IMPROVING THE COMPUTATIONAL EFFICIENCY OF TEACHING-LEARNING BASED OPTIMIZATION FOR JOB SHOP SCHEDULING PROBLEMS BY ELIMINATING UNNECESSARY OBJECTIVE FUNCTION CALLS

Leonardo Ramos Rodrigues
Instituto de Aeronáutica e Espaço - IAE
Praça Marechal Eduardo Gomes, 50, São José dos Campos, SP, Brasil, 12228-904
leonardolrr@iae.cta.br

João Paulo Pordeus Gomes, Saulo Anderson Freitas de Oliveira
Universidade Federal do Ceará
Rua Campus do Pici, sn, Fortaleza, CE, Brazil, 60440-554
jpaulo@lia.ufc.br, sauloafoliveira@lia.ufc.br

Lucas Sousa de Oliveira, Takashi Yoneyama
Instituto Tecnológico de Aeronáutica - ITA
Praça Marechal Eduardo Gomes, 50, São José dos Campos, SP, Brasil, 12228-900
lsoliveira459@gmail.com, takashi@ita.br

ABSTRACT

This paper presents a method to improve the computational efficiency of TLBO in combinatorial optimization problems. Our goal is to eliminate unnecessary calls to the objective function in order to reduce simulation time. A common approach to solve combinatorial problems with TLBO is the use of priority vectors. Each priority vector is mapped into a solution by sorting its elements. However, different vectors can be mapped into the same solution. We insert a new step, called Change Verification, which consists in comparing the solutions generated from both the current and the modified priority vectors. The objective function is called only if the solutions are different from each other. Experiments with benchmark instances of the job shop scheduling problem were carried out to evaluate the proposed method. The proposed method outperformed the original TLBO in all instances, providing an average reduction in simulation time and objective function calls of 11.7% and 12.2%, respectively.

KEYWORDS. Teaching-Learning Based Optimization. Combinatorial Optimization. Metaheuristics.

Paper topic: MH - Metaheuristics. OC - Combinatorial Optimization

1. Introduction

Among all metaheuristic methods available in the literature, the recently proposed Teaching-Learning Based Optimization (TLBO) algorithm is one of the most promising [Rao et al., 2011]. TLBO is a population based algorithm that simulates the teaching-learning process observed in a classroom. It has achieved remarkable performances in different optimization problems (see Sahu et al. [2015], Sapathy et al. [2013] and Rao et al. [2012]) with the advantage of the absence of hyperparameters [Rao and Patel, 2012].

In recent years, several works have proposed variants of the TLBO algorithm to improve different aspects of the original formulation. In Rao and Patel [2012], the authors proposed an elitist TLBO. The elitism preserves the best candidates of each iteration and thus prevents the loss of a possible good solution. Using more than one teacher and adapting the teaching factor are some of the modifications proposed in Rao and Patel [2013] in order to improve both the exploration and the exploitation capacity of TLBO. Chen et al. [2015] incorporated local learning and self-learning methods in the TLBO algorithm, to the same end.

Although TLBO was originally proposed for optimizing mechanical design problems, it has been successfully used in combinatorial optimization problems. In Baykasoğlu et al. [2014], the authors investigated the performance of TLBO in several instances of both the flow shop and the job shop scheduling problems. For the flow shop scheduling problem, the authors compared the performance of TLBO with the Novel Particle Swarm Optimization (NPSO) and the Hybrid Particle Swarm Optimization (HPSO). For the job shop scheduling problem, the authors compared the performance of TLBO with the Beam Search Algorithm (BSA) and the GRASP algorithm, among others. The performance of TLBO was comparable to the best known performances from the literature. The authors concluded that TLBO is an effective algorithm to solve combinatorial optimization problems.

Despite the numerous variants of TLBO, no previous work addressed the problem of its computational efficiency. Since TBLO is a population based algorithm, it evaluates all candidate solutions at each iteration, which can be considerably time-consuming depending on the objective function. In this paper, we propose a variant of the TLBO with reduced computational cost by exploring particularities of combinatorial problems. Numerical experiments with twenty benchmark instances of the job shop scheduling problem are carried out to evaluate the performance of the proposed method. Experiments using the original version of the TLBO algorithm are also carried out to establish a reference baseline. Promising results were obtained.

Scheduling problems are amongst the most challenging combinatorial optimization problems. Therefore, investigating the performance of TLBO in solving these problems can give us a good idea of its capability to solve other combinatorial optimization problems.

The remaining sections of this paper are organized as follows. Section 2 introduces the teaching-learning based optimization algorithm and describes the formulation of the job shop scheduling problem. Section 3 presents the proposed method to improve the computational efficiency of the TLBO algorithm. Section 4 illustrates the application of the proposed method in twenty benchmark instances of the job shop scheduling problem. Concluding remarks are presented in section 5.

2. Theoretical Background

2.1. Teaching-Learning Based Optimization Algorithm

The Teaching-Learning Based Optimization (TLBO) algorithm has been recently proposed as a novel population oriented metaheuristic algorithm based on the teaching-learning process observed in a classroom [Rao et al., 2011]. This algorithm simulates the influence of a teacher on the output of a group of students in a class. The TLBO algorithm is divided into two main parts: the Teacher Phase and the Student Phase, also known as the Learner Phase [Rao and Patel, 2013].

During the Teacher Phase, students learn from the teacher, while in the Learner Phase students learn through the interaction among themselves.

There is a solution X associated with each student, which corresponds to a possible solution for the optimization problem under consideration. Also, there is a result $f(X)$ associated with each solution (or student), which can be obtained by evaluating the solution X using the objective function $f(\cdot)$. In the job shop scheduling problem considered in this paper, a solution X corresponds to a sequence of tasks for each machine and the associated result $f(X)$ corresponds to the makespan, which is the time spent in order to process all jobs.

At the end of each iteration, the stop criteria are checked. Different stop criteria may be adopted. Some of the most commonly adopted stop criteria are the maximum number of iterations, the maximum number of successive iterations without any improvement, the maximum simulation time and the maximum number of function evaluations.

The student with the best solution in the population is considered as the Teacher. A flowchart describing the steps for implementing the TLBO algorithm are presented in Figure 1 [Rao et al., 2011]. The Teacher Phase and the Student Phase are described in the next sections.

2.1.1. Teacher Phase:

In this phase, the algorithm simulates the learning of the students from the teacher (best solution). During this phase, the teacher makes an effort to increase the mean result of the class.

Consider a group of n students. Let M_i be the mean solution of the students and T_i be the teacher at iteration i. The teacher T_i will make an effort to move M_i to its own level. Knowledge gain is based on the quality of the teacher and the quality of students in the class. The difference D_i between the solution of the teacher, X_{Ti}, and the mean solution of the students, M_i, can be expressed according to Equation (1).

$$D_i = r_i(X_{Ti} - T_F \cdot M_i) \tag{1}$$

where r_i is a random number in the range [0, 1] for iteration i and T_F is a teaching factor for iteration i, which is randomly set to either 1 or 2 according to Equation (2).

$$T_F = round(1 + rand(0,1)) \tag{2}$$

where $rand(0,1)$ is number randomly chosen from a uniform distribution between 0 and 1, and $round(\cdot)$ is a function that rounds its input to the nearest integer.

Based on the difference D_i, the existing solution of each student k in iteration i, X_{ki}, with $k \in \{1,2,\ldots,n\}$, is updated in the teacher phase according to Equation (3):

$$X_{ki}^{+} = X_{ki} + D_i \tag{3}$$

where X_{ki}^{+} is the updated value of X_{ki}.

If $f(X_{ki}^{+})$ is better than $f(X_{ki})$, X_{ki}^{+} is accepted and replaces X_{ki}. Otherwise, X_{ki}^{+} is discarded.

2.1.2. Student Phase:

In this phase, the algorithm simulates the learning of the students through interaction with one another. During this phase, students gain knowledge by discussing with students who have more knowledge [Rao and Patel, 2013].

Consider a pair of students y and z. Let X_{yi} and X_{zi} be the solutions of students y and z at iteration i, respectively. If $f(X_{yi})$ is better than $f(X_{zi})$, the solution of student z is updated according to Equation (4). Then, X_{zi}^{+} will replace X_{zi} if $f(X_{zi}^{+})$ is better than $f(X_{zi})$. Similarly, if

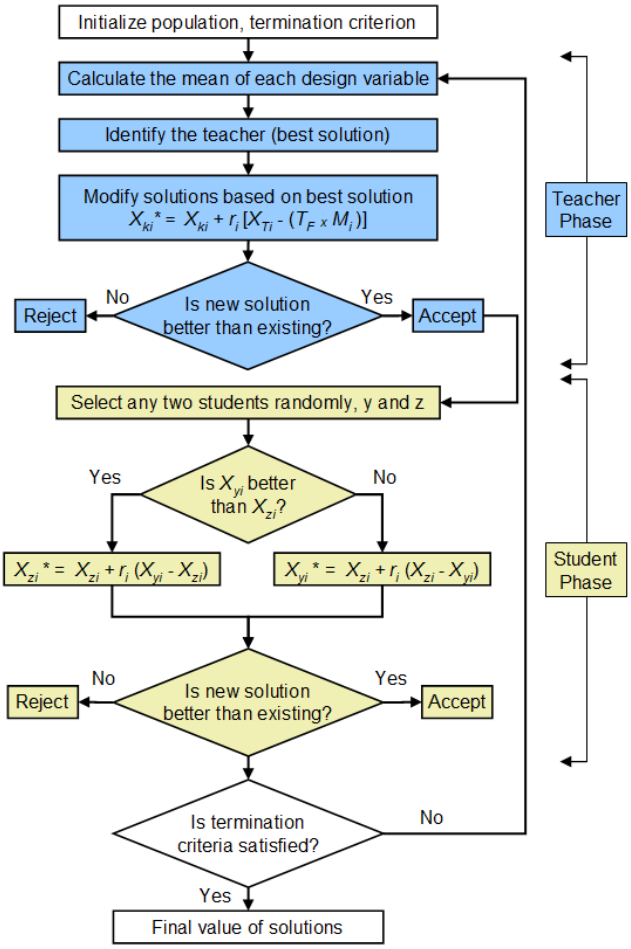

Figure 1: Flowchart for the TLBO algorithm [Rao et al., 2011]

$f(X_{zi})$ is better than $f(X_{yi})$, the solution of student y is updated according to Equation (5). Then, X_{yi}^+ will replace X_{yi} if $f(X_{yi}^+)$ is better than $f(X_{yi})$.

$$X_{zi}^+ = X_{zi} + r_i(X_{yi} - X_{zi}) \tag{4}$$

$$X_{yi}^+ = X_{yi} + r_i(X_{zi} - X_{yi}) \tag{5}$$

During the Student Phase, a partner is randomly chosen to be paired with each student. Therefore, if a population of n students is considered, then n pairs of students are evaluated.

2.2. Job Shop Scheduling Problem

The job shop scheduling problem is one of the best known and most difficult problem in the scheduling area [Cruz-Chavez et al., 2007]. It is classified into the NP-complete group [Garey et al., 1976].

A job shop scheduling problem consists of a set of j jobs and m machines. Each job contains an ordered list of operations, and each operation must be processed on a specific machine. Each job must go through the machines in the specified sequence to be completely processed [Baykasoğlu et al., 2014]. The operation belonging to the j-th job which is executed on machine m is denoted by $u(m,j)$ and has a duration $d(m,j)$. The goal is to find a schedule that minimizes the makespan (the completion time of all jobs). The job shop scheduling problem considers the following constraints on jobs and machines [Baykasoğlu et al., 2014], [Blazewicz et al., 1996]:

- All jobs are independent, and available for processing at time zero.
- Each job can be performed only on one machine at a time.
- Processing times are deterministic and independent.
- Setup times and removal times are included in processing times.
- Transportation times are negligible.
- Operations cannot be interrupted (preemption is not allowed).
- Every machine can execute only one operation at a time.
- The execution order of the operations for each job must be respected.
- All machines are always available (i.e. failures do not occur).
- There are no precedence constraints among operations of different jobs.

Let $J = \{1,\ldots,j\}$ be the set of jobs, $M = \{1,\ldots,m\}$ be the set of machines and $U = \{0,1,\ldots,j\times m,(j\times m)+1\}$ be the set of operations to be scheduled, where 0 and $(j\times m)+1$ are dummy initial and final operations, respectively. Also, let d_u and f_u be real numbers that denote the duration and the finish time of operation u, respectively.

The operations are interrelated by two kinds of constraints. First, the precedence constraints, which force the execution of each operation u to start only after all predecessor operations, P_u, are completed. Second, operation u can only be scheduled if the machine it requires is idle.

Let $A(t)$ be the set of operations being processed at time $t \in \mathbb{R}$, and $a_{u,m}$ be a binary variable which assumes value 1 if operation u is required to be processed on machine m and zero otherwise. A schedule may be represented by a vector of finish times $F = \left[f_0,\ldots,f_{(j\times m)+1}\right]$. The job shop scheduling problem can be formally stated as follows [Gonçalves et al., 2005]:

$$\text{Minimize:} \qquad f_{(j\times m)+1} \tag{6}$$

$$\text{Subjected to:}$$

$$f_u - d_u \geq f_x, \qquad \forall x \in P_u \tag{7}$$

$$\sum_{u\in A(t)} a_{u,m} \leq 1, \qquad m \in M; t \geq 0 \tag{8}$$

$$f_u \geq 0, \quad u = 1,\ldots,(j\times m)+1 \tag{9}$$

where Equation (6) is the objective function which minimizes the makespan, Equation (7) represents the precedence relations between operations, Equation (8) imposes that one machine can only process one operation at a time, and Equation (9) imposes non negative finishing times.

2.2.1. Solution Representation for the Job Shop Scheduling Problem

The solution representation adopted in this paper for the job shop scheduling problem is based on the priority matrix approach [Baykasoğlu et al., 2014]. Let X_k be an $m \times j$ real-number matrix which represents the solution of student k for a job shop scheduling problem containing m machines and j jobs, i.e.:

$$X_k = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1j} \\ x_{21} & x_{22} & \cdots & x_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mj} \end{bmatrix}$$

where each element x_{ri}, with $1 \leq r \leq m$ and $1 \leq i \leq j$ represents the priority of the operation belonging to the i-th job that is executed by the r-th machine.

The total number of possible schedules (considering both feasible and infeasible schedules) for a job shop scheduling problem with m machines and j jobs is $(j!)^m$, which makes it extremely difficult to find the optimum solution using an exhaustive search even for small instances of the problem. To reduce the search space, feasible schedules are classified into four types [Gonçalves et al., 2005]:

- **Inadmissible schedules:** These schedules contain excessive idle times. Inadmissible schedules may be improved by forward-shifting operations until no excess idle times exist.
- **Semi-active schedules:** These schedules can be obtained by sequencing the operations as early as possible. Semi-active schedules contain no excess idle time. However, they can be improved by shifting some operations to the front without delaying others.
- **Active schedules:** In this type of schedule, no operation can be started earlier without delaying some other operation or violating a precedence constraint. An optimal schedule is always an active schedule, so the search space can be safely limited to the set of all active schedules.
- **Non-delay schedules:** In this type of schedule, no machine is kept idle when it could start processing an operation. Non-delay schedules are always active.

Figure 2 shows the relationship among the different types of schedules. It is only necessary to limit the search space to the set of active schedules since optimal solutions are guaranteed to be active schedules [French, 1987]. In this paper, in order to convert the priority matrix X_k into an active schedule, the G&T algorithm is be used. The G&T algorithm is briefly described below [Giffler and Thompson, 1960].

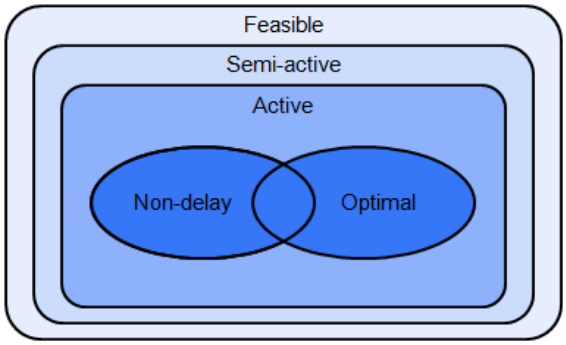

Figure 2: Relationship among schedule types

2.2.2. The G&T Algorithm

Giffler and Thompson [1960] developed a recursive algorithm to generate active schedules systematically. The notation adopted in the G&T algorithm is as follows:

- $u(m,j)$ is the operation belonging to the j-th job that must be executed on the m-th machine.
- S is the partial schedule which contains all operations already scheduled.
- Z is the set of operations that can be scheduled, i.e. the set of operations without predecessor restrictions.
- $e(m,j)$ is the earliest time at which operation $u(m,j)$ belonging to Z can be started.
- $d(m,j)$ is the duration (or execution time) of operation $u(m,j)$.
- $f(m,j)$ is the earliest time at which operation $u(m,j)$ belonging to Z can be finished, i.e. $f(m,j) = e(m,j) + d(m,j)$.

With the notation defined, the steps to implement the G&T algorithm are described in Algorithm 1:

Algorithm 1 G&T algorithm

1: **Step 1:** Initialize $S = \emptyset$ and Z to the subset of all operations without predecessors.
2: **while** $Z \neq \emptyset$ **do**
3: **Step 2:** Find the operation $u(m,j)^+ \in Z$ with the earliest possible completion time $f(m,j)^+$ and the machine m^+ on which $u(m,j)^+$ shall be executed.
4: **Step 3a:** Identify the subset of operations $Y \in Z$ such that all operations in Y are executed on machine $m^\star$ and for which $e(m,j) < f^+$.
5: **Step 3b:** Choose the operation $u(m,j)^{++}$ from Y with the largest priority value.
6: **Step 3c:** Add $u(m,j)^{++}$ to S.
7: **Step 3d:** Assign $e(m,j)^{++}$ as the starting time of operation $u(m,j)^{++}$.
8: **Step 4:** Delete $u(m,j)^{++}$ from Z and include its immediate successor, if any, in Z.
9: **end while**

To illustrate the application of the G&T algorithm, consider a job shop scheduling problem with 3 jobs and 3 machines. The sequence of operations for each job are as follows:

$$j_1 = [(2,3),(1,7),(3,2)]$$
$$j_2 = [(1,2),(3,5),(2,4)]$$
$$j_3 = [(2,5),(3,4),(1,5)]$$

where each pair (m,d) for each job j_i represents the machine m in which the operation shall be executed and the duration d of the operation. The operations of each job must be executed in the presented order. For instance, in this example job j_1 is processed in machine m_2 for 3 time units, then in machine m_1 for 7 time units, and finally in machine m_3 for 2 time units.

Also, consider the following priority matrix:

$$X_k = \begin{bmatrix} 0.11 & 4.70 & 0.67 \\ 6.17 & 9.23 & 1.23 \\ 4.19 & 3.01 & 3.47 \end{bmatrix}$$

where each element x_{mj} represents the priority of the operation belonging to the j-th job which requires to be executed by the m-th machine.

Initialization:
Step 1: $S = \emptyset$ and $Z = \{u(2,1),u(1,2),u(2,3)\}$.

Iteration 1:
Step 2: $f(2,1) = 3$, $f(1,2) = 2$ and $f(2,3) = 5$. Therefore, $f^\star = 2$ and $m^\star = 1$.
Step 3: $Y = \{u(1,2)\}$. Operation $u(1,2)$ is added to S, with $e(1,2) = 0$.
Step 4: $u(1,2)$ is deleted from Z and $u(3,2)$ is added to Z. For the next iteration, $Z = \{u(2,1),u(3,2),u(2,3)\}$.

Iteration 2:
Step 2: $f(2,1) = 3$, $f(3,2) = 7$ and $f(2,3) = 5$. Thus, $f^\star = 3$ and $m^\star = 2$.
Step 3: $Y = \{u(2,1),u(2,3\}$. Operation $u(2,1)$ has the highest priority (6.17) and is added to S, with $e(2,1) = 0$.
Step 4: $u(2,1)$ is deleted from Z and $u(1,1)$ is added to Z. For the next iteration, $Z = \{u(1,1),u(3,2),u(2,3)\}$.

Iteration 3:
Step 2: $f(1,1) = 10$, $f(3,2) = 7$ and $f(2,3) = 8$. Thus, $f^\star = 7$ and $m^\star = 3$.
Step 3: $Y = \{u(3,2)\}$. Operation $u(3,2)$ is added to S, with $e(3,2) = 2$.
Step 4: $u(3,2)$ is deleted from Z and $u(2,2)$ is added to Z. For the next iteration, $Z = \{u(1,1),u(2,2),u(2,3)\}$.

Iteration 4:
Step 2: $f(1,1) = 10$, $f(2,2) = 11$ and $f(2,3) = 8$. Thus, $f^\star = 8$ and $m^\star = 2$.
Step 3: $Y = \{u(2,2),(2,3)\}$. Operation $u(2,2)$ has the highest priority (9.23) and is added to S, with $e(2,2) = 7$.
Step 4: $u(2,2)$ is deleted from Z. No operation is added to Z because $u(2,2)$ is the last operation of job j_2. For the next iteration, $Z = \{u(1,1),u(2,3)\}$.

Iteration 5:
Step 2: $f(1,1) = 10$ and $f(2,3) = 16$. Thus, $f^\star = 10$ and $m^\star = 1$.
Step 3: $Y = \{u(1,1)\}$. Operation $u(1,1)$ is added to S, with $e(1,1) = 3$.
Step 4: $u(1,1)$ is deleted from Z and $u(3,1)$ is added to Z. For the next iteration, $Z = \{u(3,1),u(2,3)\}$.

Iteration 6:
Step 2: $f(3,1) = 12$ and $f(2,3) = 16$. Thus, $f^\star = 12$ and $m^\star = 3$.
Step 3: $Y = \{u(3,1)\}$. Operation $u(3,1)$ is added to S, with $e(3,1) = 10$.
Step 4: $u(3,1)$ is deleted from Z. No operation is added to Z because $u(3,1)$ is the last operation of job j_1. For the next iteration, $Z = \{u(2,3)\}$.

Iterations 7, 8 and 9:
Now, all operations that have not been scheduled belong to job j_3. Therefore, in iterations 7, 8 and 9 the operations of job j_3 will be scheduled following the sequence of job j_3, i.e. $u(2,3)$, $u(3,3)$ and $u(1,3)$.

Figure 3 shows the Gantt chart for the solution, which has a makespan of 25.

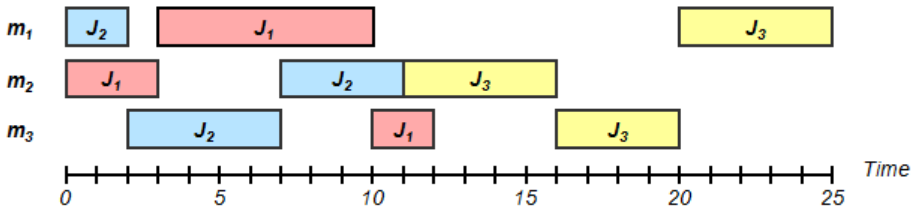

Figure 3: Gantt chart for the flow shop example

3. Proposed Method

When the priority vector approach is used, the priority vector associated with student k, denoted by X_k, must be converted into a job sequence S to be evaluated by the objective function, as described in section 2.2.1.

Each priority vector leads to a unique job sequence. However, two or more different priority vectors will lead to the same job sequence if their elements are in the same descending order. Consider the two priority vectors $v_1 = [.98, .23, .51, .48]$ and $v_2 = [.67, .02, .54, .17]$. Although the vectors are different, their elements are in the same descending order (i.e., the first element has the highest value, followed by the third, the fourth and the second elements). Therefore, both priority vectors v_1 and v_2 are mapped into the same sequence $[1,3,4,2]$.

The main idea behind the method proposed in this paper is to insert one additional step, called Change Verification, in order to compare the job sequences obtained from the updated and the original priority vectors of each student in both the Teacher Phase and the Student Phase. The TLBO algorithm calls the objective function only if the troubleshooting strategies differ from each other.

The goal is to eliminate unnecessary calls to the objective function. The reason is that some objective functions may have high computational costs [Hawe and Sykulski, 2006]. The Change Verification step consists in comparing two vectors. This task has a lower computational cost when compared to the computation of the makespan, which is the objective function for the job shop scheduling problem considered in this paper.

When the computational cost of the objective function is high, the computational cost saved by eliminating the unnecessary calls to the objective function compensates the computational cost added by the Change Verification step.

4. Numerical Experiments

Numerical experiments using twenty benchmark instances of the job shop scheduling problem were carried out in order to evaluate the performance of the proposed method in terms of total computational time. All the experiments reported in this paper were carried out on a personal computer with Intel® Core™ i3, 1.9 GHz processor and 4GB RAM, running Windows 8. The algorithm was coded in Matlab®. Table 1 summarizes the main characteristics of each benchmark instance used in the experiments. These instances were obtained from the OR-Library.

The population size, denoted by PS, is the only parameter required by the TLBO algorithm. The stop criterion used in the experiments was the maximum number of generations, denoted by GN. The population size PS used in each benchmark instance was $5 \times j$, where j is the number of jobs. A maximum number of generations of 400 was used for all benchmark instances.

Table 2 shows the makespan obtained for all benchmark instances using the chosen values for GN and PS. For each benchmark instance, a Monte Carlo method with 20 iterations was carried out for both the original TLBO and the proposed method. Column BKS shows the Best Known Solution reported in the literature for each instance. The average and the best makespan computed for each instance using both the original TLBO and proposed method are presented. The minimum values for both the average and the best makespan are shown in bold. It can be observed that

Table 1: Main features of the job shop benchmark instances used in the experiments

Instance	Jobs	Machines	Instance	Jobs	Machines
abz5	10	10	la31	30	10
abz7	20	15	la35	30	10
ft06	6	6	la36	15	15
ft20	20	5	orb05	10	10
la06	15	5	orb09	10	10
la11	20	5	swv01	20	10
la12	20	5	swv04	20	10
la16	10	10	swv05	20	10
la22	15	10	swv16	50	10
la26	20	10	yn1	20	20

both the original TLBO and the proposed method presented similar performances. This result was expected because the proposed method only eliminates the objective function calls that would not lead to better new solutions. Therefore, the quality of the final solution is not affected.

Table 2: Performance comparison between original TLBO and the proposed method in terms of makespan

Instance	BKS	Basic TLBO		Proposed TLBO	
		Avg.	Best	Avg.	Best
abz5	1234	**1344.3**	1313	1354.9	**1305**
abz7	656	**831.5**	**809**	831.8	818
ft06	55	**56.5**	**55**	57.1	**55**
ft20	1165	1231.2	1195	**1227.0**	**1193**
la06	926	926.8	**926**	**926.2**	**926**
la11	1222	**1222.3**	**1222**	1223.3	**1222**
la12	1039	1042.3	**1039**	**1039.8**	**1039**
la16	945	**1016.8**	995	1017.5	**980**
la22	927	1107.9	**1058**	**1102.1**	1060
la26	1218	**1456.0**	**1397**	1465.2	1410
la31	1784	**1950.1**	1919	1952.5	**1900**
la35	1888	2037.8	**1975**	**2033.9**	1990
la36	1268	**1490.3**	1440	1500.3	**1430**
orb05	887	966.2	**931**	**961.9**	933
orb09	934	**1011.3**	**975**	1020.5	989
swv01	1407	**1687.4**	**1636**	1690.0	1660
swv04	1470	1768.1	1725	**1752.0**	**1704**
swv05	1424	1734.3	1697	**1727.8**	**1680**
swv16	2924	2966.6	2939	**2956.8**	**2926**
yn1	884	**1112.8**	1098	1119.7	**1094**

Table 3 shows the average total simulation time, the average time spent running the objective function and the average number of objective function calls for both the original TLBO and the proposed method. For the latter, the average time spent running the Change Verification step is also presented.

It can be noticed that the proposed method presented a better performance for all instances in terms of simulation time and objective function calls when compared to the original TLBO algorithm. The average reduction provided by the proposed method in total simulation time and objective function calls were 11.7% and 12.2%, respectively. Also, the average simulation time

spent running the Change Verification routine is extremely low in comparison with the average total simulation time of the proposed method.

Table 3: Performance comparison between original TLBO and the proposed method in terms of computational times (in seconds) and number of objective function calls

Instance	Basic TLBO			Proposed TLBO			
	Total simulation time (s)	Obj. funct. simulation time (s)	Objective function calls	Total simulation time (s)	Obj. funct. simulation time (s)	Change simulation time (s)	Objective function calls
abz5	191.6	191.0	30050	157.1	155.7	0.14	25063
abz7	1336.6	1335.3	60100	1237.7	1233.5	0.32	56195
ft06	39.9	39.5	18030	25.6	25.1	0.05	11099
ft20	478.7	477.5	60100	438.5	435.1	0.30	54150
la06	241.7	240.8	45075	199.5	197.4	0.18	34956
la11	476.8	475.7	60100	411.2	408.0	0.28	51961
la12	475.3	474.2	60100	423.6	420.4	0.27	53326
la16	194.2	193.7	30050	171.0	169.3	0.15	24425
la22	477.8	477.9	45075	442.9	440.1	0.24	41504
la26	916.1	914.8	60100	852.9	848.9	0.34	56496
la31	2278.7	2276.6	90150	2078.5	2071.3	0.57	84901
la35	2331.4	2329.2	90150	2101.0	2093.2	0.60	84928
la36	742.2	741.1	45075	687.2	683.9	0.29	41572
orb05	187.8	187.3	30050	157.0	155.6	0.12	24710
orb09	190.2	189.7	30050	166.4	164.9	0.14	26248
swv01	960.1	948.9	60100	874.7	870.6	0.35	56461
swv04	979.7	978.4	60100	892.4	888.2	0.36	48991
swv05	948.9	947.6	60100	889.1	885.0	0.34	56501
swv16	7875.2	7871.0	150250	7116.8	7099.4	1.08	141677
yn1	1799.9	1798.3	60100	1648.5	1643.0	0.44	56311

5. Conclusions

This paper presents a method to improve the computational efficiency of the TLBO algorithm in combinatorial problems in terms of simulation time. The proposed method is applicable when the priority vector approach is adopted.

Twenty different benchmark instances of the job shop scheduling problem were used in the numerical experiments to evaluate the performance of the proposed method. Simulations using the original version of the TLBO algorithm were also carried out to establish a reference baseline.

The results show that the proposed method outperformed the original TLBO algorithm in terms of simulation time and objective function calls in all benchmark instances. One important characteristic of the proposed method is that it does not affect the final solution of the algorithm since it only eliminates those calls to the objective function that would not lead to better solutions.

The proposed method becomes more attractive as the computational cost of the objective function increases. In these situations, the computational cost saved by eliminating the unnecessary calls to the objective function becomes more relevant. The simulation time required to run the Change Verification step is very small.

A possible extension for this paper is to evaluate the performance of the proposed method in different combinatorial problems. Another possibility is to investigate the use of the Change verification step in other metaheuristic optimization algorithms.

References

Baykasoğlu, A., Hamzadayi, A., and Köse, S. Y. (2014). Testing the performance of teaching learning based optimization (TLBO) algorithm on combinatorial problems: Flow shop and job shop scheduling cases. *Information Sciences*, 276:204–218.

Blazewicz, J., Domschke, W., and Pesch, E. (1996). The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93:1–33.

Chen, D., Zou, F., Li, Z., Wang, J., and Li, S. (2015). An improved teaching-learning-based optimization algorithm for solving global optimization problem. *Information Sciences*, 297:171–190.

Cruz-Chavez, M. A., Martinez-Rangel, M. G., Hernandez, J. A., Zavala-Diaz, J. C., and Diaz-Parra, O. (2007). Scheduling algorithm for the job shop scheduling problem. In *Electronics, Robotics and Automotive Mechanics Conference (CERMA 2007)*, p. 336–341.

French, S. (1987). *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop*. Wiley & Sons, Chichester.

Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129.

Giffler, B. and Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503.

Gonçalves, J. F., Mendes, J. J. M., and Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167(1):77–95.

Hawe, G. I. and Sykulski, J. K. (2006). The consideration of surrogate model accuracy in single-objective electromagnetic design optimization. In *International Conference on Computational Electromagnetics (CEM)*, p. 1–2.

Rao, R. V. and Patel, V. (2012). An elitist teaching-learning based optimization algorithm for solving complex constrained optimization problems. *International Journal of Industrial Engineering Computations*, 3:535–560.

Rao, R. V. and Patel, V. (2013). An improved teaching-learning-based optimization algorithm for solving unconstrained optimization problems. *Scientia Iranica*, 20:710–720.

Rao, R. V., Savsani, V. J., and Vakharia, D. P. (2012). Teaching-learning-based optimization: An optimization method for continuous non-linear large scale problems. *Information Sciences*, 183: 1–15.

Rao, R. V., Vakharia, D. P., and Savsani, V. J. (2011). Teaching-learning-based optimization: a novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 43: 303–315.

Sahu, B. K., Pati, S., Mohanty, P. K., and Panda, S. (2015). Teaching-learning based optimization algorithm based fuzzy-PID controller for automatic generation control of multi-area power system. *Applied Soft Computing*, 27:240–249.

Sapathy, S. C., Naik, A., and Parvathi, K. (2013). A teaching learning based optimization based on orthogonal design for solving global optimization problems. *SpringerPlus*, 2(130).