Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2016, Article ID 9598041, 15 pages
http://dx.doi.org/10.1155/2016/9598041

Research Article

ILS Heuristics for the Single-Machine Scheduling Problem with Sequence-Dependent Family Setup Times to Minimize Total Tardiness

Vinícius Vilar Jacob and José Elias C. Arroyo

Department of Computer Science, Universidade Federal de Viçosa, 36570-900 Viçosa, MG, Brazil

Correspondence should be addressed to José Elias C. Arroyo; jarroyo@dpi.ufv.br

Received 19 August 2016; Revised 15 September 2016; Accepted 19 September 2016

Academic Editor: Quanke Pan

This paper addresses a single-machine scheduling problem with sequence-dependent family setup times. In this problem the jobs are classified into families according to their similarity characteristics. Setup times are required on each occasion when the machine switches from processing jobs in one family to jobs in another family. The performance measure to be minimized is the total tardiness with respect to the given due dates of the jobs. The problem is classified as $\mathcal{NP}$-hard in the ordinary sense. Since the computational complexity associated with the mathematical formulation of the problem makes it difficult for optimization solvers to deal with large-sized instances in reasonable solution time, efficient heuristic algorithms are needed to obtain near-optimal solutions. In this work we propose three heuristics based on the Iterated Local Search (ILS) metaheuristic. The first heuristic is a basic ILS, the second uses a dynamic perturbation size, and the third uses a Path Relinking (PR) technique as an intensification strategy. We carry out comprehensive computational and statistical experiments in order to analyze the performance of the proposed heuristics. The computational experiments show that the ILS heuristics outperform a genetic algorithm proposed in the literature. The ILS heuristic with dynamic perturbation size and PR intensification has a superior performance compared to other heuristics.

1. Introduction

Scheduling is a very important decision-making process that occurs in manufacturing systems. Scheduling problems deal with the allocation of resources to jobs over given time periods and its goal is to optimize one or more performance measures. This type of problems has been thoroughly studied since the mid-1950s [1]. Nowadays, scheduling problems are one of the most studied problems because they have great practical and theoretical importance. These problems have many applications in several industries (like chemical, metallurgic, and textile) and most of these problems belong to the class of $\mathcal{NP}$-hard problems.

The scheduling problem focused on in this paper is stated as follows. There is a set of n jobs to be processed on a single machine without interruption or preemption. Jobs are classified into F families and are available at time zero. Each family l has n_l jobs ($l = 1, \ldots, F$), such that $n_1 + n_2 + \cdots + n_F = n$. $f(j)$ denotes the family of job j. The processing time (p_j) and due date (d_j) of job j are previously known. There is a family setup time $s_{ij} > 0$ between jobs i and j if job j is processed immediately after job i and $f(i) \neq f(j)$. If jobs i and j belong to the same family ($f(i) = f(j)$), $s_{ij} = 0$. The setup times are sequence-dependent; that is, s_{ij} may not be equal to s_{ji}, $\forall i, j$, $f(i) \neq f(j)$.

Setup time is the time required to prepare the necessary resource (machines) to perform a task (operation or job) [2]. Setup operations include obtaining tools, positioning work in process material, return tooling, cleanup, setting the required jigs and fixtures, adjusting tools, and inspecting material [1]. In the problem under study we do not consider the group technology (GT) assumption; that is, a family of jobs is not necessarily processed as a single batch [3]. Therefore a family of jobs could be divided into multiple nonconsecutive batches in an optimal sequence and each of the batches incurs a setup time.

The goal of the problem is to find a production schedule (sequence of jobs) that minimizes the total tardiness with respect to the given due dates of the jobs. The tardiness of a job j (defined by T_j) is defined as the completion time of the job (C_j) minus the due date for the job if the job is completed after the due date, and the tardiness is equal to zero if the job is completed before the due date. T_j can be expressed as $T_j = \max(C_j - d_j, 0)$. If job j is processed immediately after job i, then $C_j = C_i + p_j$ if $f(i) = f(j)$, and $C_j = C_i + s_{ij} + p_j$ if $f(i) \neq f(j)$. The total tardiness of jobs (objective function) is computed as

$$\mathrm{TT} = \sum_{j=1}^{n} T_j. \tag{1}$$

The single-machine total tardiness (SMTT) problem with sequence-dependent family setup times is denoted by $1|\mathrm{ST}_{\mathrm{sd},f}|\sum T_j$ following the three-field notation $\alpha|\beta|\gamma$ presented by Graham et al. [4] and adapted by Allahverdi et al. [1], where 1 is the single-machine environment, $\mathrm{ST}_{\mathrm{sd},f}$ is information of sequence-dependent family or batch setup time, and $\sum T_j$ is the total tardiness objective function.

The total tardiness criterion is very important in manufacture systems because several costs may exist when a job is delivered with tardiness. Among these the following can be quoted: contractual penalties, loss of credibility resulting in a high probability of losing a client for some or for all the possible futures jobs, and damage in the company's reputation that may distance other clients.

Since the SMTT problem without sequence-dependent setup times is a binary $\mathcal{NP}$-hard problem [5], it follows that the problem $1|\mathrm{ST}_{\mathrm{sd},f}|\sum T_j$ considered in this paper is at least $\mathcal{NP}$-hard in the ordinary sense [6].

Scheduling problems that consider explicitly setup times are of great importance in manufacturing systems. Extensive literature reviews for these problems, considering different shop environments, were presented by [1, 2]. The SMTT problem with sequence-dependent setup times (without considering the grouping of jobs into families), denoted by $1|\mathrm{ST}_{\mathrm{sd}}|\sum T_j$, is one of the most researched topics in the scheduling literature. For this problem there are many studies on metaheuristics such as genetic algorithms [7–10], Simulated Annealing [11], Ant Colony Optimization [12, 13], Greedy Randomized Adaptive Search Procedure [14, 15], Iterated Local Search [16], and iterated greedy [17]. There are also some studies on exact algorithms for $1|\mathrm{ST}_{\mathrm{sd}}|\sum T_j$ [18–21].

Some studies were realized on single-machine scheduling problems with family consideration and sequence-independent family setup times ($\mathrm{ST}_{si,f}$). To better understand $\mathrm{ST}_{si,f}$, let us suppose that the jobs i, j, and k belong to different families. If job i (or j) is processed immediately before job k, then $C_k = C_i + s_k + p_k$ (or $C_k = C_j + s_k + p_k$), where s_k is the considered sequence-independent setup time. Note that the setup time s_k does not depend on the family $f(i)$ (or $f(j)$) previously processed; it only depends on family $f(k)$. Kacem [22] addressed the $1|\mathrm{ST}_{si,f}|\sum T_j$ problem and proposed a set of methods to obtain lower bounds for the optimal total tardiness. Schaller [23] developed Branch and Bound and heuristic algorithms for the $1|\mathrm{ST}_{si,f}|\sum T_j$ problem. Exact methods were used by Pan and Su [24] and Baker and Magazine [25] to solve the $1|\mathrm{ST}_{si,f}|L_{\max}$ problem, where $L_{\max}$ is the maximum lateness. To minimize the total earliness and tardiness cost ($\sum E_j + \sum T_j$), exact and heuristic methods were developed by Schaller and Gupta [26]. Gupta and Chantaravarapan [27] considered the $1|\mathrm{ST}_{si,f}|\sum T_j$ problem where jobs in each family are processed together; that is, the group technology assumption is considered. These authors provided a mixed-integer linear programming (MILP) model capable of solving small-sized problems.

Single-machine scheduling problems involving sequence-dependent family setup times have been much less studied [2]. van der Veen et al. [28] addressed the $1|\mathrm{ST}_{\mathrm{sd},f}|C_{\max}$ problem, where $C_{\max}$ is the maximum completion time (makespan). These authors proposed a polynomial time algorithm for this problem. For the problem $1|\mathrm{ST}_{\mathrm{sd},f}|\sum C_j$, where $\sum C_j$ is the total completion time, Karabati and Akkan [29] proposed a Branch and Bound algorithm. Jin et al. [30] addressed the $1|\mathrm{ST}_{\mathrm{sd},f}|L_{\max}$ problem and developed a Simulated Annealing heuristic. For the same problem, Jin et al. [31, 32] proposed dominance relations and Tabu Search heuristics. Sels and Vanhoucke [33] considered the $1|r_j, \mathrm{ST}_{\mathrm{sd},f}|L_{\max}$ problem where each job j has a release time r_j (the earliest time at which the job can star its processing). They developed a genetic algorithm with local search. Recently, Herr and Goel [34] addressed a SMTT problem with sequence-dependent family setup times and resource constraints; that is, each job requires a certain amount of resource that is supplied through upstream processes. Schedules must be generated in such a way that the total resource demand does not exceed the resource supply up to any point in time. This problem can be denoted as $1|q_j, \mathrm{ST}_{\mathrm{sd},f}|\sum T_j$, where q_j is the quantity of a resource required by the machine to process the job j. The authors proposed a MILP model and a heuristic algorithm to solve the problem.

To the best of our knowledge, there is only a paper in the literature discussing the $1|\mathrm{ST}_{\mathrm{sd},f}|\sum T_j$ problem. Chantaravarapan et al. [6] propose a MILP model and a hybrid genetic algorithm (HGA) for this problem. The effectiveness of the HGA was evaluated by instances with up to 60 jobs. The experiments presented in [6] showed that HGA performs better than other heuristics.

Motivated by the computational complexity and the practical relevance of the $1|\mathrm{ST}_{\mathrm{sd},f}|\sum T_j$ problem, in this work we test the applicability of the metaheuristic Iterated Local Search (ILS) to find good quality solutions for realistic size problem instances. We use ILS because it is a simple and generally applicable heuristic that has proved to be very effective for solving a wide range of difficult combinatorial optimization problems, especially vehicle routing problems [35–39] and scheduling problems [40–45]. Furthermore, ILS has very few control parameters and it does not require specific knowledge of the problem as in sophisticated heuristic algorithms.

The traditional ILS consists of an iterative process that combines a perturbation phase and a local search phase [46, 47]. During the perturbation phase, the current solution is modified in a probabilistic fashion similar to the mutation operator used in genetic algorithms. In the local search phase, the perturbed solution is improved leading to a local minimum solution. An acceptance criterion is used to decide

whether the search continues from the local minimum solution or from the one that served as the starting point of the most recent perturbation phase.

Besides the standard ILS heuristic, this paper presents other contributions: the ILS is enriched by two special features. The first feature consists in using a variable perturbation size to escape from local optimal solutions. This feature is inspired from the Variable Neighborhood Search (VNS) heuristic that systematically changes the neighborhood within the search [48]. The perturbation size at the beginning is fixed at $d = 1$ and incremented by 1, if the solution is not improved until a $d_{\max}$ value (the maximum perturbation size). If a solution improves in any perturbation size, it is again fixed at $d = 1$. The second feature consists in using a Path Relinking (PR) technique [49] to intensify the search of good solutions. PR generates new solutions by exploring paths that connect elite solutions. The performance of our ILS heuristic with the different features is carefully analyzed.

The remainder of the paper is organized as follows. In Section 2, we present the mathematic formulation for the problem under study. In Section 3, we describe all the phases of the proposed ILS heuristics. In Section 4, we show the design of instances and the calibration of our heuristics and report the computational results. Finally, in Section 5, we conclude this paper and give future directions.

2. Problem Model

In this section we present a mixed-integer linear programming (MILP) model of the $1|\mathrm{ST}_{\mathrm{sd},f}|\sum T_j$ problem. This mathematical model is originally from [6]. The resulting MILP is used to assess the performance of the developed heuristics for small-size problem instances. The goal of the model is to determine the optimal sequence of jobs to be processed on the single machine. A sequence is a permutation of n jobs $(j_1, j_2, \ldots, j_n)$, where j_k is the kth job of the processing sequence (i.e., j_k is the job processed in the position k).

The following parameters are used in the model (input data):

F = number of families,

n_l = number of jobs in family l $(l = 1, \ldots, F)$,

n = total number of jobs $(n = n_1 + n_2 + \cdots + n_F)$,

d_{jl} = due date of the jth job in family l $(l = 1, \ldots, F, j = 1, \ldots, n_l)$,

p_{jl} = processing time of the jth job in family l $(l = 1, \ldots, F, j = 1, \ldots, n_l)$,

s_{lq} = sequence-dependent setup time of preceding family l and following family q $(l, q = 1, \ldots, F)$,

s_{0q} = setup time of family q before the first position $(q = 1, \ldots, F)$.

The following decision variables are used within the model:

$$X_{jlk} = \begin{cases} 1, & \text{if job } j \text{ from family } l \text{ is processed in position } k. \\ 0, & \text{otherwise.} \end{cases}$$

$$Y_{lqk} = \begin{cases} 1, & \text{if setup time } s_{lq} \text{ is needed before a job at position } k. \\ 0, & \text{otherwise.} \end{cases}$$

$$Y_{0q1} = \begin{cases} 1, & \text{if setup time } s_{0q} \text{ is needed before the first job.} \\ 0, & \text{otherwise.} \end{cases}$$

C_k = completion time of job at position k.

T_k = tardiness of job at position k.

The resulting MILP model is as follows:

$$\min \quad \mathrm{TT} = \sum_{k=1}^{n} T_k \tag{2}$$

$$\text{s.t.} \quad \sum_{l=1}^{F}\sum_{j=1}^{n_l} X_{jlk} = 1, \quad k = 1, \ldots, n, \tag{3}$$

$$\sum_{k=1}^{n} X_{jlk} = 1, \quad j = 1, \ldots, n_l, \ l = 1, \ldots, F, \tag{4}$$

$$\sum_{j=1}^{n_q} X_{jq1} = Y_{0q1}, \quad q = 1, \ldots, F, \tag{5}$$

$$C_1 = \sum_{q=1}^{F} s_{0q} Y_{0q1} + \sum_{q=1}^{F}\sum_{j=1}^{n_q} p_{qj} X_{qj1}, \tag{6}$$

$$\sum_{j=1}^{n_q} X_{jqk} + \sum_{j=1}^{n_l} X_{jl(k-1)} - Y_{lqk} \le 1, \quad k = 2, \ldots, n, \ l = 1, \ldots, F, \ q = 1, \ldots, F, \ l \ne q, \tag{7}$$

$$C_k = C_{k-1} + \sum_{q=1}^{F}\sum_{l=1}^{F} s_{lq} Y_{lqk} + \sum_{q=1}^{F}\sum_{j=1}^{n_q} p_{jq} X_{jqk}, \quad k = 2, \ldots, n, \tag{8}$$

$$T_k \ge C_k - \sum_{l=1}^{F}\sum_{j=1}^{n_l} d_{jl} X_{jlk}, \quad k = 1, \ldots, n, \tag{9}$$

$$C_k, T_k \ge 0, \quad k = 1, \ldots, n, \tag{10}$$

$$X_{jlk} \in \{0, 1\}, \quad l = 1, \ldots, F, \ j = 1, \ldots, n_l, \ k = 1, \ldots, n, \tag{11}$$

$$Y_{lqk} \in \{0, 1\},$$

$$Y_{0q1} \in \{0, 1\}, \quad k = 2, \ldots, n, \ l = 1, \ldots, F, \ q = 1, \ldots, F. \tag{12}$$

The objective function (the total tardiness) to be minimized is defined in (2). Constraint set (3) assures that one position of the sequence can contain only one job. Constraint set (4) guarantees that each job should be processed only once. Constraint sets (5) and (6) calculate the completion time of job in the first position of the sequence (C_1). Constraint set (5) controls the setup time of the first position, resulting in the completion time of the first position in (6). Constraint sets (7) and (8) calculate the completion times from the second position to the last position of the sequence. For any two consecutive jobs, constraint set (7)

```
output: Best Solution S*
(1) begin
(2)     S ≔ CONSTRUCTION();
(3)     S ≔ LOCAL_SEARCH(S, γ);
(4)     S* = S; //the best solution
(5)     while stop_condition do
(6)         S1 ≔ PERTURBATION(S, d);
(7)         S2 ≔ LOCAL_SEARCH(S1, γ, pivot_rule);
(8)         if f(S2) < f(S*) then
(9)             S* ≔ S2;
(10)        if f(S2) < f(S) then
(11)            S ≔ S2;
(12)        else
(13)            S ≔ ACCEPTANCE_CRITERION(S, S2, β);
(14)    return S*
```

Algorithm 1: ILS_BASIC(d, γ, *pivot_rule*, β, *stop_condition*).

checks whether or not the preceding job and the following job are from the same family. If so, there is no setup time between them ($Y_{lqk} = 0$). Otherwise, setup time s_{lq} exists ($Y_{lqk} = 1$). Constraint (9) computes the tardiness value (T_k) of jobs at each position k. Constraints (10) represent nonnegativity conditions of the decision variables C_k and T_k, while constraints (11) and (12) ensure that X_{jlk} and Y_{lqk} are binary variables.

3. Proposed ILS Heuristics

To solve the $1|\mathrm{ST}_{\mathrm{sd},f}|\sum T_j$ problem, in this section, we propose three heuristics based on the Iterated Local Search (ILS) metaheuristic. ILS is a simple and generally applicable metaheuristic that iteratively applies a *perturbation* procedure as a diversification mechanism and a *local search* (LS) as an improvement heuristic. At each iteration, a new initial solution S_1 is generated by randomly performing an appropriate modification, called perturbation, to a good locally optimal solution previously found (current solution). Instead of generating a new initial solution from scratch, the perturbation mechanism generates a promising initial solution by retaining part of the structure that made the current solution a good solution. The perturbed solution S_1 is improved by the LS heuristic obtaining a new solution S_2. The solution S_2 is accepted as the new current solution under some conditions defined by the *acceptance criteria*. A detailed explanation of the ILS metaheuristic can be found in [47].

The first proposed heuristic, called ILS_BASIC, is a standard implementation of ILS. The pseudocode of ILS_BASIC is described in Algorithm 1. To implement the basic ILS algorithm, four procedures are specified: (i) CONSTRUCTION, where an initial solution is constructed; (ii) LOCAL_SEARCH, which improves the solution initially obtained and the perturbed solution; (iii) PERTURBATION, where a new starter point is generated through a perturbation of the current solution; (iv) ACCEPTANCE_CRITERION, which determines from which solution the search should continue. The best solution S^* found over all iterations is returned by the ILS algorithm.

```
output: Best solution S*
(1)  begin
(2)      S ≔ CONSTRUCTION();
(3)      S ≔ LOCAL_SEARCH(S, γ, pivot_rule);
(4)      d ≔ 1;
(5)      cont ≔ 0;
(6)      S* ≔ S; //the best solution
(7)      while stop_condition do
(8)          S1 ≔ PERTURBATION(S, d);
(9)          S2 ≔ LOCAL_SEARCH(S1, γ, pivot_rule);
(10)         if f(S2) < f(S*) then
(11)             S* ≔ S2;
(12)             cont ≔ 0;
(13)             d ≔ 1;
(14)         else
(15)             cont ≔ cont + 1;
(16)             if cont mod N = 0 then
(17)                 d ≔ min(d + 1, dmax);
(18)         if f(S2) < f(S) then
(19)             S ≔ S2;
(20)         else
(21)                 S ≔ ACCEPTANCE_CRITERION(S, S2, β);
(22)     return S*
```

Algorithm 2: ILS_DP(N, $d_{\max}$, γ, *pivot_rule*, β, *stop_condition*).

Our ILS_BASIC algorithm has four input parameters: d, γ, *pivot_rule*, β, and *stop_condition*. Parameter d defines the perturbation size. Parameter γ is used to reduce the size of the neighborhood in the LS procedure (this parameter defines the probability to generate a neighbor solution). *pivot_rule* defines the way of selecting the next neighbor solution in the local search process (two selection strategies are tested: first-improvement and best-improvement). Parameter β is used in the ACCEPTANCE_CRITERION (β defines the probability to accept a worse solution). The iterations of the ILS algorithm are computed until a stopping condition (defined by the parameter *stop_condition*) is satisfied.

It is important to mention that the perturbation parameter d used in the ILS_BASIC algorithm is static; that is, d is fixed *a priori* before the beginning of the ILS search. The performance of the algorithm strongly depends on the intensity of the perturbation mechanisms. If it is small, not many new solutions will be explored, while if it is too large, it will adopt almost randomly starting solutions.

The second proposed heuristic, called ILS_DP, improves ILS_BASIC by using a dynamic and adaptive perturbation; that is, the value of d (perturbation size) is defined dynamically during the search according to updating of the best solution found by the algorithm. ILS_DP algorithm, besides using the parameters γ, *pivot_rule*, and β of ILS_BASIC, uses two new parameters: N and $d_{\max}$. N controls the increment frequency of d, and $d_{\max}$ defines the maximum value for d. The steps of ILS_DP are summarized in Algorithm 2. In Step (2), an initial solution is constructed and it is improved by local search (LS) procedure (Step 3). In Step (4) the perturbation size is initialized ($d = 1$). The iterations of the algorithm ILS_DP are computed in Steps (7) to (21) until

```
output: Best solution S*
(1)  begin
(2)    S ≔ CONSTRUCTION();
(3)    S ≔ LOCAL_SEARCH(S, γ, pivot_rule);
(4)    d ≔ 1;
(5)    cont ≔ 0;
(6)    S* ≔ S; //the best solution
(7)    EliteSet ≔ {S*};
(8)    while stop_condition do
(9)      S1 ≔ PERTURBATION(S, d);
(10)     S2 ≔ LOCAL_SEARCH(S1, γ, pivot_rule);
(11)     S3 ≔ Randomly select a solution from EliteSet;
(12)     S4 ≔ PATH_RELINKING(S2, S3);
(13)     EliteSet ≔ UPDATE_ELITE_SET(S4, nE);
(14)     if f(S4) < f(S*) then
(15)       S* ≔ S4;
(16)       cont ≔ 0;
(17)       d ≔ 1;
(18)     else
(19)       cont ≔ cont + 1;
(20)       if cont mod N = 0 then
(21)         d ≔ min(d + 1, dmax);
(22)     if f(S4) < f(S) then
(23)       S ≔ S4;
(24)     else
(25)       S ≔ ACCEPTANCE_CRITERION(S, S4, β);
(26)   return S*
```

Algorithm 3: ILS_DP+PR($N, d_{\max}, \gamma$, *pivot_rule*, β, *stop_condition*, n_E).

a stopping condition is satisfied. During each iteration, the current solution S is perturbed (Step 8) and improved by local search, obtaining a solution S_2 (Step 9). In Steps (10) to (13), if solution S_2 improves the best solution S^* obtained so far, the perturbation size is set to its lowest value ($d = 1$). If the best solution is not improved during N consecutive iterations, the perturbation size is incremented (Steps 15 to 17). The maximum value of d is $d_{\max}$. In Steps (18) to (21), if solution S_2 improves the current solution S, it is accepted as the new current solution; otherwise solution S_2 is accepted if it meets the acceptance criterion.

The third proposed heuristic, named ILS_DP+PR, is an extension of the second heuristic. ILS_DP+PR combines ILS and Path Relinking (PR) generating a hybrid heuristic. PR is an approach that generates new solutions by exploring trajectories that connect high-quality solutions [50]. PR uses a set of n_E high-quality solutions (*EliteSet*). We adapted PR in the context of ILS as a form of intensification, which consists in finding a path between the solution returned by the LS procedure and an elite solution. A general pseudocode description of ILS_DP+PR is presented in Algorithm 3. The algorithm has an additional parameter: n_E (the maximum size of the elite set). The algorithm ILS_DP+PR differs from ILS_DP in the following steps. In Step (7), the elite set *EliteSet* is initialized with the best solution S^* obtained at the beginning of the algorithm. During each iteration, in Step (12), the PATH_RELINKING intensification is applied between the solution S_2 (returned by local search) and the solution S_3 selected randomly from *EliteSet* (Step 11). The PR returns the best solution found (S_4). In Step (13), the elite set is updated with solution S_4.

The next subsections provide a detailed explanation of the main components that are used in the proposed ILS heuristics.

Table 1: Input data for an instance with 7 jobs and 2 families.

(a) Families, due dates, and processing times

Job j	**1**	**2**	**3**	**4**	**5**	**6**	**7**
$f(j)$	1	2	1	2	2	1	2
d_j	2	7	18	11	8	15	3
p_j	1	2	4	2	4	3	2

(b) Setup times between families

s_{ik}	**1**	**2**
1	0	1
2	2	0

3.1. Representation of a Solution. A solution (schedule) of the $1|\mathrm{ST}_{\mathrm{sd},f}|\sum T_j$ problem is represented by a permutation of n jobs. For an instance with $n = 7$ and $F = 2$, in Table 1 is presented an example of an input data. For each job j, the family, processing time, and due date are presented in Table 1(a). The setup times between the families are shown in Table 1(b).

For the schedule $S = \{7, 1, 5, 4, 2, 6, 3\}$, illustrated in Figure 1, the completion time and tardiness of each job are $C_7 = 2, C_1 = 5, C_5 = 10, C_4 = 12, C_2 = 14, C_6 = 19, C_3 = 23$ and $T_7 = 0, T_1 = 3, T_5 = 2, T_4 = 1, T_2 = 7, T_6 = 4, T_3 = 5$, respectively. Therefore, the total tardiness for this schedule is $\mathrm{TT}(S) = 22$. We can see that the schedule S forms four batches and three setup times are required (s_{21}, s_{12}, and s_{21}). In this example, setup time is not considered at the beginning of the sequence.

3.2. Construction of the Initial Solution. The CONSTRUCTION procedure used by the ILS algorithms is based on the NEH heuristic [51]. In this work, to generate an initial solution, we adapt the NEH heuristic for single-machine scheduling and total tardiness minimization. In this heuristic first, the jobs are arranged in nondecreasing order of the due dates (Earliest Due Date dispatching rule) forming an ordered list of jobs $J = \{j_1, \ldots, j_n\}$. The first job of J is selected to form a partial sequence S ($S = \{j_1\}$). For each $i = 2, \ldots, n$, the next job j_i of J is inserted in all possible positions of S generating i partial sequences. For example, for $i = 2$, we obtain two partial sequences by inserting job j_2 in the two positions of $S = \{j_1\}$: $\{j_2, j_1\}$ and $\{j_1, j_2\}$. The best partial sequence (in relation to the partial total tardiness) is selected to replace S. The heuristic finishes when a complete sequence S is obtained (i.e., all the jobs of J were inserted into S).

3.3. Local Search. The LOCAL_SEARCH procedure is used to improve a solution S (which can be the initial solution or the perturbed solution) and is based on the insertion neighborhood. This procedure is shown in Algorithm 4. The iterations

```
output: Improved solution S
(1)  begin
(2)     improve := true;
(3)     while improve do
(4)        improve := false;
(5)        J := {j_1, ..., j_n} set of jobs determined at random;
(6)        for i := 1 to n do
(7)           j := randomly select a job from J;
(8)           J := J − {j};
(9)           if rand(0...1) ≤ γ then
(10)             S' := best solution obtained by inserting job j in all position of S;
(11)             if f(S') < f(S) then
(12)                S := S';
(13)                improve := true;
(14)                if pivot_rule = 1 then
(15)                   Go to step (4) /* First Improvement */
(16)  return S
```

Algorithm 4: LOCAL_SEARCH(S, γ, *pivot_rule*).

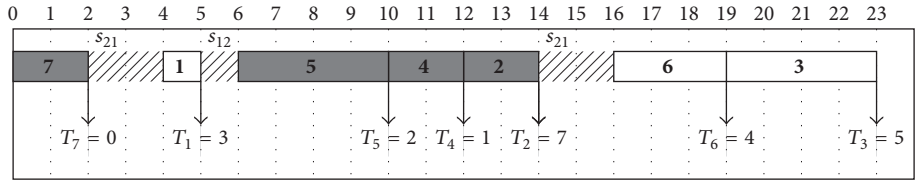

Figure 1: A schedule **S** = {**7**, **1**, **5**, **4**, **2**, **6**, **3**}.

of the local search algorithm are determined in Steps (3) to (15). In each iteration, neighbor solutions of the current sequence S are constructed by considering the set J of all jobs. Each job j is removed from the current sequence (at random and without repetition (Steps 7 and 8)) and then inserted into all possible $n-1$ positions of S (Step 10). The current solution S is replaced by the best solution (S') among the $n-1$ possible ones, only if an improvement of S can be obtained (Steps 11 and 12). If $\gamma = 1.0$, the entire neighborhood is evaluated; that is, all jobs are selected to be inserted in all possible $n-1$ positions. In this case, the size of the neighborhood is $(n-1)^2$. The parameter γ defines the job selection probability. To reduce the size of the neighborhood, we can use $\gamma < 1.0$. We also test two strategies to select the solution to be explored in next iteration: *first-improvement* and *best-improvement*. If *pivot_rule* = 1 (Step 14), the selection strategy is the *first-improvement*; otherwise, it is the *best-improvement*. Note that, with the *first-improvement* strategy (*pivot_rule* = 1), a new iteration is started from the first improved solution S'. The iterations of the LOCAL_SEARCH procedure are repeated while there is improvement in the current solution S (Steps 3–15); that is, the procedure ends when the solution is a local optimum with respect to the insertion neighborhood.

3.4. Perturbation. The goal of the perturbation method in an ILS heuristic is escape from a local optimal solution. In this work, we used a perturbation mechanism based on jobs exchanges. The perturbation size (parameter d) defines the number of exchanges to be made. To perturb a solution S, a position k ($1 \le k \le n-2d+1$) is first randomly chosen. Then, exchanges are applied between the jobs of positions $k+i$ and $k+2d+1-i$ ($i = 0, \ldots, d$). That is, the PERTURBATION procedure executes $d+1$ exchanges. For example, let us consider a schedule with $n = 8$ jobs, $S = \{j_1, j_2, j_3, j_4, j_5, j_6, j_7, j_8\}$. Let $k = 3$ be a position randomly chosen. For $d = 1$, two pairs of jobs are exchanged: (j_3, j_6) and (j_4, j_5). For $d = 2$, three pairs of jobs are exchanged: (j_3, j_8), (j_4, j_7), and (j_5, j_6). For $d = 1$ and $d = 2$, the perturbed schedules are $S' = \{j_1, j_2, j_6, j_5, j_4, j_3, j_7, j_8\}$ and $S'' = \{j_1, j_2, j_8, j_7, j_6, j_5, j_4, j_3\}$, respectively.

In ILS_BASIC algorithm, parameter d has a fixed value. In ILS_DP and ILS_DP+PR, the value of d varies in the interval $[1, d_{\max}]$, where $d_{\max}$ is a parameter to be tuned, such that $d_{\max} \le (n/2 - 1)$.

3.5. Path Relinking Intensification. The Path Relinking (PR) technique was originally proposed by [50] as a mechanism to combine intensification and diversification by exploring trajectories connecting high-quality (elite) solutions previously produced during the search. PR needs a pair of solutions, say S_I (initiating solution) and S_G (guiding solution), $S_I \neq S_G$. A path that links S_I to S_G is generated by applying neighborhood movements to the initiating solution, which progressively introduces attributes from the guiding solution [49].

In ILS_DP+PR algorithm, we use the PR variant called Mixed Path Relinking (MPR) [52]. Instead of starting from

a solution S_I and gradually transforming it into the solution S_G, the MPR procedure performs one step from S_I to S_G, obtaining an intermediate solution A. Then S_G becomes the initiating solution and A the guiding solution, obtaining a new intermediate solution B. In the next step of the procedure, A becomes the initiating solution and B the guiding solution, obtaining an intermediate solution C, and so on until the intermediate solution is equal to the guiding solution. The main advantage of this strategy is that it explores deeply neighborhoods of both input solutions. The MPR procedure returns the best solution obtained during the construction of the paths that connect the solutions S_I and S_G.

In this paper, the paths are generated applying exchange moves. For example, let us consider the solutions $S_I = \{2, 3, 1, 4, 5, 7, 6\}$ and $S_G = \{2, 7, 5, 3, 4, 6, 1\}$ ($n = 7$ jobs). Since these solutions have only one job in the same position (job 2 is in the first position in both solutions), the distance (or difference) between S_I and S_G is 6. Starting from S_I, six neighbor solutions are generated by exchanging two jobs so that the distance with respect to S_G is reduced by one: $\{2, \mathbf{7}, 1, 4, 5, 3, 6\}$, $\{2, 3, \mathbf{5}, 4, 1, 7, 6\}$, $\{2, 4, 1, \mathbf{3}, 5, 7, 6\}$, $\{2, 3, 1, 5, \mathbf{4}, 7, 6\}$, $\{2, 3, 1, 4, 5, \mathbf{6}, 7\}$, and $\{2, 3, 6, 4, 5, 7, \mathbf{1}\}$. From these solutions, the best one is chosen to be the intermediate solution A (e.g., $A = \{2, 3, \mathbf{5}, 4, 1, 7, 6\}$). Considering S_G the initiating solution and A the guiding solution, five neighbor solutions are generated. From these five solutions the best one is chosen to be the new intermediate solution B (e.g., $B = \{2, 6, \mathbf{5}, 3, 4, \mathbf{7}, 1\}$). Note that the difference between A and B is 4. The procedure ends when the difference between the solutions initiating and guiding is 1.

The initiating solution S_I is the solution returned by the LOCAL_SEARCH procedure and the guiding solution S_G is selected at random from the elite set (*EliteSet*). This set represents the pool of the best different solutions found by the algorithm. The maximum size of *EliteSet* is n_E. The UPDATE_ELITE_SET procedure tests if a solution will be added or not to *EliteSet*. When the elite set is full ($|EliteSet| = n_E$), a solution S is added to *EliteSet* if $S \notin EliteSet$ and S is better than the worst solution in *EliteSet*. In this case, S is added to *EliteSet* in place of the most similar worst solution, that is, the solution with the smallest distance (or difference) from S.

4. Computational Experiments

In this section, we describe the experiments carried out in order to study the behavior of the developed heuristic algorithms ILS_BASIC, ILS_DP, and ILS_DP+PR.

To the best of our knowledge, Chantaravarapan et al.'s study [6] is the only previous study to consider the problem $1|\mathrm{ST}_{\mathrm{sd},f}|\sum T_j$ addressed in this paper. Chantaravarapan et al. [6] proposed a hybrid genetic algorithm (HGA) which uses a local search heuristic in order to enhance performance. Once the best-fit chromosome in the offspring pool is determined, the local search heuristic is applied to that chromosome to improve the fitness before it is replaced into the population pool. The local search heuristic is based on two interchange methods: Adjacent Pairwise Interchange (API) and Randomized Pairwise Interchange (RdPI). The API swaps two consecutive jobs from the first position of the sequence to the last position. The RdPI randomly chooses any two jobs and switches the positions. If the solution is improved after switching, the initial sequence is updated and the process starts from the first position of the sequence. The local search continues until there is no improvement in solution.

Table 2: Factors and the levels of the instances generated.

Factors	Levels
n	60, 80, 100 (large instances) and 10, 15, 20 (small instances)
F	2, 3, 4, 5 (large instances) and 2, 4, 6 (small instances)
ST	S: [11, 20], M: [51, 100], and L: [101, 200]
r	0.5, 1.5, 2.5, and 3.5

We compare our ILS heuristics against HGA on 1440 randomly generated test instances. For small instances we compare the performance of the best proposed heuristic with respect to the MILP model solved by the commercial optimization solver IBM-ILOG CPLEX 12.5, which we will simply refer to as CPLEX. All of the heuristic algorithms and the MILP model have been coded in C++ and run on a PC with an Intel(R) Xeon(R) CPU X5650, 2.67 GHz with 48 GB of RAM running Ubuntu Linux 14.04.

In the following subsections, we first describe the random instance generation; then we present the parameter setting of our ILS algorithms; next we analyze the results obtained on large and small instances, and finally we analyze the convergence of the algorithms.

4.1. Random Test Instances. To test the performance of the proposed heuristic algorithms, random instances of the $1|\mathrm{ST}_{\mathrm{sd},f}|\sum T_j$ problem are generated according to Jin et al. [32]. The number of jobs is classified into two sets: $n \in \{60, 80, 100\}$ (large-size instances) and $n \in \{10, 15, 20\}$ (small-size instances). The number of families is set to be $F \in \{2, 3, 4, 5\}$ (large-size instances) and $F \in \{2, 4, 6\}$ (small-size instances).

The processing times of jobs (p_j) are uniformly distributed in the interval $[1, 99]$. Three classes of setup times (ST) are considered. Small (S), medium (M), and large (L) setup times are uniformly distributed in the ranges $[11, 20]$, $[51, 100]$, and $[101, 200]$, respectively. The due dates (d_j) of jobs are integer numbers uniformly distributed in the interval $[0, r\sum_{j=1}^{n} p_j]$, where r is a factor used to control the range of due dates, $r \in \{0.5, 1.5, 2.5, 3.5\}$. Small and large due dates are generated with $r = 0.5$ and $r = 3.5$, respectively.

The factors and the levels of the generated instances are shown in Table 2. Combining the factors n, F, r, and ST, there are 144 categories of large instances and 108 categories of small instances. For each category of small and large instances, 10 and 2 instances are generated, respectively. Therefore, a total of 1440 large instances and 216 small instances were generated. All the generated instances are available for download at http://www.dpi.ufv.br/projetos/scheduling/smtt.htm.

4.2. Performance Measures. The most common performance measure used in the literature to compare the results of

heuristic methods is the Relative Percentage Deviation (RPD) from the best known solution [53]. The RPD is computed according to the following equation:

$$\text{RPD}\,(\%) = 100 \times \frac{f_{\text{method}} - f_{\text{best}}}{f_{\text{best}}}, \tag{13}$$

where f_{method} is the solution (objective function value) obtained by a given method and f_{best} is the best solution obtained among all the methods or the best known solution, possibly optimal. With this performance measure, good methods will have RPD close to 0.

In the $1|\text{ST}_{\text{sd},f}|\sum T_j$ problem, the minimum total tardiness (best solution) could be zero, so the RPD gives a division by zero. In this case, the Relative Deviation Index (RDI) measure is used to avoid division by zero. The RDI is computed as follows:

$$\text{RDI}\% = 100 \times \frac{f_{\text{method}} - f_{\text{best}}}{f_{\text{worst}} - f_{\text{best}}}, \tag{14}$$

where f_{best} and f_{worst} are the best and the worst solutions obtained among all the compared methods, respectively. With this performance measure, a RDI between 0 and 100 is obtained for each method such that a good method will have a RDI close to 0. Note that if the worst and the best solutions take the same value, all the compared methods provide the best (same) solution and, hence, the RDI value will be 0 (best index value) for all the methods.

4.3. Calibration of Parameters. In this subsection we present the parameter setting of the developed heuristic algorithms ILS_BASIC, ILS_DP, and ILS_DP+PR. In order to calibrate the parameters of the heuristics, we generate a set of 100 random large instances (calibration instances) according to Section 4.1.

To make a fair comparison, the stopping condition (*stop_condition*) for all the heuristics is set to a maximum CPU elapsed time equal to $0.5 * n$ seconds. Setting the time limit in this way allows more computation effort as the number of jobs increases [53]. In the calibration experiments, all the heuristics are run five independent times to obtain a final average of the results.

A Design of Experiments (DOE, [54]) is carried out to calibrate the proposed heuristics, where the factors are the parameters that need calibration and the response variable is the performance of the different algorithm configurations. Since the best known solution of each calibration instance is nonzero, the performance of each algorithm configuration is measured by RPD determined by (13). The experimental results were analyzed by means of a multifactor analysis of variance (ANOVA) technique [54]. In this statistical technique two hypotheses are tested, (i) null hypothesis: the medians (or performances) of all algorithms are equal; and (ii) alternative hypothesis: the median of at least one algorithm is different. To apply ANOVA, we checked its three main hypotheses, that is, normality, homoscedasticity, and independence of residuals. Statistical analysis showed that all of the three hypotheses could be accepted.

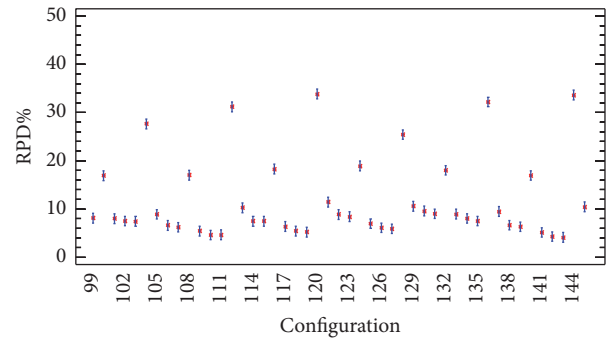

Figure 2: Means plot and Tukey's HSD intervals at the 95% confidence level for 47 configurations of ILS_BASIC.

The first heuristic ILS_BASIC depends on four parameters: d (perturbation size), γ (probability to generate a neighbor solution in the LS procedure), *pivot_rule* (which defines the selection strategy, best-improvement (0), or first-improvement (1)), and β (which defines the probability to accept a worse solution). We carry out a full factorial experiment with these four parameters. A series of preliminary experiments were conducted in order to determine a suitable set of levels to test. The following levels were tested: $d \in \{\lceil n/10\rceil, \lceil n/5\rceil, \lceil n/4\rceil, \lceil n/3\rceil, \lceil n/2\rceil - 1\}$, $\gamma \in \{0.1, 0.3, 0.6, 1.0\}$, *pivot_rule* $\in \{0, 1\}$, and $\beta \in \{0.0, 0.3, 0.6, 1.0\}$. All combinations of values of the four parameters give 160 configurations of ILS_BASIC. Each ILS_BASIC configuration is run five different times on each calibration instance. The results of ANOVA technique indicate that there is statistically significant difference between the obtained results at a 95% confidence level. Since the *p_value* of ANOVA test is less than 0.05, the null hypothesis should be rejected in favor of the alternative hypothesis. Figure 2 shows, for some configurations of ILS_BASIC, the means plot and Tukey's Honestly Significant Difference (HSD) confidence intervals with 95% confidence level from the statistical test. For easy viewing, Figure 2 only shows the confidence intervals of configurations 99–145. Overlapping intervals indicate that no statistically significant difference exists among the overlapped means. We can clearly see that there are statistically significant differences among some configurations of ILS_BASIC. For example, the configurations 99, 100, and 104 are statistically different. Configuration 99 is statistically better than configuration 100 which in turn is statistically better than configuration 104. Also, we can see that configurations 99, 101, 102, and 103 are statistically equivalent. However, configuration 143 presents the smallest average RPD. This configuration of ILS_BASIC corresponds to $d = \lceil n/3\rceil$, $\gamma = 1.0$, *pivot_rule* $= 1$, and $\beta = 0.3$. Thus, these parameter values will be used for ILS_BASIC in the next experiments.

The ILS_DP algorithm is calibrated in the same way as ILS_BASIC. For ILS_DP, five parameters should be calibrated: γ, *pivot_rule*, β, N, and $d_{\max}$. The first three parameters are the same parameters of ILS_BASIC. Parameter N controls the increment frequency of the perturbation size (d) and $d_{\max}$ is the maximum value of d. For the five parameters of ILS_DP, the following levels were tested: $\gamma \in \{0.1, 0.3, 0.6, 1.0\}$,

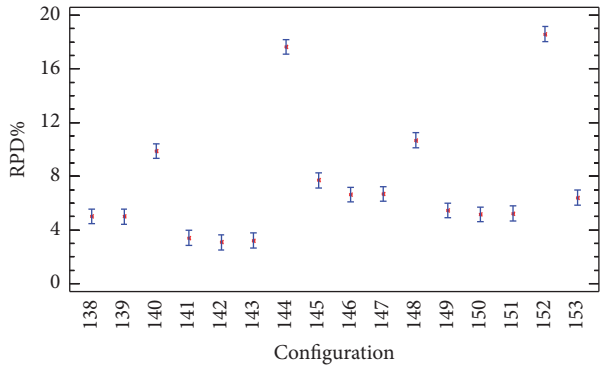

Figure 3: Means plot and Tukey's HSD intervals at the 95% confidence level for 16 configurations of ILS_DP.

$pivot\_rule \in \{0, 1\}$, $\beta \in \{0.0, 0.3, 0.6, 1.0\}$, $N \in \{1, 3, 5, 10\}$, and $d_{\max} \in \{\lceil n/10 \rceil, \lceil n/5 \rceil\}$. The combination of these parameters (factors) yields a total of 256 different configurations of ILS_DP algorithm. Each configuration is run five different times on each calibration instance. The results of ANOVA test indicate that there is statistically significant difference between the obtained results at a 95% confidence level. The results of 16 configurations (including the best configuration) are shown in Figure 3. This figure shows the means plot and Tukey's HSD confidence intervals with 95% confidence level from the statistical test. We can see that there are statistically significant differences among some configurations of ILS_DP. For example, the configurations 141, 142, and 143 are statistically better than the other configurations. These three configurations are statistically equivalent. However, configuration 142 presents the smallest average RPD. This configuration corresponds to following parameters: $\gamma = 0.6$, $pivot\_rule = 1$, $\beta = 0.6$, $N = 1$, and $d_{\max} = \lceil n/5 \rceil$. Since the value found for parameter $d_{\max} = \lceil n/5 \rceil$ is a threshold value, other levels were tested for this parameter: $d_{\max} \in \{\lceil n/4 \rceil, \lceil n/3 \rceil, \lceil n/2 \rceil - 1\}$. The best results were obtained with $d_{\max} = \lceil n/3 \rceil$. Thus, the parameters $\gamma = 0.6$, $pivot\_rule = 1$, $\beta = 0.6$, $N = 1$, and $d_{\max} = \lceil n/3 \rceil$ will be used for ILS_DP in the next experiments.

Finally, we calibrate the ILS_DP+PR algorithm. Six parameters affect the performance of this algorithm: γ, $pivot\_rule$, β, N, $d_{\max}$, and n_E. Parameter n_E is the maximum size of the elite set used in the Path Relinking intensification. Since the first-improvement strategy ($pivot\_rule = 1$) gives the best results in the algorithms ILS_BASIC and ILS_DP, parameter $pivot\_rule$ is set to be 1 in ILS_DP+PR. For the other parameters of ILS_DP+PR, the following levels were tested: $\gamma \in \{0.1, 0.3, 0.6, 1.0\}$, $\beta \in \{0, 0.3, 0.6, 1.0\}$, $N \in \{1, 3, 5, 10\}$, $d_{\max} \in \{\lceil n/10 \rceil, \lceil n/5 \rceil\}$, and $n_E \in \{10, 20\}$. All combinations of values of these parameters give 256 configurations of ILS_DP+PR. For some configurations, Figure 4 shows the means plot and Tukey's HSD confidence intervals with 95% confidence level from the ANOVA test. We can see that the configurations 121, 123, and 132 present the best results. However, configuration 123 (with $\gamma = 0.6$, $\beta = 0.3$, $N = 1$, $d_{\max} = \lceil n/5 \rceil$, and $n_E = 10$) presents the smallest average RPD. Since the values found for parameters $d_{\max}$ and n_E are threshold values, other levels were tested for these parameters: $d_{\max} \in \{\lceil n/4 \rceil, \lceil n/3 \rceil, \lceil n/2 \rceil - 1\}$ and $n_E \in \{3, 5\}$. New six configurations of ILS_DP+PR are generated by combining the values of these two parameters. The best configuration was obtained with $d_{\max} = \lceil n/3 \rceil$ and $n_E = 5$.

Table 3: Parameters setting.

Algorithm	Values of the parameters
ILS_BASIC	$\gamma = 1.0$, $\beta = 0.3$, $pivot\_rule = 1$, and $d = \lceil n/3 \rceil$
ILS_DP	$\gamma = 0.6$, $\beta = 0.6$, $pivot\_rule = 1$, $N = 1$, and $d_{\max} = \lceil n/3 \rceil$
ILS_DP+PR	$\gamma = 0.6$, $\beta = 0.3$, $pivot\_rule = 1$, $N = 1$, $d_{\max} = \lceil n/3 \rceil$, and $n_E = 5$

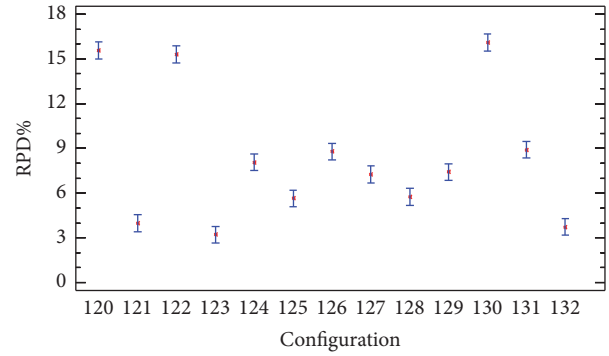

Figure 4: Means plot and Tukey's HSD intervals at the 95% confidence level for 13 configurations of ILS_DP+PR.

Based on the above results, the final parameter settings used in the proposed heuristic algorithms are shown in Table 3.

4.4. Comparison of the Heuristics on Large-Size Instances. In the first experiment, we present the comparison of solution quality generated by the proposed three ILS heuristics, ILS_BASIC, ILS_DP, and ILS_DP+PR, on 1440 large instances. We also compare our heuristics against the heuristic HGA proposed in [6].

Since the heuristics are stochastic methods, each instance is solved 30 times by each heuristic. All the heuristics are run with the same stopping criterion which is based on an amount of CPU time ($0.5 * n$ seconds).

The performances of all the heuristics are measured by the RDI from the best and worst known solutions obtained among all the methods. The RDI is computed for each instance according to (14), where f_{best} and f_{worst} are the best and the worst solutions obtained among all the compared heuristics, respectively. Since each heuristic is run 30 times for each instance, for each heuristic three RDI values are presented: (i) the Best, (ii) the Worst, and (iii) the Average (Avg).

The obtained numerical results of the heuristics are reported in Table 4, where we have averaged the 120 instances of each group $n \times F$. For the heuristics HGA, ILS_BASIC, ILS_DP, and ILS_DP+PR, the Best, Worst, and Average (Avg) results are presented.

Considering the Best RDI values, the three ILS heuristics are better than HGA heuristic. For a total of 1440 large

Table 4: Average RDIs (%) of the heuristics on large size instances.

$n \times F$	HGA			ILS_BASIC			ILS_DP			ILS_DP+PR		
	Best	Worst	Avg	Best	Worst	Avg	Best	Worst	Avg	Best	Worst	Avg
60 × 2	0.83	28.33	12.70	0.00	10.32	1.93	0.00	11.52	1.98	0.00	2.77	0.12
60 × 3	0.83	40.00	26.58	0.00	14.06	1.91	0.00	15.45	2.33	0.00	9.32	0.73
60 × 4	4.64	48.33	35.48	0.00	22.43	3.04	0.00	20.79	3.26	0.00	17.03	2.23
60 × 5	2.89	52.50	38.62	0.00	33.19	4.94	0.00	30.86	5.73	0.00	23.93	4.69
80 × 2	4.79	33.33	22.48	0.00	21.12	4.40	0.00	18.03	4.15	0.00	7.09	0.73
80 × 3	10.08	45.00	33.87	0.00	26.61	5.39	0.00	26.22	4.38	0.00	17.88	2.78
80 × 4	16.36	52.50	43.48	0.00	31.66	5.38	0.00	30.19	5.32	0.00	29.87	4.75
80 × 5	17.03	54.17	46.86	0.00	38.82	6.74	0.00	36.73	6.61	0.00	35.65	7.00
100 × 2	14.38	39.17	29.03	0.27	21.38	5.35	0.00	20.55	3.79	0.00	11.86	1.85
100 × 3	22.85	50.83	41.42	0.00	26.33	7.03	0.00	25.32	5.55	0.00	24.78	4.51
100 × 4	27.40	53.33	45.14	0.00	24.70	4.87	0.00	23.21	4.01	0.00	21.78	4.02
100 × 5	30.41	59.17	47.43	0.00	32.80	7.01	0.00	30.69	7.00	0.00	31.27	6.62
Average	12.71	46.39	35.26	0.02	24.95	4.83	0.00	24.80	4.51	0.00	19.60	3.34

Table 5: Average RDIs (%) for instances grouped by number of jobs n.

(n, F, r, ST)	HGA	ILS_BASIC	ILS_DP	ILS_DP+PR
$(60, *, *, *)$	28.34	2.95	3.32	1.94
$(80, *, *, *)$	36.67	5.48	5.09	3.81
$(100, *, *, *)$	40.76	6.06	5.11	4.25

Table 6: Average RDIs (%) for instances grouped by number of families (F).

(n, F, r, ST)	HGA	ILS_BASIC	ILS_DP	ILS_DP+PR
$(*, 2, *, *)$	21.40	3.90	3.31	0.90
$(*, 3, *, *)$	33.96	4.43	4.08	2.67
$(*, 4, *, *)$	41.37	4.77	4.20	3.67
$(*, 5, *, *)$	44.30	6.23	6.44	6.10

Table 7: Average RDIs (%) for instances grouped by due date ranges (r).

(n, F, r, ST)	HGA	ILS_BASIC	ILS_DP	ILS_DP+PR
$(*, *, 0.5, *)$	84.66	10.90	8.27	4.54
$(*, *, 1.5, *)$	45.61	5.01	5.37	4.45
$(*, *, 2.5, *)$	8.98	2.75	3.05	3.24
$(*, *, 3.5, *)$	1.78	0.66	1.34	1.11

Table 8: Average RDIs (%) for instances grouped by setup time ranges (ST).

(n, F, r, ST)	HGA	ILS_BASIC	ILS_DP	ILS_DP+PR
$(*, *, *, \mathrm{S})$	22.46	5.34	4.21	2.43
$(*, *, *, \mathrm{M})$	36.55	3.59	3.65	2.69
$(*, *, *, \mathrm{L})$	46.76	5.57	5.67	4.88

instances tested, the heuristics HGA, ILS_BASIC, ILS_DP, and ILS_DP+PR find the best known solution in 1182 (82.1%), 1343 (93.2%), 1351 (93.8%), and 1398 (97.1%) instances, respectively. These results show that all the heuristics, in 30 runs, are able to find the best known solution for most instances.

For all groups of instances, the heuristic ILS_DP+PR presents low Worst RDIs. For all groups of instances, except for groups 60×2 and 60×3, the Worst RDIs values of ILS_DP are better than the respective values of ILS_BASIC.

Considering the Avg RDI values, the performance of the heuristic ILS_DP+PR is notoriously better than the other heuristics, except for groups of instances 80×5 and 100×4. The heuristic ILS_DP outperforms, on average, ILS_BASIC for all groups of instances with $n = 80$ and $n = 100$ jobs. ILS_BASIC is better than ILS_DP for instances with $n = 60$ jobs. The obtained average results indicate that the proposed ILS heuristics have a superior performance compared to HGA heuristic. Considering all groups of instances, the heuristics HGA, ILS_BASIC, ILS_DP, and ILS_DP+PR present overall Average (Avg) RDIs of 35.26%, 4.83%, 4.51%, and 3.34%, respectively.

Tables 5, 6, 7, and 8 show the Average RDIs of the heuristics, where the instances are grouped by the number of jobs (n), number of families (F), due date range (r), and setup class (ST), respectively. In Tables 5 and 6 we can see that the Average RDI values of the heuristics increase as the number of jobs (n) and number of families (F) increase, respectively.

In Table 7 we can clearly see that the performance of the heuristics improves as the due date range increases. For all the heuristics, the greatest variation in the results occurs when the due date ranges vary. All the heuristics present short RDIs for instances with large due dates. Generally, for instances with large due dates, zero total tardiness is determined. The heuristic ILS_DP+PR presents the best results for all group of instances, except for instances with large due dates ($r = 2.5$ and $r = 3.5$), where the ILS_BASIC presents the best results (Table 7). Instances with large due dates, generally, are easy to solve because almost all jobs are completed before their due dates. In these instances, ILS_DP+PR is inferior to ILS_BASIC probably because the dynamic perturbation used in ILS_DP+PR is relatively strong that easily allows leaving local minimum solutions.

Table 9: Multiple comparisons test. (∗) denotes a statistically significant difference.

Pair of heuristics	Significant	Difference	Limits
HGA and ILS_BASIC	(∗)	30.4259	0.3043
HGA and ILS_DP	(∗)	30.7494	0.3043
HGA and ILS_DP+PR	(∗)	31.9216	0.3043
ILS_BASIC and ILS_DP	(∗)	0.3235	0.3043
ILS_BASIC and ILS_DP+PR	(∗)	1.4956	0.3043
ILS_DP and ILS_DP+PR	(∗)	1.1721	0.3043

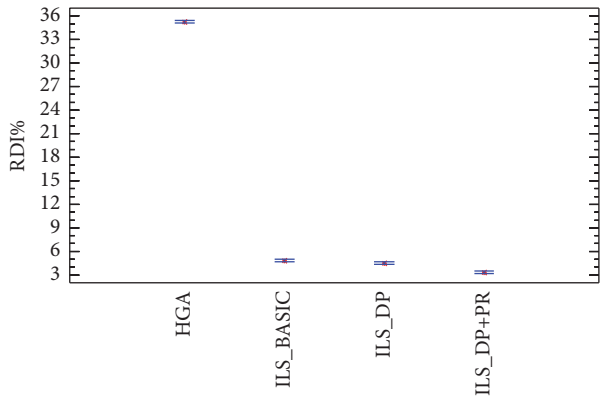

Figure 5: Means plot and Tukey's HSD intervals at 95% confidence level for all heuristics and all large instances.

In Table 8 we can observe that ILS_DP+PR presents the best results and all heuristics have large RDIs for instances with large (L) setup times.

In order to guarantee that the observed differences in the average results are indeed statistically significant, we carry out a parametric ANOVA statistical test using the RDI measure as response variable. We check the three main hypotheses of ANOVA test: normality, homoscedasticity, and independence of the residuals. No significant deviations were found in the fulfillment of the hypotheses. The results of this statistical test indicate that there is statistically significant difference between the obtained results at a 95% confidence level, with a $p\ value = 0.0 < 0.05$. Since the ANOVA test does not specify which heuristics differ significantly, we carry out a Multiple Comparisons test to compare each pair of heuristics with a 95% confidence level. Table 9 shows the result of this test. Column Difference displays the sample mean of the first heuristic minus that of the second. Column Limits shows an uncertainty interval for the difference. Any pair for which the absolute value of the difference exceeds the limit (|Difference| > Limits) is statistically significant at the selected confidence level and is indicated by (∗) in the column Significant. In Table 9, we can see that there is a significant difference between each pair of heuristics, and the heuristics ILS_BASIC and ILS_DP present the smallest difference (Difference − Limits = 0.0192).

The same analysis can be displayed in Figure 5. This figure shows, for the four heuristics and all the large instances, the means plot and Tukey's HSD confidence intervals with

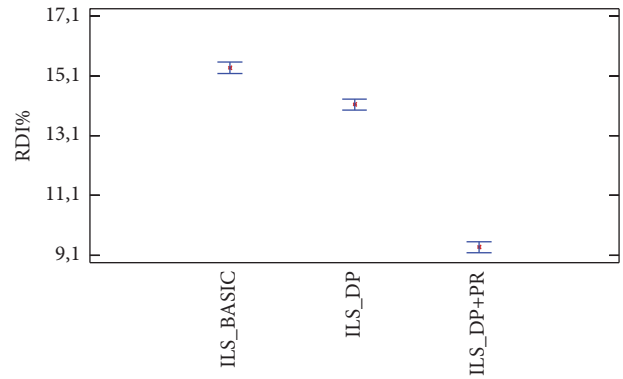

Figure 6: Means plot and Tukey's HSD intervals at 95% confidence level for heuristics ILS_BASIC, ILS_DP, and ILS_DP+PR.

95% confidence level from the ANOVA test. We can see that ILS_DP+PR is statistically the best heuristic because its confidence interval does not overlap with the intervals of any of the other heuristics. The second best algorithm is ILS_DP followed by ILS_BASIC. The proposed ILS heuristics are statistically better than HGA heuristic.

In order to have a clearer picture, the heuristic HGA is removed for subsequent statistical analyses and we carry out another statistical test considering only the three proposed ILS heuristics. In Figure 6 we can clearly see that ILS_DP+PR is statistically better than ILS_DP which in turn is statistically better than ILS_BASIC. These results show that a basic ILS heuristic can be considerably improved by using a dynamic or variable perturbation size and Path Relinking intensification technique.

The obtained average results show that the effectiveness of the heuristic ILS_BASIC is considerably improved by using a dynamic perturbation and an intensification mechanism based on the Path Relinking technique. We believe that our ILS heuristics are better than HGA because we use a local search procedure based on the insertion neighborhood. HGA uses a local search heuristic based on the interchange neighborhoods API and RPI. In scheduling problems which have a permutation representation, insertion neighborhoods are efficient and strongly preferable over the interchange neighborhoods [55]. Furthermore, local search based metaheuristics are generally more effective than population based algorithms such as Ant Colony Optimization, Particle Swarm Optimization, and genetic algorithm [40, 56]. In order to facilitate follow-up research, the results obtained in this paper are available at http://www.dpi.ufv.br/projetos/scheduling/smtt.htm.

4.5. Experimental Results on Small-Size Instances. In this subsection the best ILS heuristic (i.e., ILS_DP+PR) is compared with CPLEX solver on the 216 small-size instances. This solver is applied to solve the MILP formulation (presented in Section 2) with a threshold CPU time of 1800 seconds for each small instance. That is, if after the established time no optimal solution is obtained, the best current solution (upper bound) is returned by CPLEX. Each small instance is solved 30 times by ILS_DP+PR heuristic using $0.5 * n$ seconds as stopping

Table 10: Relative percentage improvement of ILS_DP+PR with respect to CPLEX.

(n, F, r, ST)	ILS_DP+PR	CPLEX	RPI
(20, 4, 1.5, M)	710	826	14.04%
(20, 6, 0.5, M)	7458	7972	6.44%
(20, 6, 1.5, M)	965	1125	14.22%
(20, 6, 1.5, L)	2589	3272	20.87%
(20, 6, 2.5, L)	1965	2128	7.66%
Average	2737.4	3064.6	12.64%

Table 11: Average CPU times (in seconds) of CPLEX and ILS_DP+PR.

Instance	CPU times (s)	
$n \times F$	CPLEX	ILS_DP+PR
10×2	81.20	5.00
10×4	105.8	5.00
10×6	94.02	5.00
15×2	314.23	7.00
15×4	393.62	7.00
15×6	358.82	7.00
20×2	835.83	10.00
20×4	931.20	10.00
20×6	1029.11	10.00

criterion. In the comparison analysis, for each instance, we consider the best solution obtained by each heuristic. The CPLEX is only run one time for each each instance, because it uses a deterministic method. Since the used computer has 12 cores, the CPLEX is run using all the cores and the heuristic is run using a single core.

For a total of 216 small instances, 201 optimal solutions were generated by CPLEX, from which all these optimal solutions were found by ILS_DP+PR heuristic. On 11 instances with $n = 20$ jobs, CPLEX was not able to generate a feasible solution, and on 5 instances ILS_DP+PR was better than CPLEX. Table 10 shows the results for these 5 instances. This table reports the factors of the instances (first column), the total tardiness (TT) values obtained by ILS_DP+PR and CPLEX (second and third columns, resp.), and the relative percentage improvement (RPI) of ILS_DP+PR with respect to CPLEX (final column). The RPI is determined by $100 \times (\text{TT}_{\text{CPLEX}} - \text{TT}_{\text{ILS\_DP+PR}})/(\text{TT}_{\text{CPLEX}})\%$. As can be seen, the solutions obtained by ILS_DP+PR are on average 12.64% better than the solutions (upper bounds) generated by CPLEX. The improvement of ILS_DP+PR varies from 6.44% to 20.87%.

The average CPU times spent by CPLEX and ILS_DP+PR are presented in Table 11. The CPU times of ILS_DP+PR are equal to $0.5 * n$ seconds. We can see that CPLEX spends much more CPU time than ILS_DP+PR for all small instances. When the number of jobs is increased, the performance of CPLEX (considering solution quality and computational time) worsens significantly. We tested CPLEX solver on instances with $n = 30$ jobs. For most of these instances

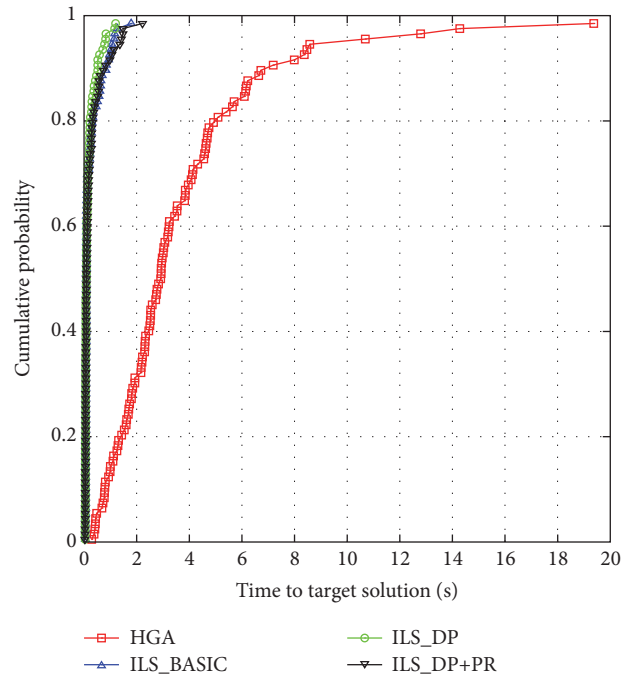

Figure 7: TTT plots of HGA, ILS_BASIC, ILS_DP, and ILS_DP+PR.

CPLEX was not able to find feasible solutions in the established CPU time. Thus, it can be asserted that to solve large instances of the problem under study the most appropriate methods are heuristic algorithms, although they do not guarantee finding optimal solutions.

4.6. Time Analysis. The experiments presented in previous subsections were done considering solution quality of the heuristics. To give additional information concerning the computational effort required by the heuristic algorithms, we use the time-to-target (TTT) plot analysis [57]. A TTT plot is generated by executing an algorithm q times and measuring the computational time required to reach a solution at least as good as a target solution. The running times are sorted in increasing order. The ith running time t_i is associated with a probability $P_i = (i - 0.5)/q$ and the points (t_i, P_i), $i = 1, \ldots, q$, are plotted. Each plotted point indicates the probability (vertical axis) for the algorithm to achieve the target solution in a given running time (horizontal axis). The more to the left is the plot, the better is the corresponding algorithm.

For this analysis we used an instance with 100 jobs and 4 families, where the heuristic algorithms HGA, ILS_BASIC, ILS_DP, and ILS_DP+PR found the best known solution (target solution) at least one time. We performed 100 runs for each algorithm, varying the random number generator seed. The algorithms were made to stop whenever a solution better than or equal to the target solution was found. Figure 7 displays the TTT plot for the four algorithms. It can be seen that the proposed ILS algorithms find solutions as good as the target solution clearly faster than HGA. All ILS algorithms have similar behaviors and a 2-second run time is enough to ensure a 98% probability of obtaining the target solution. The probability of HGA finding the target solution in 2 seconds

is about 20%. Moreover, within 14 seconds, HGA reaches the target solution with a maximum probability of 98%. This analysis indicates that the proposed ILS algorithms converge to good solutions with short CPU times.

5. Conclusions

In this paper we have considered a single-machine scheduling problem with sequence-dependent family setup times, so as to minimize the total tardiness. The main contribution of this work was the development of three effective heuristics based on ILS metaheuristic to solve the $1|ST_{sd,f}|\sum T_j$ problem. The ILS_DP extended the standard ILS_BASIC heuristic by using a dynamic perturbation size. Moreover, the ILS_DP+PR heuristic extended ILS_DP by using a Path Relinking intensification procedure that keeps a set of elite solutions. To the best of our knowledge, ILS heuristic has not been applied to solve single-machine scheduling problems with sequence-dependent family setup times. To improve the efficiency of the proposed heuristics, their parameters were fine-tuned using a Design of Experiments methodology. The heuristics were compared on the basis of computational experiments performed on a comprehensive set of large-size problem instances. The experiments demonstrated the effectiveness of our ILS heuristics. They outperformed a hybrid genetic algorithm (HGA) consistently. The heuristics ILS_DP and ILS_DP+PR performed better than the basic ILS (ILS_BASIC); that is, the results of the standard ILS were improved significantly by using a dynamic perturbation size and Path Relinking intensification. All the obtained results have been statistically validated.

We believe that the LS_DP+PR heuristic presented in this paper is a significant contribution, worthy of future study. Future research is to apply the proposed heuristic to other single-machine scheduling problems, for example, the problem with sequence-dependent family setup times and resource constraints, as the one studied very recently by Herr and Goel [34]. Other objectives could also be explored, such as the total completion times and total earliness and tardiness.

Competing Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors are thankful to the financial support of CNPq and FAPEMIG, Brazilian research agencies.

References

[1] A. Allahverdi, C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov, "A survey of scheduling problems with setup times or costs," *European Journal of Operational Research*, vol. 187, no. 3, pp. 985–1032, 2008.

[2] A. Allahverdi, "The third comprehensive survey on scheduling problems with setup times/costs," *European Journal of Operational Research*, vol. 246, no. 2, pp. 345–378, 2015.

[3] J.-B. Wang and J.-J. Wang, "Single machine group scheduling with time dependent processing times and ready times," *Information Sciences*, vol. 275, pp. 226–231, 2014.

[4] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[5] J. Du and J. Y. Leung, "Minimizing total tardiness on one machine is NP-hard," *Mathematics of Operations Research*, vol. 15, no. 3, pp. 483–495, 1990.

[6] S. Chantaravarapan, J. N. D. Gupta, and M. L. Smith, "A hybrid genetic algorithm for minimizing total tardiness on a single machine with family setups," in *Proceedings of the Production and Operations Management Society Meeting (POMS '03)*, pp. 1–35, Savannah, Ga, USA, April 2003.

[7] P. A. Rubin and G. L. Ragatz, "Scheduling in a sequence dependent setup environment with genetic search," *Computers & Operations Research*, vol. 22, no. 1, pp. 85–99, 1995.

[8] V. A. Armentano and R. Mazzini, "A genetic algorithm for scheduling on a single machine with set-up times and due dates," *Production Planning & Control*, vol. 11, no. 7, pp. 713–720, 2000.

[9] P. M. França, A. Mendes, and P. Moscato, "A memetic algorithm for the total tardiness single machine scheduling problem," *European Journal of Operational Research*, vol. 132, no. 1, pp. 224–242, 2001.

[10] A. Sioud, M. Gravel, and C. Gagné, "A hybrid genetic algorithm for the single machine scheduling problem with sequence-dependent setup times," *Computers & Operations Research*, vol. 39, no. 10, pp. 2415–2424, 2012.

[11] K. C. Tan and R. Narasimhan, "Minimizing tardiness on a single processor with sequence-dependent setup times: a simulated annealing approach," *Omega*, vol. 25, no. 6, pp. 619–634, 1997.

[12] C. Gagné, W. L. Price, and M. Gravel, "Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times," *Journal of the Operational Research Society*, vol. 53, no. 8, pp. 895–906, 2002.

[13] C.-J. Liao and H.-C. Juan, "An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups," *Computers & Operations Research*, vol. 34, no. 7, pp. 1899–1909, 2007.

[14] S. R. Gupta and J. S. Smith, "Algorithms for single machine total tardiness scheduling with sequence dependent setups," *European Journal of Operational Research*, vol. 175, no. 2, pp. 722–739, 2006.

[15] H. Akrout, B. Jarboui, P. Siarry, and A. Rebaï, "A GRASP based on DE to solve single machine scheduling problem with SDST," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 411–435, 2012.

[16] J. E. C. Arroyo, G. V. P. Nunes, and E. H. Kamke, "Iterative local search heuristic for the single machine scheduling problem with sequence dependent setup times and due dates," in *Proceedings of the 9th International Conference on Hybrid Intelligent Systems (HIS '09)*, vol. 1, pp. 505–510, IEEE, Shenyang, China, August 2009.

[17] K.-C. Ying, S.-W. Lin, and C.-Y. Huang, "Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic," *Expert Systems with Applications*, vol. 36, no. 3, pp. 7087–7092, 2009.

[18] G. L. Ragatz, "A branch-and-bound method for minimum tardiness sequencing on a single processor with sequence

dependent setup times," in *Proceedings of the 24th Annual Meeting of the Decision Sciences Institute*, pp. 1375–1377, John Wiley & Sons, 1993.

[19] X. Luo and F. Chu, "A branch and bound algorithm of the single machine schedule with sequence dependent setup times for minimizing total tardiness," *Applied Mathematics and Computation*, vol. 183, no. 1, pp. 575–588, 2006.

[20] E. C. Sewell, J. J. Sauppe, D. R. Morrison, S. H. Jacobson, and G. K. Kao, "A BB'R algorithm for minimizing total tardiness on a single machine with sequence dependent setup times," *Journal of Global Optimization*, vol. 54, no. 4, pp. 791–812, 2012.

[21] S. Tanaka and M. Araki, "An exact algorithm for the single-machine total weighted tardiness problem with sequence-dependent setup times," *Computers & Operations Research*, vol. 40, no. 1, pp. 344–352, 2013.

[22] D. Kacem, "Lower bounds for tardiness minimization on a single machine with family setup times," in *Proceedings of the IMACS Multiconference on Computational Engineering in Systems Applications*, vol. 1, pp. 1034–1039, Beijing, China, 2006.

[23] J. Schaller, "Scheduling on a single machine with family setups to minimize total tardiness," *International Journal of Production Economics*, vol. 105, no. 2, pp. 329–344, 2007.

[24] J. C.-H. Pan and C.-S. Su, "Single machine scheduling with due dates and class setups," *Journal of the Chinese Institute of Engineers*, vol. 20, no. 5, pp. 561–572, 1997.

[25] K. R. Baker and M. J. Magazine, "Minimizing maximum lateness with job families," *European Journal of Operational Research*, vol. 127, no. 1, pp. 126–139, 2000.

[26] J. E. Schaller and J. N. Gupta, "Single machine scheduling with family setups to minimize total earliness and tardiness," *European Journal of Operational Research*, vol. 187, no. 3, pp. 1050–1068, 2008.

[27] J. N. D. Gupta and S. Chantaravarapan, "Single machine group scheduling with family setups to minimize total tardiness," *International Journal of Production Research*, vol. 46, no. 6, pp. 1707–1722, 2008.

[28] J. A. van der Veen, G. J. Woeginger, and S. Zhang, "Sequencing jobs that require common resources on a single machine: a solvable case of the TSP," *Mathematical Programming*, vol. 82, no. 1-2, pp. 235–254, 1998.

[29] S. Karabati and C. Akkan, "Minimizing sum of completion times on a single machine with sequence-dependent family setup times," *Journal of the Operational Research Society*, vol. 57, no. 3, pp. 271–280, 2006.

[30] F. Jin, S. Song, and C. Wu, "A simulated annealing algorithm for single machine scheduling problems with family setups," *Computers & Operations Research*, vol. 36, no. 7, pp. 2133–2138, 2009.

[31] F. Jin, S. Shiji, and W. Cheng, "Dominance property based tabu search for single machine scheduling problems with family setups," *Journal of Systems Engineering and Electronics*, vol. 20, no. 6, pp. 1233–1238, 2009.

[32] F. Jin, J. N. D. Gupta, S. Song, and C. Wu, "Single machine scheduling with sequence-dependent family setups to minimize maximum lateness," *Journal of the Operational Research Society*, vol. 61, no. 7, pp. 1181–1189, 2010.

[33] V. Sels and M. Vanhoucke, "A hybrid genetic algorithm for the single machine maximum lateness problem with release times and family setups," *Computers & Operations Research*, vol. 39, no. 10, pp. 2346–2358, 2012.

[34] O. Herr and A. Goel, "Minimising total tardiness for a single machine scheduling problem with family setups and resource constraints," *European Journal of Operational Research*, vol. 248, no. 1, pp. 123–135, 2016.

[35] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, and D. Van Oudheusden, "Iterated local search for the team orienteering problem with time windows," *Computers & Operations Research*, vol. 36, no. 12, pp. 3281–3290, 2009.

[36] P. H. V. Penna, A. Subramanian, and L. S. Ochi, "An iterated local search heuristic for the heterogeneous fleet vehicle routing problem," *Journal of Heuristics*, vol. 19, no. 2, pp. 201–232, 2013.

[37] I. Ribas, R. Companys, and X. Tort-Martorell, "An efficient iterated local search algorithm for the total tardiness blocking flow shop problem," *International Journal of Production Research*, vol. 51, no. 17, pp. 5238–5252, 2013.

[38] P. Vansteenwegen and M. Mateo, "An iterated local search algorithm for the single-vehicle cyclic inventory routing problem," *European Journal of Operational Research*, vol. 237, no. 3, pp. 802–813, 2014.

[39] J. Brandão, "A deterministic iterated local search algorithm for the vehicle routing problem with backhauls," *TOP*, vol. 24, no. 2, pp. 445–465, 2016.

[40] X. Dong, H. Huang, and P. Chen, "An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion," *Computers & Operations Research*, vol. 36, no. 5, pp. 1664–1669, 2009.

[41] L. Fanjul-Peyro and R. Ruiz, "Iterated greedy local search methods for unrelated parallel machine scheduling," *European Journal of Operational Research*, vol. 207, no. 1, pp. 55–69, 2010.

[42] F. Della Croce, T. Garaix, and A. Grosso, "Iterated local search and very large neighborhoods for the parallel-machines total tardiness problem," *Computers & Operations Research*, vol. 39, no. 6, pp. 1213–1217, 2012.

[43] H. Xu, Z. Lü, and T. C. Cheng, "Iterated local search for single-machine scheduling with sequence-dependent setup times to minimize total weighted tardiness," *Journal of Scheduling*, vol. 17, no. 3, pp. 271–287, 2014.

[44] X. Dong, M. Nowak, P. Chen, and Y. Lin, "Self-adaptive perturbation and multi-neighborhood search for iterated local search on the permutation flow shop problem," *Computers & Industrial Engineering*, vol. 87, pp. 176–185, 2015.

[45] V. L. A. Santos, J. E. C. Arroyo, and T. F. Carvalho, "Iterated local search based heuristic for scheduling jobs on unrelated parallel machines with machine deterioration effect," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 53–54, ACM, Denver, Colo, USA, July 2016.

[46] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search," in *Handbook of Metaheuristics*, F. Glover and G. A. Kochenberger, Eds., vol. 57 of *International Series in Operations Research & Management Science*, chapter 11, pp. 320–353, Kluwer Academic, 2003.

[47] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search: framework and applications," in *Handbook of Metaheuristics*, pp. 363–397, Springer, Berlin, Germany, 2010.

[48] P. Hansen and N. Mladenović, "Variable neighborhood search," in *Search Methodologies*, pp. 313–337, Springer, Berlin, Germany, 2014.

[49] M. Laguna and R. Martí, "GRASP and path relinking for 2-layer straight line crossing minimization," *INFORMS Journal on Computing*, vol. 11, no. 1, pp. 44–52, 1999.

[50] F. Glover, "Tabu search and adaptive memory programing-advances, applications and challenges," in *Interfaces in Computer Science and Operations Research*, R. S. Barr, R. V. Helgason, and J. L. Kennington, Eds., Operations Research/Computer Science Interfaces Series, pp. 1–75, Springer, New York, NY, USA, 1996.

[51] M. Nawaz, E. E. Enscore Jr., and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[52] M. G. C. Resende and C. C. Ribeiro, "Handbook of metaheuristics," in *Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications*, M. Gendreau and J. Y. Potvin, Eds., pp. 219–249, Springer, Berlin, Germany, 2nd edition, 2010.

[53] E. Vallada, R. Ruiz, and G. Minella, "Minimising total tardiness in the m-machine flowshop problem: a review and evaluation of heuristics and metaheuristics," *Computers & Operations Research*, vol. 35, no. 4, pp. 1350–1373, 2008.

[54] D. C. Montgomery, *Design and Analysis of Experiments*, John Wiley & Sons, New York, NY, USA, 2006.

[55] M. den Besten and T. Stützle, "Neighborhoods revisited: an experimental investigation into the effectiveness of variable neighborhood descent for scheduling," in *Proceedings of the 4th Metaheuristics International Conference*, pp. 545–549, 2001.

[56] A. Subramanian, M. Battarra, and C. N. Potts, "An Iterated Local Search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times," *International Journal of Production Research*, vol. 52, no. 9, pp. 2729–2742, 2014.

[57] R. M. Aiex, M. G. Resende, and C. C. Ribeiro, "Ttt plots: a perl program to create time-to-target plots," *Optimization Letters*, vol. 1, no. 4, pp. 355–366, 2007.