Computers & Operations Research 39 (2012) 1506–1520

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

A variable neighborhood search for minimizing total weighted tardiness with sequence dependent setup times on a single machine

Gokhan Kirlik, Ceyda Oguz *

College of Engineering, Koç University, Istanbul, Turkey

ARTICLE INFO

Available online 6 September 2011

Keywords:
Single machine scheduling
Weighted tardiness
Sequence dependent setup time
Variable neighborhood search

ABSTRACT

This paper deals with the single machine scheduling problem to minimize the total weighted tardiness in the presence of sequence dependent setup. Firstly, a mathematical model is given to describe the problem formally. Since the problem is NP-hard, a general variable neighborhood search (GVNS) heuristic is proposed to solve it. Initial solution for the GVNS algorithm is obtained by using a constructive heuristic that is widely used in the literature for the problem. The proposed algorithm is tested on 120 benchmark instances. The results show that 37 out of 120 best known solutions in the literature are improved while 64 instances are solved equally. Next, the GVNS algorithm is applied to single machine scheduling problem with sequence dependent setup times to minimize the total tardiness problem without changing any implementation issues and the parameters of the GVNS algorithm. For this problem, 64 test instances are solved varying from small to large sizes. Among these 64 instances, 35 instances are solved to the optimality, 16 instances' best-known results are improved, and 6 instances are solved equally compared to the best-known results. Hence, it can be concluded that the GVNS algorithm is an effective, efficient and a robust algorithm for minimizing tardiness on a single machine in the presence of setup times.

1. Introduction

Majority of scheduling problems deal with environments where there are no setup times. Such environments are modeled with the assumption that either setup times are small compared to processing times of jobs so that they can be ignored or setup times can be independent of job processing sequence so that they are added to the processing times of jobs. However, substantial setup times are required between the processing of two jobs in some industries and the amount of setup time for a job depends on the preceding job in the processing order. In these cases, setup times should be considered explicitly. Such setup times are called sequence dependent setup times (SDST) and exist in industries that require, for example, cleaning, molding, painting and printing operations [1].

Different performance measures have been studied in the literature for scheduling problems with SDST. Since meeting the due dates is important in competitive markets, due date related performance measures are gaining importance. Among others, minimizing total weighted tardiness (TWT) has been drawing considerable attention of the researchers in the past decades since it differentiates between customers.

In this paper, we deal with the problem of scheduling jobs with sequence dependent setup times on a single machine to minimize the total weighted tardiness (SMTWT-SDST). The problem can be stated formally as follows: there exists a single machine on which a set of n independent jobs should be processed. Each job $j=1,\ldots,n$ is available at time zero and characterized with a processing time p_j, a due date d_j, a weight w_j that represents the priority of job, and a sequence dependent setup time s_{ij} if job j is processed immediately after job i. The machine is continuously available and can process only one job at a time without preemption. Suppose that $\pi=\{\pi_1,\ldots,\pi_n\}$ is a processing sequence of n jobs, where π_k is the job which is processed in the kth order of the processing sequence, $k=1,\ldots,n$. The completion time of job π_k can be computed as $C_k=C_{k-1}+s_{\pi_{k-1}\pi_k}+p_{\pi_k}$. When $k=1$, $C_0=0$ and $s_{\pi_0\pi_1}$ represent the initial setup time of job π_1. The tardiness of job π_k is $T_k=\max(0,C_k-d_{\pi_k})$. The scheduling objective, that is the minimization of the total weighted tardiness, is expressed as $z_\pi=\sum_{k=1}^{n} w_{\pi_k}T_k$. In three-field notation [2], this problem is represented as $1|s_{ij}|\sum w_jT_j$.

Even though $1|s_{ij}|\sum w_jT_j$ can be used to model many real life problems, it has received much less attention in the literature compared to single machine TWT ($1||\sum w_jT_j$) problem. This can be attributed to the complex nature of the problem due to SDST beside the performance measure used. The $1|s_{ij}|\sum w_jT_j$ problem is NP-hard since its simpler version, that is $1||\sum w_jT_j$, has been shown to be NP-hard [3,4].

* Corresponding author.
E-mail addresses: gkirlik@ku.edu.tr (G. Kirlik), coguz@ku.edu.tr (C. Oguz).

0305-0548/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2011.08.022

In the literature, both exact and heuristic algorithms have been proposed for the single machine total weighted tardiness problem [5–7]. However, exact algorithms can solve only small sized instances due to the computational complexity of the problem. Hence, the design of heuristic algorithms became the main aspect in recent studies. These heuristic algorithms can be classified as constructive and improvement heuristics. A constructive heuristic forms a sequence by assigning a job to the current position at each step of the algorithm according to a dispatching rule. A dispatching rule can be either static, such as the earliest due date (EDD), or dynamic, such as apparent tardiness cost (ATC) rule. ATC rule was proposed for the single machine total weighted tardiness problem in [8] and was extended for the $1|s_{ij}|\sum w_jT_j$ problem by considering sequence dependent setup times (apparent tardiness cost with setups (ATCS) rule) [9]. Even though ATCS rule is the best performing one among constructive heuristics for the $1|s_{ij}|\sum w_jT_j$ problem in the literature, its solution quality becomes unsatisfactory for large sized problems. An improvement heuristic starts with an initial solution and improves it by changing the sequence at each step according to certain rules. In [10], several heuristics were compared for single machine total tardiness problem and it was shown that the pairwise interchange rule outperforms the others. In [9], swap and insertion based hill-climbing search procedures were proposed for the $1|s_{ij}|\sum w_jT_j$ problem.

In addition to above constructive and improvement heuristics, metaheuristics have been extensively used in the literature to solve the $1|s_{ij}|\sum w_jT_j$ problem. Cicirello and Smith [11] generated 120 problem instances, each with 60 jobs, for the $1|s_{ij}|\sum w_jT_j$ problem and analyzed the effectiveness of stochastic sampling approaches, such as value-biased stochastic sampling (VBSS), a VBSS with hill-climbing, limited discrepancy search, and heuristic-biased stochastic sampling, together with a simulated annealing (SA) approach for the $1|s_{ij}|\sum w_jT_j$ problem. In [12] and [13], ant colony optimization (ACO) algorithms were proposed. Lin and Ying compared results of three different metaheuristics including Genetic Algorithm (GA), Tabu Search (TS) and SA [14]. In [15], the results of [11] were improved by means of a GA approach by introducing a non-wrapping order crossover. Later, a beam search with variable beam and filter widths was proposed in [16]. In addition, a discrete particle swarm optimization algorithm (DPSO) was presented with the best known results [17]. Recently, a discrete differential evolution (DDE) algorithm was developed for the $1|s_{ij}|\sum w_jT_j$ problem and the previous best known solutions were improved with excellent results [18].

The status of the literature motivated us to develop a heuristic algorithm with the aim of producing solutions which are closer to the optimal ones for the $1|s_{ij}|\sum w_jT_j$ problem. To this end, in this paper, we propose a general variable neighborhood search algorithm for the $1|s_{ij}|\sum w_jT_j$ problem, which performs significantly better in terms of the solution quality without degrading the run time compared to existing heuristics. Given the successful performance of the algorithm, we then analyzed it on the $1|s_{ij}|\sum T_j$ problem.

The remainder of the paper is organized as follows. In Section 2, a mathematical programming formulation of the $1|s_{ij}|\sum w_jT_j$ problem is given. After explaining the general variable neighborhood search in Section 3, the proposed implementation of the general variable neighborhood search algorithm is given in Section 4. The computational results of the proposed method are presented in Section 5. Finally, conclusions are stated in Section 6.

2. Mathematical programming formulation

The $1|s_{ij}|\sum w_jT_j$ problem can be formulated as a mixed integer linear programming model by defining the following decision variables and considering the notation given earlier:

Decision Variables

$$x_{jk}=\begin{cases}1 & \text{if job } j \text{ is processed at the } k\text{th position,}\\ 0 & \text{otherwise.}\end{cases}$$

$$y_{ijk}=\begin{cases}1 & \text{if job } j \text{ is processed after job } i \text{ at position } k,\\ 0 & \text{otherwise.}\end{cases}$$

Below we present the mixed integer linear programming formulation of the $1|s_{ij}|\sum w_jT_j$ problem for the sake of completeness.

Model

$$\min \sum_{j=1}^{n} w_jT_j$$

s.t.

$$\sum_{k=1}^{n} x_{jk}=1 \quad \forall j \tag{1}$$

$$\sum_{j=1}^{n} x_{jk}=1 \quad \forall k \tag{2}$$

$$C_j \ge (s_{0j}+p_j)x_{jk} \quad \forall j \text{ and } k=1 \tag{3}$$

$$C_j \ge C_i - M(1-y_{ijk})+s_{ij}+p_j \quad \forall i,j,k \quad \text{where } i\ne j \text{ and } k\ge 2 \tag{4}$$

$$T_j \ge C_j-d_j \quad \forall j \tag{5}$$

$$x_{i(k-1)}+x_{jk} \le y_{ijk}+1 \quad \forall i,j,k \text{ where } i\ne j \text{ and } k\ge 2 \tag{6}$$

$$x_{jk}\in\{0,1\} \quad \forall j,k \tag{7}$$

$$y_{ijk}\in\{0,1\} \quad \forall i,j,k \tag{8}$$

$$C_j,T_j \ge 0 \quad \forall j \tag{9}$$

In this model, the objective function minimizes the total weighted tardiness. Constraint set (1) ensures that each job is assigned to exactly one position. Similarly, constraint set (2) enforces that each position in the sequence is assigned to exactly one job. Constraint set (3) represents the completion time of job j which is assigned to the first position ($k=1$) in the sequence. Constraint set (4) calculates the completion time of job j, C_j. Constraint set (5) defines the tardiness of the jobs. Constraint set (6) is used to interrelate two different sets of decision variables. Constraint sets (7) and (8) are integrality constraints, whereas constraint set (9) defines the continuous variables.

As stated in Section 1, the $1|s_{ij}|\sum w_jT_j$ problem is NP-hard and hence solving the above mathematical model to obtain a solution to the problem is not practical. In the following two sections, we present the general variable neighborhood search algorithm which is developed to find better solutions compared to the ones obtained by the algorithms given in the literature.

3. General variable neighborhood search

In order to explain the variable neighborhood search (VNS), let us consider a general deterministic optimization problem, which can be represented formally as $\min\{f(x)|x\in X, X\subseteq S\}$, where S, X, x and f denote the solution space, the feasible solution set, the feasible solution and the objective function, respectively. If S is a finite but large set, then a combinatorial optimization problem, such as $1|s_{ij}|\sum w_jT_j$, is defined. Since most of the combinatorial optimization problems are NP-hard, metaheuristics are widely used to solve especially large sized instances.

VNS is a metaheuristic algorithm which uses the idea of changing the neighborhood systematically and has been successfully applied to different combinatorial optimization problems. The basic scheme of the VNS was proposed by Mladenović and Hansen [19]. Its advanced principles for solving combinatorial optimization problems and applications were further introduced in [20–22] and recently in [23].

VNS uses a finite set of pre-selected neighborhood structures denoted as N_k ($k=1,\ldots,k_{max}$). $N_k(x)$ denotes the set of solutions in the kth neighborhood of solution x. VNS employs a local search to obtain a solution $x \in X$, called as a local minimum, such that there exists no solution $x' \in N_k(x) \subseteq X$ with $f(x') < f(x)$. The local search can be performed in different ways. The generic way consists of choosing an initial solution x, finding a direction of descent from x within a neighborhood $N(x)$, and moving to the minimum of $f(x)$ within $N(x)$ in the same direction. If there is no direction of descent, the heuristic stops; otherwise, it is iterated. Usually the steepest direction of descent, also referred to as the best improvement, is used. The steps of the best improvement are given in Algorithm 1.

Algorithm 1. BestImprovement.

```
Input: x (initial solution)
Output: x (local minimum)
repeat
| x' ← x
| x ← arg min_{y ∈ N_k(x)} f(y)
until f(x) ≥ f(x')
```

After the local search, a change in the neighborhood structure is performed. The generic form of the neighborhood change function is given in Algorithm 2. Function `NeighborhoodChange` compares the value $f(x')$ of a new solution x' with the value $f(x)$ of the incumbent solution x obtained in the neighborhood k. If an improvement is obtained, k is returned to its initial value and the incumbent solution is updated with the new one. Otherwise, the next neighborhood is considered.

Algorithm 2. NeighborhoodChange.

```
input: x,x',k
output: x,k
if f(x') < f(x) then
| x ← x'
| k ← 1
else
| k ← k+1
```

The VNS can be summarized as in Algorithm 3. VNS uses two parameters: t_{max}, which is the maximum time allowed as the stopping condition, and k_{max}, which is the number of neighborhood structures used. Step 4 of Algorithm 3, which is called *shaking*, randomly chooses a solution x' from the kth neighborhood of the incumbent solution x. After improving this solution via the `BestImprovement` local search (Algorithm 1), a neighborhood change is employed with the function `NeighborhoodChange` (Algorithm 2).

Algorithm 3. Variable Neighborhood Search.

```
  input: x, k_max, t_max
  output: x
1 repeat
2 | k ← 1
3 | repeat
4 | | x' ← Shake(x,k)
5 | | x'' ← BestImprovement(x')
6 | | NeighborhoodChange(x,x'',k)
7 | until k = k_max
8 | t ← CpuTime ()
9 unit t > t_max
```

If we eliminate the randomness in VNS, then the Variable Neighborhood Descent (VND) is obtained (Algorithm 4). While this deterministic variant of VNS can be used as it is, it might be useful as a local search within a VNS. In the latter case, one can obtain a better solution at the end of the local search since VND itself uses more than one neighborhood structure. Hence the chances to reach a global solution are larger when using VND rather than a single neighborhood structure [23].

Algorithm 4. Variable Neighborhood Descent.

```
input: x,k_max
output: x
k ← 1
repeat
| x' ← arg min_{y ∈ N_k(x)} f(y)
| NeighborhoodChange(x,x',k)
until k = k_max
```

As a variant of the VNS, if the local search step of the VNS is replaced by VND, then we obtain the general VNS (GVNS) [24]. Steps of the GVNS are given in Algorithm 5. GVNS uses one additional parameter other than t_{max} and k_{max}, that is k'_{max}, the number of neighborhoods used in the inner VND loop.

Algorithm 5. General Variable Neighborhood Search.

```
input: x, k_max, k'_max, t_max
output: x
repeat
| k ← 1
| repeat
| | x' ← Shake(x,k)
| | x'' ← VND(x',k'_max)
| | NeighborhoodChange(x,x'',k)
| until k = k_max
| (t ← CpuTime ()
until t > t_max
```

4. Implementation of GVNS for $1|s_{ij}|\sum w_j T_j$

In this section, we describe the details of the GVNS algorithm as it is implemented to solve the $1|s_{ij}|\sum w_j T_j$ problem. As explained in Section 3, in GVNS approach (Algorithm 5), the local search step of the VNS algorithm is replaced with the VND algorithm. Shaking step of the GVNS algorithm consists of two ($k_{max}=2$) different random neighborhood structures which will be explained in Section 4.3. In the VND step, three ($k'_{max}=3$) neighborhood structures are used. As VND is a deterministic metaheuristic, the best solution in each neighborhood is determined. In the following subsections solution representation, initial solution generation, neighborhood structures and local search procedures are detailed.

4.1. Solution representation

In most of the scheduling problems where there are no idle times between jobs, a permutation of jobs is the typical solution representation for any metaheuristic algorithm. In GVNS, we follow the same idea and represent the solution by the processing sequence of jobs which is denoted by $\pi=\{\pi_1,\ldots,\pi_k,\ldots,\pi_n\}$, where π_k is the job which is processed in the kth order.

4.2. Initial solution generation

Since GVNS is a trajectory-based metaheuristic, we need to start from a given solution. In this study, we used the ATCS heuristic [9] to obtain a reasonably good starting solution for the $1|s_{ij}|\sum w_jT_j$ problem. Essentially, ATCS heuristic generates a schedule by using the job order priority, which considers weights, sequence dependent setups and due dates.

ATCS heuristic consists of two stages. The first stage of ATCS heuristic includes performing a statistical analysis of the problem instances. The result of this stage is the estimation of three factors, namely due date tightness, due date range and setup time severity factor, that define the problem instances and their respective makespan value. This stage can be viewed as a pre-processing stage. In the second stage, by using these estimates, the look-ahead parameter values, that is, k_1 and k_2 are calculated. These values are then used to calculate a priority index, which determines the sequence of the jobs. A solution to the $1|s_{ij}|\sum w_jT_j$ problem is then obtained by assigning the jobs to the machine at the earliest time point available.

Due date tightness τ, due date range R and setup time severity factor η are estimated from the following equations, respectively:

$$\tau=1-\frac{\overline{d}}{C_{max}},\quad R=\frac{d_{max}-d_{min}}{C_{max}},\quad \text{and}\quad \eta=\frac{\overline{s}}{\overline{p}}$$

In above equations, C_{max} is the makespan, which is the completion time of the last job in the sequence, $\overline{d}$ is the average of the due dates, d_{max} and d_{min} represent the maximum and the minimum due date values, respectively, $\overline{s}$ denotes the average setup time and $\overline{p}$ stands for the average processing time. In estimating both τ and R, the value of C_{max} is needed, however, due to the sequence dependent setup times, determining the makespan value beforehand is difficult. Hence one needs to estimate the C_{max} value, too. Such an estimate can be obtained by correlating the C_{max} value with the average processing time, the average setup time and a coefficient β:

$$C_{max}=n(\overline{p}+\beta\overline{s})$$

Variability of setup times and the number of jobs in the instance would affect the value of β [9]. By using the estimates of τ, R and η, the parameters k_1 and k_2 are calculated as follows:

$$k_1=\begin{cases}4.5+R, & R\le 0.5\\ 6.0-2R, & R>0.5\end{cases}$$

$$k_2=\frac{\tau}{2\sqrt{\eta}}$$

Finally, the priority index is determined with the following equation in the ATCS heuristic:

$$I_j(t,i)=\frac{w_j}{p_j}\exp\left[-\frac{\max(d_j-p_j-t,0)}{k_1\overline{p}}\right]\exp\left[-\frac{s_{ij}}{k_2\overline{s}}\right] \tag{10}$$

In Eq. (10), t denotes the current time and i is the index of the job just completed. The ATCS rule separates the effect of the setup time. The priority of a job given by the weighted shortest processing time ratio is exponentially discounted twice, once based on the slack and again based on the setup time. These two effects are scaled separately by the parameters k_1 and k_2, which jointly provide the look-ahead capabilities of the ATCS rule. The values of the parameters depend on the problem instance as they essentially perform the scaling.

We note that we analyzed the effect of a random solution as the initial solution and observed that while the performance of the GVNS does not change with respect to the solution quality, it is slightly affected regarding the CPU time. Hence we can say that the proposed GVNS is not sensitive to the initial solution for the $1|s_{ij}|\sum w_jT_j$ problem. Since, the ATCS rule provides a good starting solution to the algorithm, the GVNS algorithm converges to a desired solution faster than the random initial solution.

4.3. Neighborhood structures

The literature indicates that three problem specific decisions should be made to obtain an efficient VNS [19]. Firstly, we have to decide which neighborhoods to use. Secondly, the order of these neighborhoods in the search process should be decided. Thirdly, we have to identify the search strategy to be used in changing neighborhoods. We address the first question below. We studied the remaining two questions at the implementation stage of the GVNS for the $1|s_{ij}|\sum w_jT_j$ problem via computational experiments and explain our findings in Section 5.1.

To address which neighborhoods to use, we tested four different neighborhood structures to search through the solution space, which are swap, insertion, edge-insertion and 2-edge exchange. These neighborhood structures have been widely used in the literature for different combinatorial optimization problems and have been shown to work well especially for those problems which have a permutation representation. In GVNS, we can obtain a random solution at the shaking step by using swap, insertion, edge-insertion and 2-edge exchange neighborhoods. The steps of these random moves are given in the Appendix (Algorithms 6, 7, 8, and 9, respectively).

We experimented with the same neighborhood structures further in the VND step of the GNVS. Note that VND (Algorithm 4) determines the best neighborhood of the input solution in its third step. This is known as the steepest hill climbing or best improvement in the literature. We refer to these best improvement algorithms as `BestSwapMove`, `BestInsertionMove`, `BestEdgeInsertionMove`, and `2−OptMove` and the details of these algorithms are given in the Appendix (Algorithms 10, 11, 13, and 12, respectively). All these algorithms have a time complexity of $O(n^2)$. In `BestSwapMove`, `Swap`(π_{k_1},π_{k_2}) function exchanges the jobs π_{k_1} and π_{k_2}, while in `BestInsertionMove`, `Insert`(π_{k_1},k_2) function removes job π_{k_1} from position k_1 and inserts it to the position k_2. In `BestEdge−InsertionMove`, `EdgeInsertion`$((\pi_{k_1},\pi_{k_1+1}),k_2)$ function removes the consecutive jobs (π_{k_1},π_{k_1+1}) and inserts them to the position k_2 in the sequence.

Best neighbor of the 2-edge exchange move is called as 2-Opt. This local search is a well-known improvement heuristic for the traveling salesman problem. In `2−OptMove`, `EdgeExchange`$((\pi_{k_1},\pi_{k_1+1}),(\pi_{k_2},\pi_{k_2+1}))$ function represents 2-edge exchange move that consists of the second and the third steps of Algorithm 8.

In all these algorithms, the function `Evaluate`(π) calculates the objective function value of the sequence π.

5. Computational experiments

The performance of the GVNS algorithm is compared with that of other heuristic algorithms given in the literature, which were tested on the same benchmark data. The benchmark data include 120 instances, each with 60 jobs and are available at http://www.ozone.ri.cmu.edu/benchmarks.html [11]. The instances were generated to cover a wide range of constraints imposed by three parameters, the due date tightness factor τ, the due date range factor R and the setup time severity factor η. The parameters take the following values: $\tau=\{0.3,0.6,0.9\}$, $R=\{0.25,0.75\}$, and $\eta=\{0.25,0.75\}$. For each of 12 combinations, which will be referred to as a *class*, 10 problem instances were generated, totaling up to 120 instances. The details of these parameters were explained in Section 4.2.

In the rest of this section, we first describe the algorithms given in the literature and then explain how we designed the details of the GVNS algorithm, particularly with respect to neighborhood structures. The GVNS algorithm was coded in C++ and run on an Intel Pentium Xeon 3.3 GHz CPU with 32 GB memory machine with Linux operating system.

5.1. Best known results

The following four sets are the best known results so far for the $1|s_{ij}|\sum w_j T_j$ problem:

1. OBK: Overall aggregated best known solutions composed of the solutions generated by the SA, GA, and TS algorithms by Lin and Ying [14], the ACO algorithm by Liao and Juan [12] and the GA algorithm by Cicirello [15].
2. ACO_AP: Solutions of ACO reported by Anghinolfi and Paolucci [13].
3. DPSO: Solutions of DPSO presented by Anghinolfi and Paolucci [17].
4. DDE: Results of DDE given by Tasgetiren et al. [18].

When we analyze the results of the above studies, we observe that even though ACO_AP and DPSO results improved almost all the OBK solutions, DDE results are superior to the others. We note that in these algorithms, the maximum run time and the maximum number of function evaluations were used as a stopping criterion. In this study, instead of a maximum run time, the maximum number of consecutive non-improving moves was used along with 20,000,000 objective function evaluations. Lastly, we observe that the computer environments were different in each of these studies. While ACO_AP and DPSO were run on 2.8 GHz speed computer, DDE used a computer with 3.2 GHz CPU speed. In order to make a fair comparison regarding the efficiency of the algorithms, we converted the CPU times of previous studies into our computer environment by using the CPU comparison results presented in [25]. The analysis of the CPU times are presented in Section 5.3.

5.2. Neighborhood structures

As described in Section 4.3, the order of the neighborhood structures used in the search process and the search strategy employed in changing neighborhoods will affect the efficiency of the GVNS algorithm. In order to establish the best order of the four neighborhood structures identified in Section 4.3, we performed a preliminary computational experiment both for the VND component and the GVNS itself. Four different orderings of these four neighborhood structures together with their subsets are listed in Table 1, where the numbers refer to the test case numbers corresponding to the neighborhood list.

We first applied the VND algorithm to all $N=120$ test instances with $M=20$ replicas by using test cases given in Table 1. In testing the performance of VND, we compared its results with the best known results in the literature by using the following equation:

$$\Delta_{Case-k}=\sum_{i=1}^{N}\sum_{j=1}^{M}\left(\frac{(VND_{ij}-BK_i)\times 100}{BK_i}\right)\Big/(N\times M)$$

where Δ_{Case-k} represents the average percentage difference for the test case k by using the proposed VND algorithm, $k=1,\ldots,27$, VND_{ij} represents the objective function value of the VND for the test instance i and replica j, and BK_i represents the best-known result for the test instance i. The results of the VND tests are given in Table 2.

As seen from Table 2, three test cases gave better results compared to other test cases, which are highlighted in bold. The best result was obtained for the test case 7. This test case consists

Table 1
Neighborhood structure test cases (1–27).

Swap Insertion Edge-Insertion 2-edge exchange ①	Insertion Edge-Insertion Swap 2-edge exchange ②	Edge-Insertion Insertion Swap 2-edge exchange ③	2-edge exchange Swap Edge-Insertion Insertion ④
Swap Insertion Edge-Insertion ⑤	Swap Insertion 2-edge exchange ⑥	Swap Edge-Insertion Insertion ⑦	Swap Edge-Insertion 2-edge exchange ⑧
Insertion Swap Edge-Insertion ⑨	Insertion Swap 2-edge exchange ⑩	Insertion Edge-Insertion Swap ⑪	Insertion Edge-Insertion 2-edge exchange ⑫
Edge-Insertion Swap Insertion ⑬	Edge-Insertion Swap 2-edge exchange ⑭	Edge-Insertion Insertion Swap ⑮	Edge-Insertion Insertion 2-edge exchange ⑯
2-edge exchange Swap Insertion ⑰	2-edge exchange Swap Edge-Insertion ⑱	2-edge exchange Insertion Edge-Insertion ⑲	2-edge exchange Insertion Swap ⑳
2-edge exchange Edge-Insertion Insertion ㉑	Swap Insertion ㉒	Swap Edge-Insertion ㉓	Insertion Swap ㉔
Insertion Edge-Insertion ㉕	Edge-Insertion Swap ㉖	Edge-Insertion Insertion ㉗	

of three neighborhood structures and their order is as follows: swap, edge-insertion, insertion.

When we analyzed the interactions among these three neighborhood structures, we observed that swap neighborhood successfully obtains very good solutions in a very short time. However, the effect of the edge-insertion and the insertion neighborhoods is also apparent as they improve the local minimum obtained within the swap neighborhood at later stages. We illustrate the working of these neighborhood structures in Fig. 1 for the first instance of 120 benchmark instances. In this experiment, we run VND alone and the initial solution was obtained by applying the ATCS algorithm. In the figure, we observe the local search exchanges which also serve as a diversification mechanism in the VND algorithm.

Next, we performed the same test for the shaking step of the GVNS algorithm, while using the test case 7 for the VND step, to determine the neighborhood structures and their order for GVNS. Differently from VND tests, GVNS tests were only realized for the first instance of each class of the benchmark data because GVNS tests were more time consuming then VND tests. To compare the results of the GVNS algorithm, the following equation is used:

$$\Delta_{Case'-k} = \sum_{\substack{i=1 \\ (i \bmod 10) \equiv 1}} \sum_{j=1}^{M} \left(\frac{(GVNS_{ij} - BK_i) \times 100}{BK_i} \right) \Big/ (N \times M)$$

where $\Delta_{Case'-k}$ represents the average percentage difference for the test case k by using the proposed GVNS algorithm, $k = 1, \ldots, 27$, and $GVNS_{ij}$ represents the objective function value of the GVNS for the test instance i and replica j, with $i = \{1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 101, 111\}$. The results of the GVNS tests are given in Table 3.

As seen from Table 3, six test cases gave better results compared to other test cases, which are pointed out in bold. The best result was obtained for the test case 25. This test case consists of two neighborhood structures and their order is as follows: insertion and edge-insertion.

From these preliminary experiments, we observe that insertion is successful in obtaining good solutions in a very short time and the diversification needed is provided by edge-insertion. We also note that when we have a good perturbation at the shaking step in the algorithm, swap move is not needed. We illustrate the working of these neighborhood structures within the GVNS algorithm in Fig. 2 for the first instance of 120 benchmark instances. In the figure, 100 neighborhood exchanges are plotted. Solid line represents insertion neighborhood structure, dashed line represents edge insertion neighborhood structure and dotted line represents the best solution obtained. Also, the updates of the best solution obtained are shown.

The above preliminary tests also indicate that 2-edge exchange is not an efficient neighborhood structure for the $1|s_{ij}|\sum w_j T_j$ problem. The reason for this behavior can be attributed to the problem structure. We infer that 2-edge exchange's destructive nature may not be applicable to the $1|s_{ij}|\sum w_j T_j$ problem.

Finally, we have to decide about the search strategy employed in changing neighborhoods. In the literature, two common strategies are sequential and nested search [23]. Since we obtained very satisfactory results with the sequential search and the nested strategy searches a much larger neighborhood and hence requires more computational time, the sequential search was preferred in the GVNS implementation.

Table 2
Neighborhood structure testing for the VND.

Case-k	Δ_{Case-k} (%)	Time (s)
1	45.47	0.04
2	48.54	0.06
3	46.95	0.07
4	**41.90**	**0.05**
5	46.15	0.04
6	67.73	0.03
7	**41.45**	**0.04**
8	68.31	0.03
9	46.42	0.06
10	76.67	0.05
11	47.34	0.06
12	56.02	0.06
13	**43.11**	**0.07**
14	84.48	0.06
15	49.08	0.07
16	53.35	0.07
17	67.02	0.03
18	72.24	0.04
19	53.12	0.05
20	66.65	0.04
21	52.06	0.06
22	64.87	0.03
23	74.29	0.03
24	70.32	0.05
25	58.28	0.06
26	85.26	0.06
27	51.29	0.07

5.3. Results for the $1|s_{ij}|\sum w_j T_j$ problem

In this subsection, we present the results of our computational experiments to demonstrate the performance of the proposed GVNS algorithm compared to the former studies which were explained in Section 5.1. In the computational experiments, 120 test instances were solved with the GVNS algorithm. For each test instance we run GVNS algorithm 20 times and the best of these

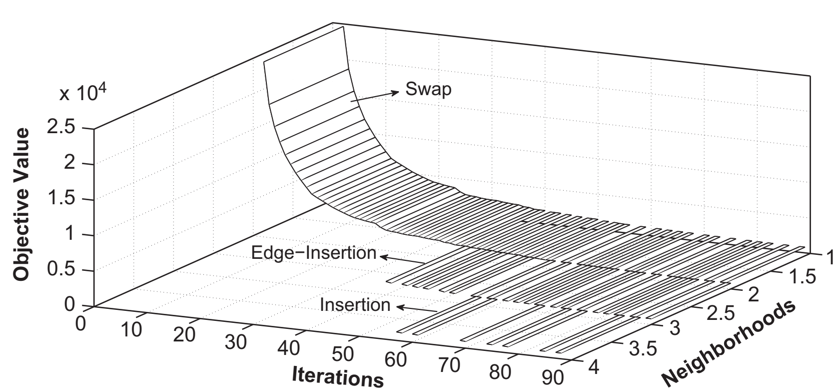

Fig. 1. Change of neighborhood structures within the VND algorithm.

20 replicas is reported (we note that in previous studies, the best result out of 10 replicas was reported). The results of the comparison are presented in Table 4. In this table, we give the values of OBK, ACO_AP, DPSO, DDE, and GVNS in columns 2–6 for each instance of the benchmark set. In columns 7–10, we then report the percentage differences of these values from GVNS. The results displayed in bold fonts indicate that the corresponding solution is the best known solution and other methods could not obtain an equal solution (that is, if $z_{ij} < z_{kj}; \forall i,k,j, i \neq k \Longrightarrow z_{ij}$ is in bold where i, k are the indices to represent different algorithms, j is the index referring to an instance and z_{ij} is the objective function value of the ith algorithm for the jth instance). Equally solved best known solutions are further shown in italic (that is, if $z_{ij} \leq z_{kj}; \forall i,k,j, i \neq k \Longrightarrow z_{ij}$ is in italic).

From Table 4, we can see that the results for 37 (30.83%) out of 120 instances are improved (represented in bold), and for 65 (54.17%) out of 120 instances are equally solved (represented in italic). The GVNS algorithm obtains inferior result only in 18 (15.00%) out of 120 instances compared to all algorithms presented in the literature (that is over all OBK, ACO_AP, DPSO and DDE results). In order to justify the above conclusions, we carried out a paired-sample t-test for pairwise comparisons based on the results given in Table 4. Each pairwise comparison of the GVNS approach was significant at 95% confidence level. This indicates that the differences between solutions obtained by the GVNS algorithm and the competing algorithms are statistically significant. A summary of these observations is given in Table 5. The table exhibits the number of improved, equally solved and unimproved solutions of the GVNS algorithm with respect to each of the algorithms from the literature. In the last column, the table displays the number of improved, equally solved and unimproved solutions of the GVNS algorithm with respect to the best known solution from the literature prior to our study.

Table 3
Neighborhood structure testing for the GVNS.

Case-k	$\Delta_{Case'-k}$ (%)	Time (s)
1	5.82	8.36
2	5.83	8.21
3	5.87	8.50
4	5.88	8.78
5	**5.04**	**6.66**
6	5.92	7.94
7	5.19	6.57
8	5.99	9.04
9	5.21	6.33
10	5.88	8.07
11	**4.98**	**6.53**
12	5.66	9.93
13	5.14	6.78
14	6.33	8.49
15	**4.97**	**6.39**
16	5.44	10.38
17	6.23	8.32
18	6.10	8.65
19	5.41	10.19
20	6.28	9.02
21	5.49	10.75
22	**5.08**	**5.59**
23	5.75	5.82
24	5.13	5.52
25	**4.70**	**8.39**
26	5.71	5.85
27	**4.76**	**8.49**

We further analyzed the effects of major parameters that characterize the problem instances. We summarize our results in Table 6. We note that τ gives an information about the tightness of the due dates; if τ value is large, then the due dates are tighter compared to the case that τ values are small. R provides an empirical measurement about the spread of the due dates; a small value of R indicates a clustered set of due dates. η describes the relative importance of the setup times compared to the processing times; a larger value for η designates more significant setup value with respect to the processing time.

Combining the results given in Tables 5 and 6, we observe the following points:

1. GVNS is better in terms of improving the existing result when η value is high given that τ and R values are constant. This shows the success of GVNS in solving problems having significant setup times with respect to the processing times. In other words, if the importance of the setup times increases, GVNS is more powerful compared to the existing algorithms.
2. GVNS improves more of the existing solutions for the instances when R value is small. This demonstrates the success of GVNS for those instances where the due dates are not scattered.
3. The highest improvement obtained by GVNS is for the second class of instances where $\tau = 0.3$, R=0.25 and $\eta = 0.75$. This corresponds to the case where the due dates are tighter but distributed in a smaller area and the setup times are significant with respect to the processing times. While the improvement over the most successful algorithm in the literature (DDE) is 13.23%, the number of improved solutions is 8 out of 10 instances in this class. The remaining two instances are solved equally well in comparison with the existing algorithms.

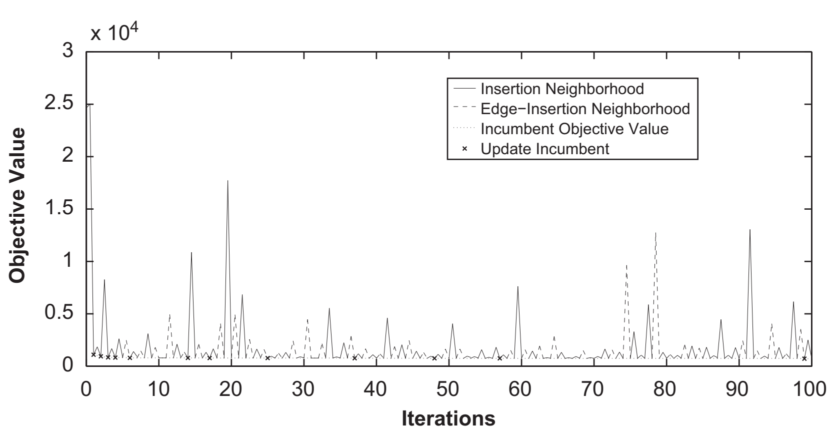

Fig. 2. Neighborhood structure characteristics for the GVNS.

Table 4
The comparison of the GVNS algorithm with the best known results from the literature for $1|s_{ij}|\sum w_jT_j$ problem.

Ins.	OBK	ACO_AP	DPSO	DDE	GVNS	ΔOBK	ΔACO_AP	ΔDPSO	ΔDDE
1	684	513	531	474	**471**	−31.14	−8.19	−11.30	−0.63
2	5082	5082	5088	4902	**4878**	−4.01	−4.01	−4.13	−0.49
3	1792	1769	1609	1465	**1430**	−20.20	−19.16	−11.12	−2.39
4	6526	6286	6146	**5946**	6006	−7.97	−4.45	−2.28	1.01
5	4662	4263	4339	**4084**	4114	−11.75	−3.50	−5.19	0.73
6	**5788**	7027	6832	6652	6667	15.19	−5.12	−2.42	0.23
7	3693	3598	3514	3350	**3330**	−9.83	−7.45	−5.24	−0.60
8	142	129	132	114	**108**	−23.94	−16.28	−18.18	−5.26
9	6349	6094	6153	5803	**5751**	−9.42	−5.63	−6.53	−0.90
10	2021	1931	1895	1799	**1789**	−11.48	−7.35	−5.59	−0.56
11	3867	3853	3649	3294	**2998**	−22.47	−22.19	−17.84	−8.99
12	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
13	5685	4597	4430	4194	**4068**	−28.44	−11.51	−8.17	−3.00
14	3045	2901	2749	2268	**2260**	−25.78	−22.10	−17.79	−0.35
15	1458	1245	1250	964	**935**	−35.87	−24.90	−25.20	−3.01
16	4940	4482	4127	3876	**3381**	−31.56	−24.56	−18.08	−12.77
17	204	128	75	61	**0**	−100.00	−100.00	−100.00	−100.00
18	1610	1237	971	857	**845**	−47.52	−31.69	−12.98	−1.40
19	208	*0*	*0*	*0*	*0*	−100.00	0.00	0.00	0.00
20	2967	2545	2675	2111	**2053**	−30.81	−19.33	−23.25	−2.75
21	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
22	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
23	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
24	1063	1047	1043	1033	**920**	−13.45	−12.13	−11.79	−10.94
25	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
26	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
27	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
28	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
29	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
30	165	130	*0*	*0*	*0*	−100.00	−100.00	0.00	0.00
31	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
32	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
33	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
34	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
35	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
36	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
37	755	400	186	107	**46**	−93.91	−88.50	−75.27	−57.01
38	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
39	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
40	*0*	*0*	*0*	*0*	*0*	0.00	0.00	0.00	0.00
41	71 186	70 253	**69 102**	69 242	69 242	−2.73	−1.44	0.20	0.00
42	58 199	57 847	**57 487**	57 511	57 511	−1.18	−0.58	0.04	0.00
43	147 211	146 697	145 883	*145 310*	*145 310*	−1.29	−0.95	−0.39	0.00
44	35 648	35 331	35 331	*35 289*	*35 289*	−1.01	−0.12	−0.12	0.00
45	59 307	*58 935*	59 175	*58 935*	59 025	−0.48	0.15	−0.25	0.15
46	35 320	35 317	34 805	*34 764*	*34 764*	−1.57	−1.57	−0.12	0.00
47	73 984	73 787	73 378	73 005	**72 853**	−1.53	−1.27	−0.72	−0.21
48	65 164	65 261	*64 612*	*64 612*	*64 612*	−0.85	−0.99	0.00	0.00
49	79 055	78 424	77 771	**77 641**	77 833	−1.55	−0.75	0.08	0.25
50	32 797	31 826	31 810	31 565	**31 292**	−4.59	−1.68	−1.63	−0.86
51	52 639	50 770	49 907	49 927	**49 761**	−5.47	−1.99	−0.29	−0.33
52	99 200	95 951	94 175	94 603	**93 106**	−6.14	−2.97	−1.14	−1.58
53	91 302	87 317	86 891	*84 841*	*84 841*	−7.08	−2.84	−2.36	0.00
54	123 558	120 782	**118 809**	119 226	119 074	−3.63	−1.41	0.22	−0.13
55	69 776	68 843	68 649	66 006	**65 400**	−6.27	−5.00	−4.73	−0.92

Table 4 (*continued*)

Ins.	OBK	ACO_AP	DPSO	DDE	GVNS	ΔOBK	ΔACO_AP	ΔDPSO	ΔDDE
56	78 960	76 503	75 490	75 367	**74 940**	−5.09	−2.04	−0.73	−0.57
57	67 447	66 534	64 575	**64 552**	64 575	−4.26	−2.94	0.00	0.04
58	48 081	47 038	45 680	*45 322*	*45 322*	−5.74	−3.65	−0.78	0.00
59	55 396	54 037	52 001	52 207	**51 649**	−6.76	−4.42	−0.68	−1.07
60	68 851	62 828	63 342	**60 765**	61 755	−10.31	−1.71	−2.51	1.63
61	76 396	*75 916*	*75 916*	*75 916*	*75 916*	−0.63	0.00	0.00	0.00
62	*44 769*	44 869	*44 769*	*44 769*	*44 769*	0.00	−0.22	0.00	0.00
63	*75 317*	*75 317*	*75 317*	*75 317*	*75 317*	0.00	0.00	0.00	0.00
64	*92 572*	*92 572*	*92 572*	*92 572*	*92 572*	0.00	0.00	0.00	0.00
65	127 912	*126 696*	*126 696*	*126 696*	*126 696*	−0.95	0.00	0.00	0.00
66	59 832	*59 685*	*59 685*	*59 685*	*59 685*	−0.25	0.00	0.00	0.00
67	*29 390*	*29 390*	*29 390*	*29 390*	*29 390*	0.00	0.00	0.00	0.00
68	22 148	*22 120*	*22 120*	*22 120*	*22 120*	−0.13	0.00	0.00	0.00
69	**64 632**	71 118	71 118	71 118	71 118	10.04	0.00	0.00	0.00
70	*75 102*	*75 102*	*75 102*	*75 102*	*75 102*	0.00	0.00	0.00	0.00
71	150 709	145 825	145 771	*145 007*	*145 007*	−3.78	−0.56	−0.52	0.00
72	46 903	45 810	43 994	43 904	**43 286**	−7.71	−5.51	−1.61	−1.41
73	29 408	28 909	*28 785*	*28 785*	*28 785*	−2.12	−0.43	0.00	0.00
74	33 375	32 406	30 734	30 313	**30 136**	−9.70	−7.00	−1.95	−0.58
75	21 863	22 728	*21 602*	*21 602*	*21 602*	−1.19	−4.95	0.00	0.00
76	55 055	55 296	53 899	**53 555**	54 024	−1.87	−2.30	0.23	0.88
77	34 732	32 742	31 937	32 237	**31 817**	−8.39	−2.83	−0.38	−1.30
78	21 493	20 520	19 660	*19 462*	*19 462*	−9.45	−5.16	−1.01	0.00
79	121 118	117 908	*114 999*	*114 999*	*114 999*	−5.05	−2.47	0.00	0.00
80	20 335	18 826	*18 157*	*18 157*	*18 157*	−10.71	−3.55	0.00	0.00
81	384 996	*383 485*	383 703	*383 485*	*383 485*	−0.39	0.00	−0.06	0.00
82	410 979	409 982	409 544	409 544	**409 479**	−0.36	−0.12	−0.02	−0.02
83	460 978	458 879	458 787	*458 752*	*458 752*	−0.48	−0.03	−0.01	0.00
84	330 384	*329 670*	*329 670*	*329 670*	*329 670*	−0.22	0.00	0.00	0.00
85	555 106	*554 766*	555 130	554 993	*554 766*	−0.06	0.00	−0.07	−0.04
86	364 381	361 685	*361 417*	*361 417*	*361 417*	−0.81	−0.07	0.00	0.00
87	399 439	398 670	*398 551*	398 670	*398 551*	−0.22	−0.03	0.00	−0.03
88	434 948	434 410	433 519	**433 186**	433 244	−0.39	−0.27	−0.06	0.01
89	410 966	410 102	*410 092*	*410 092*	*410 092*	−0.21	0.00	0.00	0.00
90	402 233	401 959	*401 653*	*401 653*	*401 653*	−0.14	−0.08	0.00	0.00
91	344 988	340 030	343 029	340 508	**339 933**	−1.47	−0.03	−0.90	−0.17
92	365 129	361 407	*361 152*	*361 152*	*361 152*	−1.09	−0.07	0.00	0.00
93	410 462	408 560	406 728	**404 548**	404 917	−1.35	−0.89	−0.45	0.09
94	335 550	333 047	332 983	333 020	**332 949**	−0.78	−0.03	−0.01	−0.02
95	521 512	517 170	521 208	**517 011**	517 646	−0.74	0.09	−0.68	0.12
96	461 484	461 479	459 321	*457 631*	*457 631*	−0.83	−0.83	−0.37	0.00
97	413 109	411 291	410 889	409 263	**407 590**	−1.34	−0.90	−0.80	−0.41
98	532 519	526 856	522 630	523 486	**520 582**	−2.24	−1.19	−0.39	−0.55
99	370 080	368 415	365 149	364 442	**363 977**	−1.65	−1.20	−0.32	−0.13
100	439 944	436 933	432 714	**431 736**	432 068	−1.79	−1.11	−0.15	0.08
101	353 408	*352 990*	*352 990*	*352 990*	*352 990*	−0.12	0.00	0.00	0.00
102	493 889	493 936	493 069	492 748	**492 572**	−0.27	−0.28	−0.10	−0.04
103	379 913	*378 602*	*378 602*	*378 602*	*378 602*	−0.35	0.00	0.00	0.00
104	358 222	358 033	*357 963*	*357 963*	*357 963*	−0.07	−0.02	0.00	0.00
105	450 808	*450 806*	*450 806*	*450 806*	*450 806*	0.00	0.00	0.00	0.00
106	455 849	455 093	455 152	*454 379*	*454 379*	−0.32	−0.16	−0.17	0.00
107	353 371	353 368	352 867	*352 766*	*352 766*	−0.17	−0.17	−0.03	0.00
108	462 737	461 452	*460 793*	*460 793*	*460 793*	−0.42	−0.14	0.00	0.00
109	413 205	413 408	*413 004*	*413 004*	*413 004*	−0.05	−0.10	0.00	0.00
110	419 481	418 769	*418 769*	*418 769*	*418 769*	−0.17	0.00	0.00	0.00
111	347 233	346 763	*342 752*	*342 752*	*342 752*	−1.29	−1.16	0.00	0.00

112	373 238	373 140	369 237	*367 110*	*367 110*	−1.64	−1.62	−0.58	0.00
113	261 239	260 400	260 176	260 872	**259 649**	−0.61	−0.29	−0.20	−0.47
114	470 327	464 734	464 136	465 503	**463 474**	−1.46	−0.27	−0.14	−0.44
115	459 194	457 782	457 874	457 289	**457 189**	−0.44	−0.13	−0.15	−0.02
116	**527 459**	532 840	532 456	530 803	530 601	0.60	−0.42	−0.35	−0.04
117	512 286	506 724	503 199	**502 840**	503 046	−1.80	−0.73	−0.03	0.04
118	352 118	355 922	350 729	*349 749*	*349 749*	−0.67	−1.73	−0.28	0.00
119	579 462	573 910	*573 046*	*573 046*	*573 046*	−1.11	−0.15	0.00	0.00
120	398 590	397 520	*396 183*	*396 183*	*396 183*	−0.60	−0.34	0.00	0.00
Avg.						−7.51	−5.21	−3.41	−1.81

Table 5
Comparing the GVNS algorithm with the existing algorithms for $1|s_{ij}|\sum w_jT_j$ problem.

	OBK	ACO_AP	DPSO	DDE	BEST
Number of improved solutions	94	83	63	41	37
Number of equal solutions	23	35	52	66	65
Number of inferior solutions	3	2	5	13	18
Number of all solutions	120	120	120	120	120

Table 6
Average improvement of the GVNS algorithm with respect to the existing algorithms in each class of instances of $1|s_{ij}|\sum w_jT_j$ problem.

Instance	ΔOBK	ΔACO_AP	ΔDPSO	ΔDDE
$\tau=0.3, R=0.25, \eta=0.25$(001−010)	−11.46	−8.11	−7.20	−0.89
$\tau=0.3, R=0.25, \eta=0.75$(011−020)	−42.24	−25.63	−22.33	−13.23
$\tau=0.3, R=0.75, \eta=0.25$(021−030)	−11.35	−11.21	−1.18	−1.09
$\tau=0.3, R=0.75, \eta=0.75$(031−040)	−9.39	−8.85	−7.53	−5.70
$\tau=0.6, R=0.25, \eta=0.25$(041−050)	−1.68	−0.92	−0.29	−0.07
$\tau=0.6, R=0.25, \eta=0.75$(051−060)	−6.07	−2.90	−1.30	−0.29
$\tau=0.6, R=0.75, \eta=0.25$(061−070)	0.81	−0.02	0.00	0.00
$\tau=0.6, R=0.75, \eta=0.75$(071−080)	−6.00	−3.48	−0.52	−0.24
$\tau=0.9, R=0.25, \eta=0.25$(081−090)	−0.33	−0.06	−0.02	−0.01
$\tau=0.9, R=0.25, \eta=0.75$(091−100)	−1.33	−0.62	−0.41	−0.10
$\tau=0.9, R=0.75, \eta=0.25$(101−110)	−0.19	−0.09	−0.03	0.00
$\tau=0.9, R=0.75, \eta=0.75$(111−120)	−0.90	−0.68	−0.17	−0.09
Average	−7.51	−5.21	−3.41	−1.81

In order to show the robustness of the GVNS algorithm with respect to the solution quality, we further analyzed the deviation to be observed within 20 replicas we performed. For this purpose, the minimum, the average, the maximum and the standard deviation of the relative percentage deviations from the reference objective function values ($REF_i, i=1,\ldots,N$) were calculated, where REF corresponds to OBK, ACO_AP or DDE. Percentage deviation of the GVNS algorithm's ith test problem's jth replica (l_{ij}) is calculated as follows:

$$l_{ij}=\frac{GVNS_{ij}-REF_i}{REF_i} \tag{11}$$

The minimum, the average, the maximum and the standard deviation of the relative percentage deviations are further calculated as in equations from (12) to (15), respectively:

$$\Delta_{min}=\sum_{i=1}^{N}\min_{j=1,\ldots,M}(l_{ij})\Big/N \tag{12}$$

$$\Delta_{max}=\sum_{i=1}^{N}\max_{j=1,\ldots,M}(l_{ij})\Big/N \tag{13}$$

$$\Delta_{avg}=\sum_{i=1}^{N}\left(\sum_{j=1}^{M}l_{ij}\Big/M\right)\Big/N \tag{14}$$

$$\Delta_{std}=\sum_{i=1}^{N}(\sigma_{j=1,\ldots,M}(l_{ij}))\Big/N \tag{15}$$

To provide additional information regarding the robustness of the GVNS algorithm, we calculated the percentage of the number of improved solutions (Im%) and the percentage of the number of instances which are equally solved (Eq%) among 2400 runs (20 replicas for each of 120 instances) when compared to the reference best known solutions. Im% and Eq% are defined in (16) and (17), respectively. All of these statistics are given in Table 7:

$$\mathrm{Im}\% = \sum_{i=1}^{N} \left(\sum_{j=1}^{M} ((l_{ij} > 0) \Longrightarrow 1) \Big/ M \right) \Big/ N \tag{16}$$

$$\mathrm{Eq}\% = \sum_{i=1}^{N} \left(\sum_{j=1}^{M} ((l_{ij} = 0) \Longrightarrow 1) \Big/ M \right) \Big/ N \tag{17}$$

Table 7
Comparing the GVNS algorithm with the existing algorithms with respect to different dimensions for $1|s_{ij}|\sum w_j T_j$ problem.

Algorithm	Δ_{min}	Δ_{avg}	Δ_{max}	Δ_{std}	Im%	Eq%
OBK	−7.51	−5.61	−3.22	1.35	71.21	17.33
ACO$_{AP}$	−5.21	−2.83	0.04	1.63	48.62	26.08
DPSO	−3.41	−0.07	4.28	2.31	26.42	30.83
DDE	−1.81	2.64	7.95	2.91	10.38	33.12

It can be observed from Table 7 that GVNS outperforms all previous algorithms if it is compared on the basis of the best solution found within 20 replicas. When the average solution found by GVNS over 20 replicas is compared with the best solution of previous studies, we see that GVNS still outperforms all but DDE. When the worst solution of GVNS is compared with

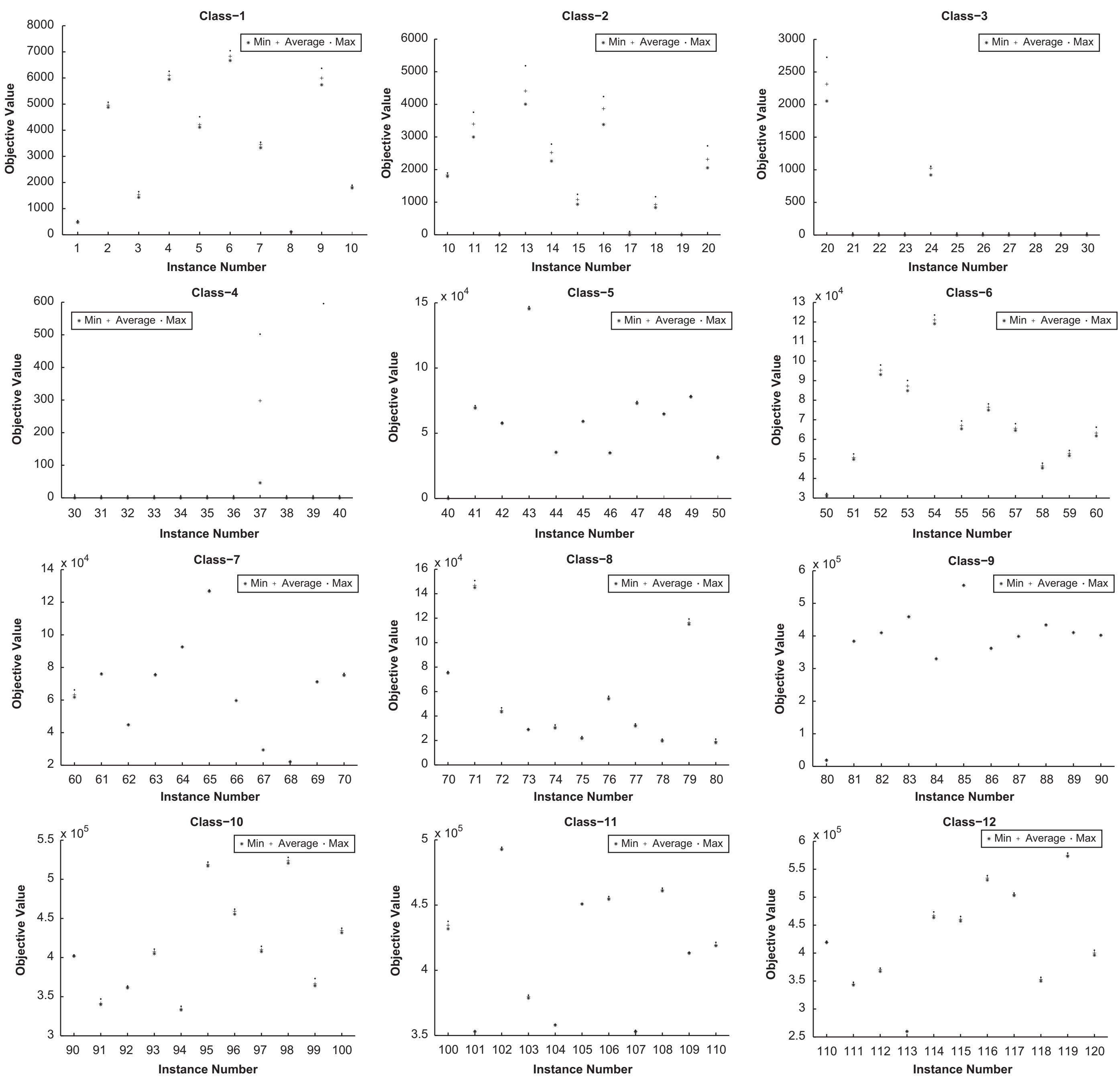

Fig. 3. GVNS algorithm's minimum, average and maximum results among the 20 replicas for each class of instances.

the best solution of other algorithms, as expected, the success of GVNS deteriorates. However, we notice that the average standard deviation from the former studies is less than 3% which shows that the solution quality of GVNS does not deviate too much within 20 replicas.

Next, we plotted the minimum, the average and the maximum solution values among 20 replicas in Fig. 3 to provide a detailed analysis of these results with respect to each instance. These plots clearly indicate that the GVNS algorithm is able to produce similar solutions within 20 replicas except a few instances.

Overall, we can say that the results presented in Table 7 and Fig. 3 demonstrate the robustness of the GVNS algorithm, given its stochastic nature, regarding the solution quality it produces.

Finally, we analyze the efficiency of the GVNS algorithm. It was reported that the CPU time of ACO_AP, DPSO and DDE were 65 s, 22.5 s and 9 s on the average, respectively. The converted CPU times are 38.7 s, 13.4 s and 6.4 s in our computer environment. Since the CPU time required by the proposed GVNS is 8.9 *s* on the average, we can state that the GVNS algorithm is an efficient algorithm on the average. We note that even though DDE seems to have a better efficiency on the average, it has at most 25 s to find its best solution [18]. Furthermore, DDE uses speedup methods, which gives an advantage to this algorithm with respect to the solution time. The converted maximum CPU time for DDE is 17.8 s. During our experiments, we observed that the maximum CPU time of the GVNS algorithm without speedup methods is just 12.3 s. This further confirms that the GVNS algorithm is a robust algorithm with respect to the efficiency.

5.4. Results for the $1|s_{ij}|\sum T_j$ problem

Given the superior performance of the GVNS for the $1|s_{ij}|\sum w_jT_j$ problem, we tested its robustness by applying it to another NP-hard problem, that is, the single machine total tardiness with sequence dependent setup times (STTSDS) problem. STTSDS is a special case of $1|s_{ij}|\sum w_jT_j$ problem and represented as $1|s_{ij}|\sum T_j$ with the three-field notation. Du and Leung [26] show that an objective to minimize total tardiness is NP-hard even for the single machine $(1||\sum T_j)$. Since $1|s_{ij}|\sum T_j$ incorporates sequence dependent setup times, problem is still NP-hard.

In the literature, the $1|s_{ij}|\sum T_j$ problem has been solved with different metaheuristic approaches due to its time complexity. Among these, the genetic search algorithm and the random search pairwise interchange method (RPSI) developed by Rubin and Ragatz [27], the ant colony optimization algorithm (ACO_GPG) proposed by Gagné et al. [28], a hybridization of the tabu search and the variable neighborhood search (TS-VNS) presented by Gagné et al. [29] and the ant colony optimization algorithm (ACO_LJ) proposed by Liao and Juan [12] can be mentioned. Apart from these metaheuristic approaches, Bigras et al. developed a branch-and-bound (B&B) algorithm [30]. In the most recent study, Ying et al. proposed an iterated greedy search algorithm (IG) for the $1|s_{ij}|\sum T_j$ problem [31]. Their study resulted in improving the best known results for most of the instances for the $1|s_{ij}|\sum T_j$ problem.

In our study, we compared the proposed GVNS algorithm with the metaheuristic approaches mentioned above and with the B&B algorithm by using two sets of benchmark data from the literature. The first set includes 32 small and medium sized instances which were generated by Rubin and Ragatz [27]. These instances were classified into four groups, each with 15, 25, 35, and 45 jobs, respectively. In each group, there are eight instances, which were derived by means of three-factor experimental design. Factors are processing time variance, the tardiness factor and due dates range. The second set was generated by Gagné et al. by using the same procedure [28]. This set includes large sized instances with 55, 65, 75 and 85 jobs.

Table 8
The best known results for small and medium sized instances of the $1|s_{ij}|\sum T_j$ problem.

Ins.	Jobs	RSPI	ACO_GPG	Tabu-VNS	ACO_LJ	B&B	IG	GVNS
Prob401	15	90*	90*	90*	90*	90*	90*	90*
Prob402	15	0*	0*	0*	0*	0*	0*	0*
Prob403	15	3418*	3418*	3418*	3418*	3418*	3418*	3418*
Prob404	15	1067*	1067*	1067*	1067*	1067*	1067*	1067*
Prob405	15	0*	0*	0*	0*	0*	0*	0*
Prob406	15	0*	0*	0*	0*	0*	0*	0*
Prob407	15	1861*	1861*	1861*	1861*	1861*	1861*	1861*
Prob408	15	5660*	5660*	5660*	5660*	5660*	5660*	5660*
Prob501	25	266	261*	261*	263	261*	261*	261*
Prob502	25	0*	0*	0*	0*	0*	0*	0*
Prob503	25	3497*	3497*	3503	3497*	3497*	3497*	3497*
Prob504	25	0*	0*	0*	0*	0*	0*	0*
Prob505	25	0*	0*	0*	0*	0*	0*	0*
Prob506	25	0*	0*	0*	0*	0*	0*	0*
Prob507	25	7225*	7268	7225*	7225*	7225*	7225*	7225*
Prob508	25	1915*	1945	1915*	1915*	1915*	1915*	1915*
Prob601	35	36	16	12*	14	12*	12*	12*
Prob602	35	0*	0*	0*	0*	0*	0*	0*
Prob603	35	17 792	17 685	17 605	17 654	17 587*	17 587*	17 587*
Prob604	35	19 238	19 213	19 168	*19 092*	*19 092*	*19 092*	*19 092*
Prob605	35	273	247	228*	240	228*	228*	228*
Prob606	35	0*	0*	0*	0*	0*	0*	0*
Prob607	35	13 048	13 088	12 969*	13 010	12 969*	12 969*	12 969*
Prob608	35	4733	4733	4732*	4732*	4732*	4732*	4732*
Prob701	45	118	103	98	103	97*	103	99
Prob702	45	0*	0*	0*	0*	0*	0*	0*
Prob703	45	26 745	26 663	26 506	26 568	26 533	**26 496**	26 506
Prob704	45	15 415	15 495	15 213	15 409	16 577	*15 206*	*15 206*
Prob705	45	254	222	200*	219	200*	200*	202
Prob706	45	0*	0*	0*	0*	0*	0*	0*
Prob707	45	24 218	24 017	23 804	23 931	23 797	23 794	**23 789**
Prob708	45	23 158	23 351	22 873	23 028	22 829	22 829	**22 807**

We applied the GVNS algorithm to the $1|s_{ij}|\sum T_j$ problem without changing any of the implementation issues, except that $w_j = 1 \forall j$. Initial solution for the problem is also obtained by using the ATCS heuristic. The solutions of the previous studies are taken from [31]. We present the results of the GVNS algorithm for the small and medium sized instances in Table 8 together with the results of previous approaches. In the table, solutions with "$*$" represent that the corresponding solution is optimal, whereas bold and italic font results have the same meaning as given in Section 5.3.

From Table 8, we observe that the GVNS algorithm was able to find the optimal solution for 25 out of 32 instances. Among the remaining instances, the GVNS algorithm improved the best known results for two instances, and solved another two instances equally. The results obtained by the GRASP algorithm are not presented in this table because these results are given relative to the branch-and-bound results [32]. Hence we further compare the GVNS algorithm with the GRASP algorithm and present this comparison in Table 10.

After the small and medium sized instances, large sized instances were solved and the results are given in Table 9. We note that the GVNS algorithm improves the results of 14 instances out of 32 and equally solves another 14 instances.We note that the solution of the "Prob751" is not given in [31].

Summary results for all instances are presented in Table 10. In Table 10, the solutions of small and medium sized, and large sized instances are given separately. The entries given by '-' for the RSPI algorithm denote that these instances were not solved by that algorithm. Table 10 indicates that the results obtained by the GVNS algorithm are highly competitive compared to other algorithms.

6. Conclusions

This paper presents a general variable neighborhood search algorithm applied to the single machine total weighted tardiness problem with sequence dependent setup times. To obtain the initial solution for the problem, the well-known ATCS heuristic was used. The general variable neighborhood search algorithm was tested on a set of benchmark instances from the literature and compared to the best performing algorithms. Out of 120 instances, 37 overall aggregated best known solutions were improved, and 65 instances were solved equally by the general variable neighborhood search algorithm. Moreover, the proposed algorithm does not use any speedup methods and yet solves the test instances much faster compared to the existing algorithms.

Table 9
The best known results for large sized problem instances of the $1|s_{ij}|\sum T_j$ problem.

Ins.	Jobs	ACO_GPG	Tabu-VNS	ACO_LJ	GRASP	IG	GVNS
Prob551	55	212	185	*183*	242	*183*	194
Prob552	55	*0*	*0*	*0*	*0*	*0*	*0*
Prob553	55	40 828	40 644	40 676	40 678	40 598	**40 540**
Prob554	55	15 091	14 711	14 684	*14 653*	*14 653*	*14 653*
Prob555	55	*0*	*0*	*0*	*0*	*0*	*0*
Prob556	55	*0*	*0*	*0*	*0*	*0*	*0*
Prob557	55	36 489	35 841	36 420	35 883	**35 827**	35 830
Prob558	55	20 624	19 872	19 888	*19 871*	*19 871*	*19 871*
Prob651	65	295	268	268	333	268	**264**
Prob652	65	*0*	*0*	*0*	*0*	*0*	*0*
Prob653	65	57 779	57 602	57 584	57 880	57 584	**57 515**
Prob654	65	34 468	34 466	34 306	34 410	34 306	**34 301**
Prob655	65	13	*2*	7	30	*2*	4
Prob656	65	*0*	*0*	*0*	*0*	*0*	*0*
Prob657	65	56 246	55 080	55 389	55 355	55 080	**54 895**
Prob658	65	29 308	27 187	27 208	*27 114*	*27 114*	*27 114*
Prob751	75	263	*241*	*241*	317	–	*241*
Prob752	75	*0*	*0*	*0*	*0*	*0*	*0*
Prob753	75	78 211	77 739	77 663	78 211	77 663	**77 627**
Prob754	75	35 826	35 709	35 630	35 323	35 250	**35 219**
Prob755	75	*0*	*0*	*0*	*0*	*0*	*0*
Prob756	75	*0*	*0*	*0*	*0*	*0*	*0*
Prob757	75	61 513	59 763	60 108	60 217	59 763	**59 716**
Prob758	75	40 277	38 789	38 704	38 368	38 341	**38 339**
Prob851	85	453	**384**	455	531	390	402
Prob852	85	*0*	*0*	*0*	*0*	*0*	*0*
Prob853	85	98 540	97 880	98 443	98 794	97 880	**97 595**
Prob854	85	80 693	80 122	79 553	80 338	79 631	**79 271**
Prob855	85	333	283	324	393	283	**280**
Prob856	85	*0*	*0*	*0*	*0*	*0*	*0*
Prob857	85	89 654	87 244	87 504	88 089	87 244	**87 075**
Prob858	85	77 919	75 533	75 506	75 217	75 029	**74 755**

Table 10
Comparing the GVNS algorithm with previous methods for $1|s_{ij}|\sum T_j$ problem.

	RSPI	ACO_GPG	Tabu-VNS	ACO_LJ	GRASP	IG
Number of improved solutions	13/-(13)	14/22 (36)	11/17 (28)	4/19 (23)	10/19 (29)	3/13 (16)
Number of equal solutions	19/-(19)	18/10 (28)	21/11 (32)	26/11 (37)	17/13 (30)	27/13 (40)
Number of unimproved solutions	0/-(0)	0/0 (0)	0/4 (4)	2/2 (4)	5/0 (5)	2/5 (7)
Number of all solutions	32/-(32)	32/32 (64)	32/32 (64)	32/32 (64)	32/32 (64)	32/31 (63)

The superior results obtained from the GVNS algorithm for the $1|s_{ij}|\sum w_jT_j$ problem encouraged us to apply the algorithm to the $1|s_{ij}|\sum T_j$ problem. The GVNS algorithm was applied to the single machine total tardiness with sequence dependent setup times problem without changing any parameters of the algorithm. In total 64 different test instances were solved varying from small to large sizes. The GVNS algorithm succeeded in obtaining all the known optimal solutions except in two instances. Moreover, best-known solutions for 16 instances were improved by the GVNS algorithm.

As a consequence of the results obtained, we can conclude that the GVNS algorithm is a very effective and efficient solution procedure for the $1|s_{ij}|\sum w_jT_j$ and the $1|s_{ij}|\sum T_j$ problems. Moreover, it is a robust algorithm.

Appendix A

In this section, we present the algorithms used in the GVNS.

Algorithm 6. SwapMove $N_1(x)$.

1 Choose two random jobs π_{k_1} and π_{k_2} in π, randomly.
2 Swap π_{k_1} and π_{k_2}

Algorithm 7. InsertionMove $N_2(x)$.

1 Select a random job π_{k_1} and position k_2, randomly.
2 Remove π_{k_1} in π
3 Insert π_{k_1} to the position k_2

Algorithm 8. Edge-InsertionMove $N_3(x)$.

1 Choose one consecutive job couple π_{k_1},π_{k_1+1} and position k_2, randomly.
2 Remove (π_{k_1},π_{k_1+1}) in π
3 Insert (π_{k_1},π_{k_1+1}) to the position k_2 in π

Algorithm 9. 2-edgeExchangeMove $N_4(x)$.

1 Choose two consecutive job couples π_{k_1},π_{k_1+1} and π_{k_2},π_{k_2+1} where $|k_2-k_1|\geq 3$, randomly.
2 Remove (π_{k_1},π_{k_1+1}) and (π_{k_2},π_{k_2+1}) in the sequence
3 Reconnect the sequence π

Algorithm 10. BestSwapMove $N'_1(x)$.

input: π
Output: π^b
$\pi^b \leftarrow \pi$;
best $\leftarrow$ Evaluate(π);
foreach $\pi_{k_1} \in \pi$ **do**
  foreach $\pi_{k_2} \in \pi$ *and* $\pi_{k_1} \neq \pi_{k_2}$ **do**
    $\pi \leftarrow$ Swap(π_{k_1},π_{k_2});
    temp $\leftarrow$ Evaluate(π);
    if temp $<$ best **then**
      $\pi^b \leftarrow \pi$;
      best $\leftarrow$ temp;
    $\pi \leftarrow$ Swap(π_{k_1},π_{k_2});

Algorithm 11. BestInsertionMove $N'_2(x)$.

input: π
Output: π^b
$\pi^b \leftarrow \pi$;
best $\leftarrow$ Evaluate(π);
foreach $\pi_{k_1} \in \pi$ **do**
  for $k_2 \leftarrow 1$ **to** n **do**
    $\pi \leftarrow$ Insert(π_{k_1},k_2);
    temp $\leftarrow$ Evaluate(π);
    if temp $<$ best **then**
      $\pi^b \leftarrow \pi$;
      best $\leftarrow$ temp;
    $\pi \leftarrow$ Insert(π_{k_2},k_1);

Algorithm 12. 2-OptMove $N'_3(x)$.

input: π
Output: π^b
$\pi^b \leftarrow \pi$;
best $\leftarrow$ Evaluate(π);
foreach $(\pi_{k_1},\pi_{k_1+1}) \in \pi$ **do**
  foreach $(\pi_{k_1},\pi_{k_1+1}) \in \pi$ *and* $|k_2-k_1| \geq 3$ **do**
    $\pi \leftarrow$ EdgeExchange($(\pi_{k_1},\pi_{k_1+1}),(\pi_{k_2},\pi_{k_2+1})$);
    temp $\leftarrow$ Evaluate(π);
    if temp $<$ best **then**
      $\pi^b \leftarrow \pi$;
      best $\leftarrow$ temp;
    $\pi \leftarrow$ EdgeExchange($(\pi_{k_1},\pi_{k_2+1}),(\pi_{k_1+1},\pi_{k_2})$);

Algorithm 13. BestEdge-InsertionMove $N'_4(x)$.

input: π
Output: π^b
$\pi^b \leftarrow \pi$;
best $\leftarrow$ Evaluate(π);
foreach $(\pi_{k_1},\pi_{k_1+1}) \in \pi$ **do**
  for $k_2 \leftarrow 1$ **to** n **do**
    $\pi \leftarrow$ EdgeInsertion($(\pi_{k_1},\pi_{k_1+1}),k_2$);
    temp $\leftarrow$ Evaluate(π);
    if temp $<$ best **then**
      $\pi^b \leftarrow \pi$;
      best $\leftarrow$ temp;
    $\pi \leftarrow$ EdgeInsertion($(\pi_{k_2},\pi_{k_2+1}),k_1$);

References

[1] Allahverdi A, Ng CT, Cheng TCE, Kovalyov MY. A survey of scheduling problems with setup times or costs. European Journal of Operational Research 2008;187:985–1032.
[2] Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 1979;4:287–326.
[3] Lawler EL. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics 1977;1:331–42.
[4] Lenstra JK, Rinnoy Kan AHG, Brucker P. Complexity of machine scheduling problems. Annals of Discrete Mathematics 1977;1:343–62.
[5] Abdul-Razaq TS, Potts CN, Van Wassenhove LN. A survey of algorithms for the single machine total weighted tardiness scheduling problems. Discrete Applied Mathematics 1990;26:235–53.
[6] Potts CN, Van Wassenhove LN. A branch and bound algorithm for the total weighted tardiness problem. Operations Research 1985;26:363–77.
[7] Potts CN, Van Wassenhove LN. Dynamic programming and decomposition approaches for the single machine total tardiness problem. European Journal of Operational Research 1987;32:405–14.
[8] Vepsalainen APJ, Morton TE. Priority rules for job shops with weighted tardiness cost. Management Science 1987;33(8):1035–47.
[9] Lee YH, Bhaskaram K, Pinedo M. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transactions 1997;29:45–52.

[10] Potts CN, Wassenhove LNV. Single machine tardiness sequencing heuristics. IIE Transactions 1991;23(4):346–54.
[11] Cicirello VA, Smith SF. Enhancing stochastic search performance by value-biased randomization of heuristics. Journal of Heuristics 2005;11:5–34.
[12] Liao C-J, Juan H-C. An ant colony optimization for single machine tardiness scheduling with sequence dependent setups. Computers & Operations Research 2007;34:1899–909.
[13] Anghinolfi D, Paolucci M. A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem. International Journal of Operations Research 2008;5(1):44–60.
[14] Lin S-W, Ying K-C. Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics. International Journal of Advanced Manufacturing Technology 2007;34:1183–90.
[15] Cicirello VA. Non-wrapping order crossover: an order preserving crossover operator that respects absolute position. In: Proceedings of the 8th annual conference on genetic and evolutionary computation, Seattle, Washington, USA, 2006, p. 1125–32.
[16] Valente JMS, Alves RAFS. Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups. Computers & Operations Research 2008;35(7):2388–405.
[17] Anghinolfi D, Paolucci M. A new discrete particle swarm optimization approach for the single machine total weighted tardiness scheduling problem with sequence dependent setup times. European Journal of Operational Research 2009;193:73–85.
[18] Tasgetiren MF, Pan Q-K, Liang Y-C. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. Computers & Operations Research 2009;36(6):1900–15.
[19] Mladenović N, Hansen P. Variable neighborhood search. Computers & Operations Research 1997;24:1097–100.
[20] Mladenović N, Hansen P. An introduction to variable neighborhood search. In: MetaHeuristics: advances and trends in local search paradigms for optimization. Boston: Kluwer Academic; 1999. p. 449–67 [Chapter 30].
[21] Mladenović N, Hansen P. Variable neighborhood search: principles and applications. European Journal of Operational Research 2001;130:449–67.
[22] Hansen P, Mladenović N. An introduction to variable neighborhood search, Handbook of MetaHeuristics. Amsterdam: Kluwer; 2003. p. 145–84 [Chapter 6].
[23] Hansen P, Mladenović N, Moreno Pérez J. Developments of variable neighborhood search. Annals of Operations Research 2010;175(1):367–407.
[24] Hansen P, Mladenovic N, Urosevic D. Variable neighborhood search and local branching. Computers & Operations Research 2006;33(10):3034–45.
[25] Corporation SPE, Spec cpu results; 2011, URL 〈http://www.spec.org/benchmarks.html〉.
[26] Du J, Leung JY-T. Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research 1990;15(3):483–95.
[27] Rubin PA, Ragatz GL. Scheduling in a sequence dependent setup environment with genetic search. Computers & Operations Research 1995;22(1):85–99.
[28] Gagné C, Price W, Gravel M. Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. Journal of the Operational Research Society 2002;53(8):895–906.
[29] Gagné C, Gravel M, Price WL. Using metaheuristic compromise programming for the solution of multiple-objective scheduling problems. Journal of the Operational Research Society 2005;56(6):687–98.
[30] Bigras L, Gamache M, Savard G. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. Discrete Optimization 2008;5(4):685–99.
[31] Ying K-C, Lin S-W, Huang C-Y. Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. Expert Systems With Applications 2009;36(3):7087–92, doi:10.1016/j.eswa.2008.08.033.
[32] Tan K-C, Narasimhan R, Rubin PA, Ragatz GL. A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. Omega 2000;28(3):313–26.