be applied to the initial solution in order to arrive at a pseudo-optimal solution. Since we are especially interested in running multiple iterations of any given instance, which can be seen as a form of biased GRASP, we designed a way to generate different randomized NEH solutions with similar properties. We use the decreasing triangular distribution, i.e. the triangular probability distribution with coincident lower limit and mode values. This biased distribution was used to randomize the NEH heuristic in a similar way Juan et al. [21] used the geometric distribution to randomize another classical heuristic for the Vehicle Routing Problem. As described before, the NEH heuristic is an iterative algorithm which employs 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 which is 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 of 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. Finally, the selection process should be done without introducing any parameters in the methodology -otherwise, it would be necessary to perform fine-tuning processes, which tend to be non-trivial and time-consuming. Figure 5 shows the main pseudo-code associated with this process.

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. 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 which makespan is almost equal or even better than the original NEH solution. 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.

Regarding the *local search* process that our algorithm uses, it is the same simple and intuitive process used by Ruiz and Stützle in [42]. Therefore, we refer the reader to their paper for more details.

Figure 6 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. When comparing the ILS-ESP with other ILS approaches successfully applied to the PFSP, we can notice that

```
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 6: ILS-ESP general procedure.

all these methods share a general ILS structure as well as the local search component. However, ILS-ESP significantly differs from other ILS-based approaches in the following components and operators: (i) the "enhanced-swap" perturbation operator, which is parameter-free; (ii) the Demon-based (also parameter-free) acceptance criteria; and (iii) the biased randomization ot the NEH heuristic in order to generate alternative initial solutions of similar quality.

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 than the complexity of the NEH heuristic, which is employed in most modern metaheuristics to generate an initial solution. Using Taillard's accelerations, its relative speed is $O(n^2m)$ [45]. As described before, the local search procedure is the same classical process employed in other similar ILS-based approaches. The complexity of this local search is $O(n^2m)$. Although it is iteratively applied until the current solution cannot be improved any further, the number of iterations in a row 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 stoping criterion we use in practice is to limit 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, and according to the computational experiments we will discuss in this paper, it is worthy to notice that a fast and 'high-quality' (pseudo-) Random Number Generator (RNG) can enhance somewhat the performance of some ILS-based algorithms. In particular, we have observed that employing a 'good' RNG [24] instead of the RNG natively provided by the programming language itself, seems to have some positive impact on the overall results of those algorithms including a SA-like acceptance criterion, where an intensive use of uniform pseudo-random numbers in the interval $(0, 1)$ is required.

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

The parameter-free algorithm described in this paper, the *ILS-ESP*, 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, being one of the simplest object-oriented languages, it facilitates the rapid development of prototypes. The expected counterpart is that, since Java code runs over a virtual machine, a Java version of an algorithm will probably 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 param-

eters D and T) were also codified in Java by the same programmers:

- The *ILS98-T04*, which is the algorithm proposed in [43] using $T = 0.4$. According to the algorithm's author, this parameter value is the one offering the best algorithm's 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 [42]. The first set of parameters were obtained by the IG's authors after completing a DOE fine-tuning process. The second set of parameters have been selected by us in order to show how IG's performance can be affected if other parameter values are selected instead of the 'optimal' ones. At this point, it is worthy to remember that in multiple experiments carried out by different authors [42, 49, 37], the IG-D4T04 algorithm has outperformed any other algorithm so far, including GA and TS with more parameters than IG. Thus, the IG algorithm is highly cited in the PFSP literature and, to the best of our knowledge, it is probably the most efficient algorithm 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 to the IG algorithm. This version also employs a high-quality pseudo-random number generator [24].

As briefly mentioned in the previous section, 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 it will be shown in the experimental section, this can be especially useful when multiple instances of the algorithm are run in parallel.

- The new parameter-free 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 'magic number' ($D = 4$) which determines how many jobs must be extracted during the destruction phase. This value is considered to be independent of the problem size, which is somewhat surprising (our guess is that the authors fixed this value to avoid introducing more complexity in the algorithm's fine-tuning process).

- A transparent and parameter-free acceptance criterion component, which is based on a Demon-like process. This 'white-box' approach substitutes the 'black-box' SA-like approach employed in other ILS-based algorithms, which also uses an additional parameter ($T = 0.4$).

For each one of the aforementioned algorithms, we designed and performed extensive tests -using the same machine, same language program, same execution time, and same programmer developer- on the 120 Taillard's benchmark instances [46]. 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, 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 [49]. 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. [49]: "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, it can be derived 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 setting, 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 parameter-free 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 the later 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.

An interesting phenomenon that we have observed during the computer experiments is that the use of a 'high-quality'

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)
20x5	0.00%	0.04%	0.00%	0.04%	0.00%	1
20x10	0.00%	0.00%	0.00%	0.00%	0.04%	2
20x20	0.00%	0.01%	0.00%	0.00%	0.03%	4
50x5	0.00%	0.00%	0.00%	0.00%	0.00%	2.5
50x10	0.38%	0.48%	0.47%	0.45%	0.53%	5
50x20	0.62%	0.66%	0.71%	0.74%	1.00%	10
100x5	0.00%	0.01%	0.00%	0.01%	0.00%	5
100x10	0.10%	0.07%	0.10%	0.07%	0.10%	10
100x20	0.94%	1.04%	1.13%	1.07%	1.25%	20
200x10	0.08%	0.08%	0.09%	0.11%	0.14%	20
200x20	1.18%	1.29%	1.24%	1.32%	1.48%	40
500x20	0.62%	0.63%	0.63%	0.67%	0.71%	100
Averages	0.33%	0.36%	0.36%	0.37%	0.44%	--

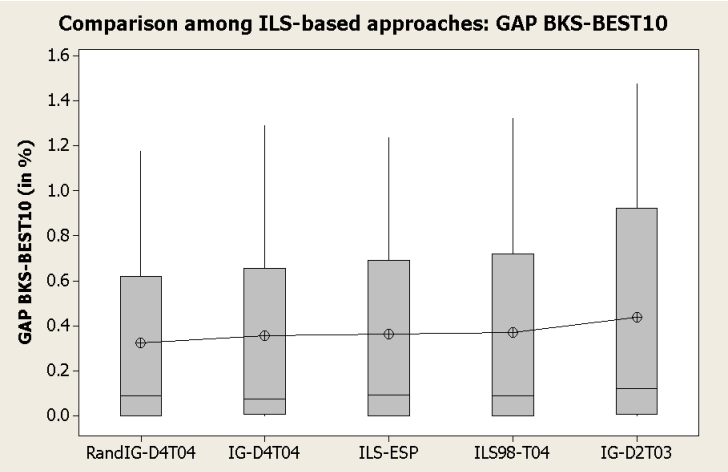

Figure 7: Multiple box-plot for the BEST10 metric.

pseudo-random number generator (RNG) seems to have a positive effect only on those ILS-based algorithms which incorporate an SA-like acceptation criterion. However, the use of a high-quality RNG does not seem to have a perceptive impact on those ILS-based algorithms which do not incorporate an SA-like stage. Our hypothesis here is that the SA-like process is more sensible to the use of a high-quality RNG because it employs continuous random variables. On the contrary, the remaining random processes are basically associated with discrete variables -selection of job positions- and, therefore, they do not seem to be too much sensible with respect to the RNG quality.

Figure 7 shows a multiple-boxplot which allows for a visual comparison of the algorithms' performance. This Figure reinforces the idea that results from the first four approaches are quite equivalent, although maybe the RandIG-D4T04 is performing slightly better than the rest. It seems also clear from this Figure that the non-optimized version of the IG algorithm performs slightly worse than the rest.

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 seems not to be 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 worthy 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 effect, most current workstations include multi-core processors and, therefore, different instances of randomized algorithms -like the ones compared in this paper- can be run in parallel even in a single machine after conveniently setting the initial

Table 2: Gaps between Best Known Solution and *AVG*10. 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)
20x5	0.04%	0.04%	0.05%	0.05%	0.07%	1
20x10	0.05%	0.05%	0.06%	0.05%	0.17%	2
20x20	0.04%	0.05%	0.06%	0.05%	0.17%	4
50x5	0.00%	0.01%	0.01%	0.01%	0.02%	2.5
50x10	0.71%	0.69%	0.78%	0.69%	0.84%	5
50x20	1.02%	1.04%	1.08%	1.11%	1.48%	10
100x5	0.02%	0.01%	0.02%	0.04%	0.03%	5
100x10	0.28%	0.27%	0.32%	0.31%	0.37%	10
100x20	1.46%	1.47%	1.55%	1.46%	1.68%	20
200x10	0.23%	0.23%	0.26%	0.26%	0.28%	20
200x20	1.57%	1.52%	1.57%	1.57%	1.73%	40
500x20	0.78%	0.80%	0.79%	0.83%	0.84%	100
Averages	0.52%	0.52%	0.55%	0.54%	0.64%	--

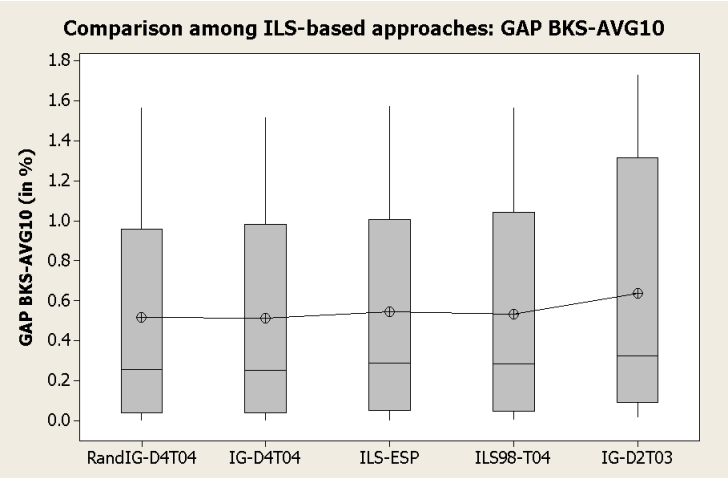

Figure 8: Multiple box-plot for the AVG10 metric.

seeds for the RNG and without increasing the total clock time employed. Of course, by using a computer cluster instead of a single machine, even much more instances of a randomized algorithm can be run for a given clock time. Traditionally, however, the most usual metric to compare sequential algorithms in the PFSP literature is the one referred to 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.

From the last row in the table, it can be derived 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 beyond (average gaps = 0.55% and 0.54% respectively). Lower results are obtained for the non-optimally parameterized IG-D2T03 (average gap = 0.64%). Again, the later result seems to imply that performance of the IG algorithm greatly depends on the proper selection of its parameters.

Figure 8 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 are performing 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.

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 overlap-

Table 3: Gaps between Best Known Solution and $BEST(n)$ for Taillard 50x20. Java code running on an Intel Xeon at 2.0 GHz.

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

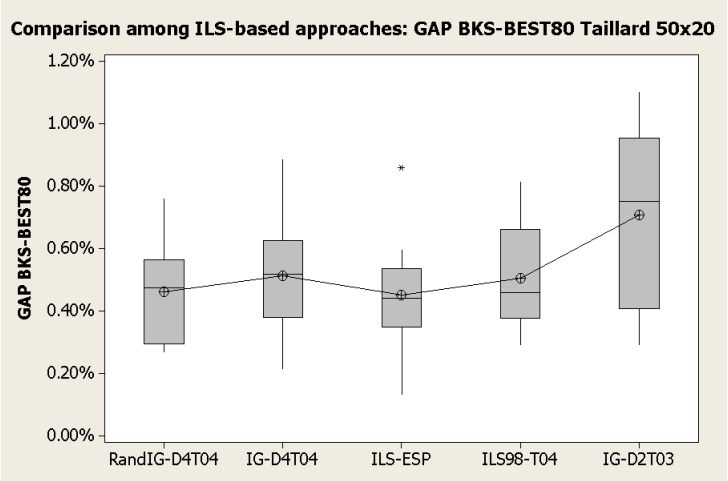

Figure 9: Multiple box-plot for the BEST80 metric.

ping 95% confidence intervals, no statistically significant difference has been found among the various algorithms' means. Again, the normality assumption seems not to be 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 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 it was noticed 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 the fact that they are "black-boxes" to non-experts -and thus, there is a lack of credibility. 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, more likely to be used in real-life scenarios. Finally, regarding 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 PARALLELIZATION OVER THE ILS-ESP

As it has been previously noticed, the algorithm proposed here can be easily parallelized by splitting the random-number-generation sequence in 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. In spite of this, 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 [20]. There are new processor design paradigms based on gaining computation capacity through the parallel execution of multiple processes and threads (multi-core). Following this concept, new, affordable Graphic Processing Units (GPUs) have recently been introduced on the market that offer the capacity of executing hundreds -or even thousands- of threads concurrently.

Table 4: Gaps between Best Known Solution and $AVG(n)$ for Taillard $50x20$. Java code running on an Intel Xeon at 2.0 GHz.

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

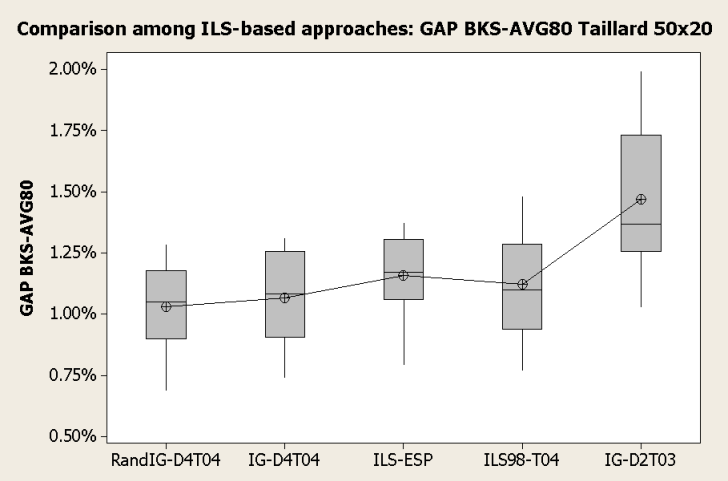

Figure 10: Multiple box-plot for the AVG80 metric.

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 as 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, being 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 contribute to 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 on 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. As before, the termination condition of each execution or replica is given by an instance-size dependent maximum time $t_{max} = 0.01\text{s} \times k \times m$. 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 9 show the results obtained for the BEST(n) metric.

Notice how 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 from 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 10 show the results obtained for the AVG(n) 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 show the existence of significant differences among the different al-

gorithms ($p - value = 0.001$).

7. CONCLUSION

In this paper an efficient, simple, and parameter-free algorithm for solving the Permutation Flowshop Sequencing Problem (PFSP) has been presented. The ILS-ESP algorithm, which is based on an embarrassingly parallel iterated local search framework, is able to compete with state-of-the-art metaheuristics. These other metaheuristics typically use a deterministic initial solution and contain several parameters. The proposed ILS-ESP method has the four desirable features of a good metaheuristic: *accuracy*, *speed*, *simplicity*, and *flexibility*. In order to develop our approach, we have designed a new perturbation operator (enhanced swap), a new demon-based acceptation criterion, and a biased randomization process of the well-known NEH heuristic.

Another contribution of this paper is the incorporation of the aforementioned randomization process to an existing ILS-based algorithm containing an SA-like acceptance criterion. According to the computational tests performed, the performance of the parameterized ILS-based algorithm can be enhanced by simply adding our GRASP-inspired randomization process combined with the use of a high-quality pseudo-random generator.

All in all, either our randomized version of the parameterized ILS or the proposed parameter-free algorithm seem to be excellent alternatives for solving the PFSP.

ACKNOWLEDGMENTS This work has been partially supported by the Spanish Ministry of Science and Innovation (grants TRA2010-21644-C03, ECO2009-11307, and DPI2007-61371), by the Catalan Department of Universities, Research & Information Society (grant 2009 CTP 00007) and by the CYTED-IN3-HAROSA Network (http://dpcs.uoc.edu).

References

[1] C. Alabas-Uslu, B. Dengiz, A self-adaptive local search algorithm for the classical vehicle routing problem, Expert Systems and Applications 38 (2011) 89908998.

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

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

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

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

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

[7] Companys, R. and M. Mateo, 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.

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

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

[10] D.G. Dannenbring, An evaluation of flowshop sequence heuristics, Management Science 23 (1977) 1174-1182.

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

[12] T.A. Feo, M.G.C. Resende, Greedy randomized adaptive search procedures, Journal of Global Optimization 6 (1995) 109-133.

[13] P. Festa, M.G.C. Resende, An annotated bibliography of GRASP Part I: algorithms, Int Trans Opl Res 16 (2009) 1-24.

[14] P. Festa, M.G.C. Resende, An annotated bibliography of GRASP Part II: applications, Int Trans Opl Res 16 (2009) 131-172.

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

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

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

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

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

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

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

[22] A. Juan, J. Faulin, R. Ruiz, B. Barrios, M. Gilibert, X. Vilajosana, Using oriented random search to provide a set of alternative solutions to the capacitated vehicle routing problem, in: Operations Research and Cyber-Infrastructure. Springer Operations Research/Computer Science Interfaces Series, vol. 47, 2009, pp. 331-346.

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

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

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

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

[27] S. Matsui, S. Yamada, 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, 2007, pp. 3796-3803.

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

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

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

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

[32] Q.K. Pan, L. Wang B.H. Zhao, 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) (2008)

778-786.
[33] M. Prais, C.C. Ribeiro, Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment, INFORMS Journal on Computing 12 (2000) 164-176.
[34] C.R. Reeves, Improving the efficiency of tabu search for machine scheduling problems, Journal of the Operational Research Society 44(4) (1993) 375-382.
[35] C.R. Reeves, A genetic algorithm for flowshop sequencing, Computers and Operations Research 22(1) (1995) 5-13.
[36] M.G.C. Resende, C.C. Ribeiro, GRASP: Greedy Randomized Adaptive Search Procedures, in: Search Methodologies, Springer.
[37] I. Ribas, R. Companys, X. Tort-Martorell, Comparing three-step heuristics for the permutation flow shop problem, Computers & Operations Research 37-12 (2010) 2062-2070.
[38] A.H.G. Rinnooy Kan, Machine Scheduling Problems: Classification, Complexity and Computations, Springer, 1976.
[39] S. Robinson, Simulation model verification and validation: increasing the users confidence, in: Proceedings of the 1997 Winter Simulation Conference, 1997, pp. 53-59.
[40] R. Ruiz, C. Maroto, A comprehensive review and evaluation of permutation flowshop heuristics, European Journal of Operational Research 165 (2005) 479-494.
[41] R. Ruiz, C. Maroto, J. Alcaraz, Two new robust genetic algorithms for the flowshop scheduling problem, Omega-International Journal of Management Science 34 (2006) 461-476.
[42] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flow-shop scheduling problem, European Journal of Operational Research 177 (2007) 2033-2049.
[43] T. Stützle, Applying Iterated Local Search to the Permutation Flow Shop Problem, 1998. Available at: `http://iridia.ulb.ac.be/~stuetzle/publications/AIDA-98-04.pdf`
[44] S. Suliman, A two-phase heuristic approach to the permutation flow-shop scheduling problem, International Journal of Production Economics 65(1-3) (2000) 143-152.
[45] E. Taillard, Some efficient heuristic methods for the flow shop sequencing problem, European Journal of Operational Research 47 (1990) 65-74.
[46] E. Taillard, Benchmarks for basic scheduling problems, European Journal of Operations Research 64 (1993) 278-285.
[47] M. Widmer, A. Hertz, A new heuristic method for the flow shop sequencing problem, European Journal of Operational Research 41(2) (1989) 186-193.
[48] T. Zheng, M. Yamashiro, Solving flow shop scheduling problems by quantum differential evolutionary algorithm, Int. J. Adv. Manuf. Technol., 49 (2010) 643-662.
[49] C. Zobolas, C. Tarantilis, G. Ioannou, Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm, Computers & Operations Research 36 (2009) 1249-1267.