EFFECTIVE LEAGUE CHAMPIONSHIP ALGORITHM AND LOWER BOUND PROCEDURE FOR SCHEDULING A SINGLE BATCH-PROCESSING MACHINE WITH NON-IDENTICAL JOB SIZES AND JOB REJECTION

SAEED AFKHAMI, ALI HUSSEINZADEH KASHAN* AND BAKHTIAR OSTADI

Abstract. We address the scheduling problem of a set of non-identical size jobs on a single batch-processing machine (SBPM) wherein the scheduler can make decision whether to schedule a job in batches or not to schedule it with a job-dependent penalty. The processing time of a batch is the greatest job processing time in that batch (parallel batching or p-batching). The scheduler wants to minimize a given objective function \( f \), where \( f \) is the total rejection penalties of the rejected jobs (rejection cost) plus the makespan of the scheduled ones. We formulate the aforementioned problem as a 0–1 mixed integer programming model. We also apply an effective dynamic programming algorithm (DPA) to calculate a lower bound (LB) on the optimal cost of the problem. To tackle the problem, we propose a grouping algorithm, based on league championship algorithm (LCA), with new updating equations maintaining the major characteristics of the original updating equations of the LCA and well-suited to the structure of the problem. For small problems, performance of the proposed LCA is compared with GAMS/CPLEX solver. For large-scale instances, a genetic algorithm is adopted as a basis for comparison. Simulated experiments confirm the performance of the proposed methods.

Mathematics Subject Classification. 90B35.

Received August 7, 2021. Accepted April 6, 2023.

1. Introduction

Batch processing machine scheduling problem (BPMS) is one of the most famous scheduling problems. While machines in the classical scheduling problems can process only one job simultaneously, the batch processing machine (BPM) is a machine that is capable of handling more than one job simultaneously. A group of jobs processed simultaneously is called a batch. There is an assumption in the most of the classical BPMSPs that all jobs must be processed (see [31], a survey paper on this field). Nevertheless, in the real world, the manufacturers regularly reject some jobs with relatively larger processing time and smaller profits because of the limitation of resources and maximization of profits. In these cases, the manufacturers may have to make an intelligent decision about which jobs to accept and which ones to be rejected. Each rejected job can be either outsourced or not processed at all. Both situations will result in a rejection cost. Manufacturers have to make two decisions:

*Corresponding author: a.kashan@modares.ac.ir
which jobs to accept, and how to schedule the accepted jobs. To reach the best solution, the manufacturers must coordinate these two decisions.

In comparison to the classic BPMSP, BPMSP with rejection is closer to the real world and has more practical application. However, the research on BPMSP rarely consider the possibility of job rejection. In this research, we consider the scheduling of non-identical size jobs on an SBPM wherein the job rejection is allowed. Our considered objective function, which must be minimized, is the summation of the total penalty cost ($R_{\text{tot}}$) of the rejected jobs and the makespan ($C_{\text{max}}$) of the scheduled ones. All the jobs are available at time zero, and each job has its own processing time, size, and penalty cost. The manufacturer can either reject a job and pay the penalty cost or accept and process it in the batches. The machine has a capacity and the summation of the job sizes in a batch must not be more than this capacity. A batch processing time is exactly equal to the greatest job processing time in that batch. Research problem description and its mathematical model will be given in Section 3.

The rest of this paper is organized as follows: the related works are reviewed in Section 2. Problem characteristics and its mathematical modeling are presented in Section 3. The algorithm of calculating an LB on the optimal cost is given in Section 4. The fundamentals of the LCA are briefly described in Section 5. The details of the proposed LCA (GLCA) are discussed in Section 6. Performance of the proposed LCA is assessed in Section 7. Finally, the conclusions are presented in Section 8.

2. Literature review

In the past three decades, a considerable amount of literature has been published on scheduling problems, particularly scheduling on BPMs. Uzsoy [33] first introduced scheduling problem of a set of non-identical size jobs on an SBPM to minimize $C_{\text{max}}$, and established the NP-hardness proof of this problem. After that, more and more researchers began to show their interests in SBPM scheduling problem (SBMSP). Over the years, more constraints were being added to the problem, which makes the problem more complex. In another paper, Ghazvini and Dupont [8] considered the total flow time as the objective function of the same problem. Parsa et al. [29] developed a branch and price (B&P) algorithm to minimize $C_{\text{max}}$ of an SBPM.

The above-mentioned approaches are applicable only to a specific problem. With this in mind, and also the complexity of the problem, many researchers have applied metaheuristic algorithms. Melouk et al. [27] proposed a simulated annealing approach to minimize $C_{\text{max}}$ of an SBMSP with non-identical job sizes, and compared their computational results with CPLEX optimization software. Husseinzadeh Kashan et al. [13] developed an effective batch based hybrid genetic algorithm (BHGA) for the same problem and demonstrated that their algorithm is superior to the SA algorithm developed by Melouk et al. [27]. Parsa et al. [30] proposed an ant colony method for the case of an SBM with different job sizes in order to minimize total flow time. Jia et al. [20] investigated the parallel BPMs scheduling problem with unequal weights, identical processing time, job delivery, and non-identical job sizes. To tackle the problem, they proposed two heuristics and a metaheuristic algorithm based on ant colony optimization (ACO). There are also some other papers in the literature that apply metaheuristics to solve the batch-scheduling problem, such as [2,35,38], and etc.

The scheduling problem with rejection was first discussed by Bartal et al. [3] for identical parallel machines. Their objective was the minimization of $R_{\text{tot}} + C_{\text{max}}$. There are a number of research works that focus on the scheduling problem with rejection, such as [6,37], and etc.

There is already a large volume of literature on batch scheduling and a large volume of literature on scheduling with rejection. However, there is still limited literature that considers batch scheduling and job rejection simultaneously. Lu et al. [24] considered the unbounded parallel-BMPSP with job rejection and release dates. The objective was the minimization of $C_{\text{max}} + R_{\text{tot}}$. They proved the NP-hardness of the problem and presented a two-approximation algorithm. Cao and Yang [5] considered the bounded parallel-BMPSP with job rejection and release dates. Their objective function was the same as that of considered by Lu et al. [24]. They generated a polynomial-time approximation scheme (PTAS) for the problem. Jia et al. [19] investigated the scheduling problem of a set of non-identical size jobs on the parallel BPMs with job rejection in order to minimize $C_{\text{max}}$ and
They considered two problems, P1 and P2. In P1, they combined two objectives linearly, and in P2, they minimized two objectives simultaneously in order to find Pareto non-dominated solutions. They proposed two different multi-objective ACO algorithms for P1 and P2. Kong et al. [22] studied a single bounded p-batching scheduling problem with non-identical job sizes, job rejection, setup time, and deteriorating jobs. They proposed a hybrid algorithm to minimize $C_{\text{max}} + R_{\text{tot}}$. Jin et al. [21] investigated the single machine p-batching scheduling problem considering job rejection and non-identical job sizes. They developed three heuristic algorithms to minimize $R_{\text{tot}} + C_{\text{max}}$. Ou [28] considered the SBPMSP with job rejection, different release dates, identical job sizes, and a fixed setup time. He presented effective approximation algorithms for solving the problem. There are also some other papers in the literature that discussed the BPMSP with job rejection, such as [9,25,32,36], and etc. However, it can be found from the literature review that there are a limited number of works on the batch-scheduling problem with job rejection. Moreover, although the NP-hardness of the research problem is proved by Lu et al. [24], based on our knowledge, there is no existing metaheuristic in the literature for the single batch-scheduling problem with rejection. Therefore, in this paper we consider a scheduling problem of a set of non-identical size jobs on an SBPM with rejection to minimize the $R_{\text{tot}}$ of the rejected jobs (rejection cost) plus the $C_{\text{max}}$ of the scheduled ones, and try to find a satisfactory solution for this problem. To do so, a grouping league championship algorithm is proposed.

3. Problem description

In this section, first, the characteristics of the SBPMSP with job rejection are delineated. Then, a 0–1 mixed-integer programming formulation is presented for this problem. There are an SBPM and a collection of jobs $J$ that must be processed in batches on machine. Each job $j$ ($j = 1, \ldots, n$) is available at time zero and has its own size $s_j$, processing time $p_j$ and a rejection penalty $e_j > 0$. We assume that all $s_j$, $p_j$, and $e_j$ are positive integers. Each job $J_j$ either can be rejected at the cost of $e_j$ or accepted. The accepted jobs must be processed in batches on the machine. The processing time of a batch is exactly equal to the greatest job processing time in that batch, and the summation over the job sizes in a batch must not be greater than the machine capacity $B$. It is assumed that the size of each job is smaller than $B$ and jobs cannot be separated into two or more parts. When the processing of a batch is started, it cannot be interrupted for any reason until the processing is completed. Therefore, if job $j$ is processed in batch $b$, its completion time will be equal to that of the batch $b$. The objective pursued in this study is the minimization of $R_{\text{tot}}$ of the rejected jobs plus the $C_{\text{max}}$ of the scheduled ones. Based on the general notation of scheduling problems, the research problem is denoted by $1|\text{p-batch, rej}, \sum s_j \leq B|C_{\text{max}} + R_{\text{tot}}$.

### 3.1. Parameters, indices, and decision variables

Parameters, indices, and decision variables of the mathematical model of the research problem are defined as follows:

**Parameters and notations:**

- $N$: Number of jobs
- $J$: Index of jobs
- $B$: Index of batches
- $p_j$: Processing time of job $j$
- $s_j$: Size of job $j$
- $e_j$: Rejection penalty of job $j$
- $B$: Capacity of a batch
- $M$: A sufficiently large number
**Decision variables:**

\[ X_{jb} = \begin{cases} 
1 & \text{If job } j \text{ is assigned to the batch } b \\
0 & \text{Otherwise} 
\end{cases} \]

\[ Y_j = \begin{cases} 
1 & \text{If job } j \text{ is selected for processing} \\
0 & \text{Otherwise}. 
\end{cases} \]

**Dependent variable:**

\( P_b \)  
Processing time of batch \( b \)

### 3.2. Mixed integer linear programming formulation (MILP)

Minimize

\[
\sum_{b=1}^{n} P_b + \sum_{j=1}^{n} e_j (1 - Y_j) 
\]  
(1)

Subject to

\[
\sum_{b=1}^{n} X_{jb} = Y_j, \quad j = 1, \ldots, n \]  
(2)

\[
\sum_{j=1}^{n} s_j X_{jb} \leq B, \quad b = 1, \ldots, n \]  
(3)

\[
X_{jb} p_j \leq P_b, \quad j = 1, \ldots, n; \ b = 1, \ldots, n \]  
(4)

\[
P_b \geq 0, \quad b = 1, \ldots, n \]  
(5)

\[
X_{jb} \in \{0, 1\}, \quad j = 1, \ldots, n; \ b = 1, \ldots, n \]  
(6)

\[
Y_j \in \{0, 1\}, \quad b = 1, \ldots, n \]  
(7)

The objective function (1) minimizes the sum of the total penalty cost of the rejected jobs and the \( C_{\text{max}} \) of the scheduled ones. Constraint (2) guarantees that the rejected jobs are not scheduled and each selected job is assigned to one and exactly one batch. Capacity restriction of batches is enforced by the constraint (3). Constraint (4) sets the processing time of batch \( b \) equal to the greatest job processing time in that batch. Domains of the variables are defined by the constraint sets (5)–(7). This model contains \( n^2 + 2n \) variables and \( n^2 + 2n \) constraints.

### 4. LOWER BOUND (LB)

In this section, an algorithm based on dynamic programming (DP) is proposed to calculate an LB on the optimal cost of the problem on hand. We will use this LB as a tool for evaluating the performance of the proposed algorithms. First, we presume that jobs can be split and processed in different batches. As we know the optimal solution of this relaxed problem can be used as an LB of the original problem. In order to calculate the optimal solution of the relaxed problem, a dynamic programming algorithm is proposed based on the method used by Lu et al. [26].

The problem of \( 1|p\text{-batch}, \text{split}, s_j \leq B|C_{\text{max}} \) has been solved optimally by the following FBLPT (Full Batch Longest Processing Time) rule [23].

**Step 1.** Re-index the jobs in such a way that \( p_1 \geq p_2 \geq \cdots \geq p_n \).

**Step 2.** Open a batch, and then fill it with the longest jobs among the remaining jobs. If there is not enough room in the batch for some job, put part of the job in the batch in such a manner that the batch is at full capacity.

**Step 3.** Repeat Step 2 while the job list is empty.
By considering the optimality of the above FBLPT-rule, Lu et al. stated the following lemma.

**Lemma 1.** There is an optimal solution for 1\(|p\)-batch, split, \(s_j \leq B|C_{\text{max}} + R_{\text{tot}}\) wherein the accepted jobs are scheduled using the FBLPT-rule [26].

Based on Lemma 1, first, we re-index the jobs such that \(p_1 \geq p_2 \geq \cdots \geq p_n\). Let \(F_j(S)\) denote the optimal objective function value when the jobs in consideration are \(J_1, \ldots, J_j\) and the total size of the accepted jobs is exactly \(S\). Now, consider the optimal schedule corresponding to \(F_j(S)\). We have two possible cases:

**Case 1.** \(J_j\) is rejected. In this case, since the total size of the accepted jobs among \(J_1, \ldots, J_{j-1}\) is still equal to \(S\), we have \(F_j(S) = F_{j-1}(S) + e_j\).

**Case 2.** \(J_j\) is accepted. In this case, the total size of the accepted jobs among \(J_1, \ldots, J_{j-1}\) is \(S - s_j\). Let \(P = \lceil S/B \rceil\) and \(Q = \lceil (S - s_j)/B \rceil\), where \([x]\) represents the minimum integer greater than or equal to \(x\).

Therefore, if \(S = s_j\), then \(F_j(S) = F_{j-1}(0) + p_j\). If \(S > s_j\) and \(P = Q\), \(J_j\) has to be processed in the current batch, which yields \(F_j(S) = F_{j-1}(S - s_j)\). If \(S > s_j\) and \(P > Q\), \(J_j\) cannot be assigned in the current batch, completely. Considering the FBLPT-rule, \(J_j\) has to be split, and a part of it starts a new batch and is processed alone in the opened batch. By doing so, we have \(F_j(S) = F_{j-1}(S - s_j) + p_j\).

Combining the above two cases, we can propose the following dynamic programming algorithm (DPA) to calculate the optimal solution of 1\(|p\)-batch, split, \(s_j \leq B|C_{\text{max}} + R_{\text{tot}}\).

**Dynamic Programming Algorithm (DPA)**

**The boundary conditions:**
\[F_1(0) = e_1, \ F_1(s_1) = p_1 \text{ and } F_1(S) = +\infty \text{ for any } S \neq \{0, s_1\}\]

**The recursive function:**
\[F_j(S) = \begin{cases} F_{j-1}(S) + e_j & \text{if } S < s_j \\ \min \{F_{j-1}(0) + p_j, F_{j-1}(S) + e_j\} & \text{if } S = s_j \\ \min \{F_{j-1}(S - s_j), F_{j-1}(S) + e_j\} & \text{if } S > s_j \text{ and } P = Q \\ \min \{F_{j-1}(S - s_j) + p_j, F_{j-1}(S) + e_j\} & \text{if } S > s_j \text{ and } P > Q \end{cases}\]

The optimal objective function value is given by \(\min \{F_n(S) : 0 \leq S \leq \sum_{j=1}^n s_j \}\).

**Theorem 1.** The proposed DPA is an LB on the optimal objective value of 1\(|p\)-batch, rej, \(s_j \leq B|C_{\text{max}} + R_{\text{tot}}\).

**Proof.** Based on the above discussion, the proposed DPA finds the optimal solution of 1\(|p\)-batch, rej, split, \(s_j \leq B|C_{\text{max}} + R_{\text{tot}}\). This problem is a relaxed version of the original problem; therefore, its optimal solution is an LB on the optimal solution of the original problem. \(\square\)

**Theorem 2.** The DPA finds the LB of the original problem in \(O(n^2B)\).

**Proof.** It is obvious that \(1 \leq j \leq n\) and \(0 \leq S \leq \sum_{j=1}^n s_j\). On the other hand we know that for each \(j = 1, \ldots, n\), \(s_j\) is smaller than or equal to \(B\). therefore, \(S\) is always smaller than or equal to \(n \times B\), and the maximum number of states of the recursive function is \(n^2B\). Since each iteration takes a constant amount of time, the time complexity of the proposed algorithm is bounded by \(O(n^2B)\). \(\square\)

Although the proposed lower bound algorithm uses the notion of job splitting and assigns job parts to different batches, the output is not equivalent to the results obtained by solving the relaxed MILP model in which the integrality of \(X_{jb}\) variable is dropped and it can take a value between 0 and 1. The lower bound value obtained by DPA is far better than the bound obtained by the relaxed MILP model. This is because of the fact that...
when we relax the integrality of $X_{jb}$, constraint (4) which is for computation of the batch processing time, underestimates the batch processing times. It computes the batch processing time proportional to the part of the jobs it accommodates. For example, if $X_{jb} = 0.3$, then 30% of job $j$ is assigned to batch $b$. Assume that the processing time of job $j$ is the longest among all jobs in the batch. Also, assume that $X_{jb} > X_{kb}$ holds for all jobs such as $k$ in batch $b$ in the optimal solution of the relaxed MILP. Hence by (4), the processing time of batch $b$ is computed as $0.3p_j$ but not $p_j$. In this sense, such a relaxation performs too weak and is not competitive. However, in our proposed lower bound algorithm, each job is split into the job parts whose sizes are equal to 1 and whose processing times are equal to the processing time of the original job. In this sense, our lower bound algorithm is much tighter compared to the relaxed MILP model.

5. League Championship Algorithm (LCA)

LCA, originally introduced by Husseinzadeh Kashan [10] for continuous search spaces, is a population-based metaheuristic algorithm. Besides, it has been used also for discrete optimization problems (e.g. in [17]). There is a unique mapping between a typical evolutionary algorithm and LCA. First, the initial population of LCA (league) is created by a set of $N$ randomly generated solutions ($N$ is an even number). Each randomly generated solution corresponds to the formation of one of the $N$ teams. In LCA, the fitness function value is the playing strength related to a particular team formation. LCA uses a greedy approach for selection – the current best formation is only replaced with a better one. The algorithm stops after a certain number of seasons ($NS$), and each season contains $N - 1$ weeks (iterations). Therefore, there are $NS(N - 1)$ weeks of contests in LCA altogether. LCA has four main steps as described below.

5.1. Generating the league schedule

LCA imitates the process of sport league championships. A common denominator of all sports leagues is a league schedule. Therefore, the first step is to schedule the matches of the artificial league. To do so, LCA uses a method called single round-robin schedule. In each of $N - 1$ weeks of a sports league of size $N$ ($N$ is an even number), $N/2$ matches will be held in parallel; therefore, the single round-robin tournament needs $N(N - 1)/2$ matches.

The single round-robin scheduling process is simple. Figure 1 illustrates the algorithm for a sports league composed of 8 teams. Matches of the first week can be scheduled by paring off the teams, randomly. In Figure 1a, in the first week of the league, we have the following matches: 1 versus 5, 2 versus 6, and so on. In order to schedule the next week, fix team 1 and revolve the others clockwise. Then as shown in Figure 1b, teams 1, 5, 2, and 3 are paired with teams 6, 7, 8, and 4, respectively. To schedule the third week, once again revolve the order clockwise. So, as shown in Figure 1c, 1 plays 7, 6 plays 8 and so on. This process continues till reaching the initial state. By doing so, each team plays against all the other participants once a season. In LCA, the schedule created in the first season will be used in all seasons.

5.2. Winner and loser recognition

In the real world, when two sports teams competing against each other, the stronger team is more likely to win. LCA considers that the probability of winning for each team has a linear relationship with its playing strength (note that draw is not allowed in LCA). Suppose that there is a match between team $i$ and team $j$ at
week \( t \). Their formation strategies are \( x^i_t \) and \( x^j_t \), and their playing strengths are \( f(x^i_t) \) and \( f(x^j_t) \), respectively. The probability of team \( i \) winning \( (p^i_t) \), can be determined as follows:

\[
p^i_t = \frac{f(x^i_t) - \hat{f}}{f(x^i_t) + f(x^i_t) - 2\hat{f}}
\]

where \( \hat{f} \) represents the best objective function value ever found. A random number between zero and one is generated to determine the winner and loser teams. The team \( i \) is winner if the generated random number is not greater than \( p^i_t \); otherwise team \( i \) loses.

### 5.3. SWOT analysis

Before each match of a real sports league, coaches typically assess the strengths and weaknesses of each player and also the team as a whole (internal factors). They also evaluate the opportunities and threats by analyzing their next week rival (external factors). This analysis is the same as SWOT analysis. LCA also make a similar SWOT analysis before generating new team formation.

SWOT analysis for team \( i \) is displayed in Figure 2, in which the following notations are used.

- \( j \): the team that has played against team \( i \) at week \( t \)
- \( l \): the team that is playing against team \( i \) at week \( t+1 \)
- \( K \): the team that has played against team \( l \) at week \( t \)

### 5.4. Team reformation

In this step, based on the results obtained from SWOT analysis, four different conditions are possible. If both \( i \) and \( l \) have won their matches at week \( t \), then:

\[
x^{t+1}_{id} = b^t_{id} + y^t_{id}(c_1 r_1(b^t_{id} - b^t_{kd}) + c_1 r_2(b^t_{id} - b^t_{jd})) \quad \forall d = 1, \ldots, n
\]

Else if \( i \) has won and \( l \) has lost, then:

\[
x^{t+1}_{id} = b^t_{id} = y^t_{id}(c_2 r_1(b^t_{id} - b^t_{kd}) + c_1 r_2(b^t_{id} - b^t_{jd})) \quad \forall d = 1, \ldots, n
\]

Else if \( i \) has lost and \( l \) has won, then:

\[
x^{t+1}_{id} = -b^t_{id} = y^t_{id}(c_1 r_2(b^t_{id} - b^t_{kd}) + c_2 r_1(b^t_{id} - b^t_{jd})) \quad \forall d = 1, \ldots, n
\]
If both $i$ and $l$ have lost their matches at week $t$, then:

$$x_{id}^{t+1} - b_{id}^t = y_{id}^t (c_2 r_2 (b_{kd}^t - b_{id}^t) + c_2 r_1 (b_{jd}^t - b_{id}^t)) \quad \forall d = 1, \ldots, n \tag{12}$$

where, $d = 1, \ldots, n$ is the dimension index. $c_1$ and $c_2$ are constant coefficients and used to adjust the contribution of retreat and approach components. $r_1$ and $r_2$ are uniform random numbers between zero and one. Finally, $b_{id}^t$ is the best obtained solution for team $i$, until week $t$. Figure 3 shows the flowchart of LCA. Note that, in this paper, we use a special version of LCA called LCA/best, introduced by Husseinzadeh Kashan [11].

6. Adaptation of LCA for solving batch-scheduling problem with rejection (GLCA)

Falkenauer [7] defines grouping problems as the partitioning a collection $U$ of items into a set of mutually disjoint groups $G_i$, such that: $U = \bigcup G_i$ and $G_i \cap G_j = \emptyset$, $i \neq j$. In other words, the aim of these problems is to partition the items of the set $U$ into $D$ different groups ($1 \leq D \leq |U|$), with each item in exactly one group.

The research problem can be seen as a grouping problem and it was also shown to be NP-hard. Therefore, in this section, a grouping version of league championship algorithm (GLCA) is developed to obtain near optimal solutions for large-scale problems. Based on the authors’ knowledge, it is the first time that a metaheuristic algorithm is applied to the single batch-scheduling problem with job rejection. The following steps describe the details of our proposed algorithm.

6.1. Solution representation

One of the main challenging issues in designing a metaheuristic algorithm is to find a way by which the solution to the problem in hand can be represented. In order to solve different combinatorial optimization problems by metaheuristics, it is necessary to use a problem-specific solution representation. Since the BPM scheduling problem can be considered as a grouping problem, we use a special encoding scheme called grouping representation. When dealing with a grouping problem, the grouping representation scheme not only has lower redundancy than typical types of representation such as number encoding, but also it manipulate the groups rather than isolated items and, as we know, in the grouping problems these are groups that are meaningful building blocks of the problem.

In general, the grouping encoding/representation is formed from two parts: an item part, which is an array of size $n$ – $n$ is the number of items and a group part, which keeps a permutation of $E$ group labels. The $k$th element of the item part array can take any of the $E$ group tags, and determines the group to which the $k$th element belongs. An example of a grouping of seven items and its corresponding group encoding is given in Figure 4.

Husseinzadeh Kashan et al. [14] and [16] used the group encoding to solve the single batch-scheduling problem without rejection. Since in the research problem, it is possible to reject the jobs, we consider a group for the rejected jobs. To do so, we assume that all the rejected jobs are inserted in the first group and the accepted jobs are inserted in the second batch onward. For example, in the sample grouping of Figure 4, jobs 3 and 4 have been rejected, and the other jobs have been accepted. It is possible to reject all the jobs; therefore, when creating the initial population, we suppose that the first batch has an unlimited capacity.

6.2. Measuring similarity of grouping structures

In order to reconstruct the original updating equations of LCA in such a manner that they become applicable to discrete space of the research problem and work with groups instead of scalars, the key idea is to use some appropriate operators instead of the mathematical operators. Particularly, we substitute the mathematical minus operator (“−”), with a dissimilarity measure. Dissimilarity measure of two sets calculates the distance between them. Therefore, in this paper, we use “distance” to refer to such an operator.

There are different kinds of distance measures for grouping structures that can be used in this problem, such as Kulczynski’s coefficient, Sorensen–Dices’ coefficient, Tversky’s coefficient, or Jaccard’s coefficient of
Figure 3. Flowchart of the LCA.
dissimilarity. However, the most widely used distance measure, to calculate the degree of dissimilarity between groups \( G \) and \( G' \) with cardinality \( |G| \) and \( |G'| \), is the Jaccard’s distance, which is expressed as follows:

\[
\text{Jaccard Dissimilarity } (G, G') = 1 - \frac{|G \cap G'|}{|G \cup G'|}. \tag{13}
\]

Jaccard’s distance finds the degree of mismatch between two groups and varies from zero (identical groups) to one (disjoint groups). Unlike the Kulczynski, Sorensen–Dice, and Tversky coefficients, triangle inequality is fulfilled for Jaccard’s coefficient. Consequently, Jaccard’s distance is a metric. We can apply any of the mentioned distance measures. We, however, apply only the Jaccard’s distance owing to its simplicity and most popularity.

Now we are ready to develop the updating equations of our algorithm:

### 6.3. Updating equations in the proposed algorithm

As mentioned before, equations (9)–(12) in LCA has been designed to work in continuous space and cannot work with groups of items. Consequently, to deal with grouping problems we should modify equations (9)–(12) in such a manner that they work with groups rather than items. To do so, some appropriate operators must be used instead of the mathematical operators used in equations (9)–(12). Particularly, we substitute “−” operator with appropriate measures introduced in the previous subsection. First, we reshape the original updating equations of LCA as follows:

\[
x_{id}^{t+1} - b_{id}^t = y_{id}^t (c_1 r_1 (b_{id}^t - b_{kd}^t) + c_1 r_2 (b_{id}^t - b_{jd}^t)) \quad \forall d = 1, \ldots, n \tag{14}
\]

\[
x_{id}^{t+1} - b_{id}^t = y_{id}^t (c_2 r_1 (b_{id}^t - b_{id}^t) + c_1 r_2 (b_{id}^t - b_{id}^t)) \quad \forall d = 1, \ldots, n \tag{15}
\]

\[
x_{id}^{t+1} - b_{id}^t = y_{id}^t (c_1 r_2 (b_{id}^t - b_{kd}^t) + c_2 r_1 (b_{id}^t - b_{jd}^t)) \quad \forall d = 1, \ldots, n \tag{16}
\]

\[
x_{id}^{t+1} - b_{id}^t = y_{id}^t (c_2 r_2 (b_{id}^t - b_{kd}^t) + c_2 r_1 (b_{id}^t - b_{jd}^t)) \quad \forall d = 1, \ldots, n \tag{17}
\]

To construct applicable updating equations, suppose a situation in which both teams \( i \) and \( l \) won their last week’s matches, and we are dealing with the comparison of the winning team, \( i \), and the losing teams, \( k \) and \( j \), in the right-hand side of the related updating equation. In such a condition, our aim is to escape from the teams \( k \) and \( j \), and also approach team \( i \). We want to generate the new solution in such a way that the distance between the new solution \( x_{id}^{t+1} \) and the best obtained solution \( b_{id}^t \) becomes approximately equal to the resultant of the distance between \( i \) and \( k \) and the distance between \( i \) and \( j \). In other words, the newly generated solution must have the elements of \( b_{id}^t \) that are not in \( b_{kd}^t \) and \( b_{jd}^t \). Therefore, we substitute “−” by “Distance” in \( b_{id}^t - b_{jd}^t \) and \( b_{id}^t - b_{kd}^t \). Moreover, it is obvious that \( b_{jd}^t - b_{id}^t = -b_{id}^t + b_{jd}^t \) (and also \( b_{kd}^t - b_{id}^t = -(b_{id}^t - b_{kd}^t) \)) and now by substitution of “Distance” for “−” we can write them as \(-\text{Distance}(b_{id}^t, b_{jd}^t)\) (and also \(-\text{Distance}(b_{id}^t, b_{kd}^t)\)). Since the inverse of the distance measure operator is the similarity measure operator, we can use “similarity” instead of “−Distance”. Substitution of these suitable operators for subtraction operator (“−”) leads to the following updating equations that work with groups rather than isolated items.

\[
\text{Distance } (x_{id}^{t+1}, b_{id}^t) \approx y_{id}^t (c_2 r_2 \text{ Distance } (b_{id}^t, b_{kd}^t) + c_2 r_1 \text{ Distance } (b_{id}^t, b_{jd}^t)) \quad \forall d = 1, \ldots, n \tag{18}
\]
where \( d = 1, \ldots, D_x \) denotes the group indices and \( i = 1, \ldots, N_P \) (number of population). Note that, in equations (18)–(21) \( b_{id}^t, b_{jd}^t \) and \( b_{id}^t \) are groups of items while in equations (9)–(12) these are real scalars. Now we are ready to create a new solution.

### 6.4. Creation of new solution

After reconstructing the updating equations of the original algorithm in such a way that they become appropriate for the research problem, we can produce a new set of solutions. To do so, first, we calculate the right-hand side of the corresponding updating equation, which is one of the equations (18)–(21). In these equations, the symbol “\( \approx \)” implies “almost equal”. We choose this symbol rather than “=” as it may be impossible to form the new solution \( x_{id}^{t+1} \), in such a manner that the value of “distance” between the new solution and \( b_{id}^t \) becomes exactly equal to the value of the right-hand side of the equation. By taking \( R \) equal to the value obtained from the right-hand side of the updating equation, we have:

\[
\text{Distance} \ (x_{id}^{t+1}, b_{id}^t) \approx R
\]  
(22)

As we know \( \text{Distance}(...) \leq 1 \). Consequently, when the value of \( R \) is greater than one, we set \( R \) equal to one.

Now, to produce a new solution, we need to take two steps. First, we must make a decision about those parts of the parent which the offspring should inherit. We call this step as the **inheritance phase**. During this step, a number of items may be absent in the offspring solution. For this reason, in the second step, missing items are inserted back into an existing or a new group. We nominate this step as the **post-assignment** or **reinsertion phase**.

In our algorithm, the inheritance phase is governed by equation (22). This equation indicates that the creation of the new group \( x_{id}^{t+1} \), throughout the inheritance phase, should be in such a manner that its distance with \( b_{id}^t \) be almost equal to \( R \). In other words, if \( n_{id}^{t+1} = |x_{id}^{t+1} \cap b_{id}^t| \) stands for the number of common items between groups \( b_{id}^t \) and \( x_{id}^{t+1} \), we want to find \( n_{id}^{t+1} \) in such a manner that the value of Distance \( (x_{id}^{t+1}, b_{id}^t) \) gets close to the \( R \). In fact, the common items between \( b_{id}^t \) and \( x_{id}^{t+1} \) are the items that the offspring solution \( x_{id}^{t+1} \) inherits from the parent solution \( b_{id}^t \). As \( x_{id}^{t+1} \) can inherit up to all of the items in \( b_{id}^t \), throughout the inheritance phase it is acceptable to assume that \( x_{id}^{t+1} \subseteq b_{id}^t \). Given the value of \( R \) and starting from (22) we have:

\[
\text{Distance} \ (x_{id}^{t+1}, b_{id}^t) = \text{Jaccard’s coefficient of dissimilarity} \ (x_{id}^{t+1}, b_{id}^t)
\]

\[
= 1 - \frac{n_{id}^{t+1}}{|b_{id}^t|} \approx R \Rightarrow n_{id}^{t+1} = (1 - R)|b_{id}^t|
\]  
(23)

\( n_{id}^{t+1} \) denotes the number of shared items between \( b_{id}^t \) and \( x_{id}^{t+1} \); therefore, it must be an integer number. To get an integer number for \( n_{id}^{t+1} \), we round its value to the nearest integer. So we have:

\[
n_{id}^{t+1} = \text{round}[(1 - R)|b_{id}^t|]
\]  
(24)

where \( \text{round}[\cdot] \) denotes the rounding to the nearest integer.

As mentioned before, when creating the initial population, we suppose that the first batch has an unlimited capacity for the rejected jobs. At the end of the inheritance phase, first, the rejection capacity must be modified. We want the algorithm to properly explore the search space during the initial iterations and exploit the promising regions afterward. Therefore, to determine the rejection capacity in the first iteration, we generate a random number in an interval between a number smaller than the minimum rejection cost of the population and a number greater than the maximum rejection cost of the population. This issue is depicted in Figure 5. The
The interval from which rejection capacity is selected in first iteration.

Figure 5. The interval from which rejection capacity is selected in first iteration.

interval is slightly larger than that of between minimum and maximum of the rejection costs. This helps the algorithm to explore the entire search space and to avoid falling in local optimum. At first, the probability to produce the rejection capacity of the newly generated solution by the above-mentioned method is equal to one.

As the algorithm proceeds, we want the algorithm to gradually restrict its view, and exploit the near optimal solution region. To achieve this goal, another way in which the rejection capacity approaches the rejection capacity of the best member of the population is used to produce the rejection capacity of the newly generated solution. In this method the rejection capacity is obtained by equation (25). At first, the probability of producing new rejection capacity from equation (25) is equal to zero, and as the algorithm proceeds, the probability of using the first method is reduced with a constant rate (FS) and that of the second method is increased such that the sum of probabilities is equal to one.

\[
\text{new rejection capacity} = \text{current rejection capacity} + \alpha (\text{rejection cost of the best solution-current rejection cost})
\]  
where \(\alpha\) is a number between zero and one.

As we are still in inheritance phase, after determining the rejection capacity, if the rejection cost of the rejected jobs is greater than the rejection capacity, currently rejected jobs are missed one by one until the rejection cost become close enough to the rejection capacity. Else, if the rejection cost of the rejected jobs is lower than the rejection capacity, currently missed jobs are rejected one by one until the rejection cost become close enough to the rejection capacity. Now, the operation on the first group, which is the group of the rejected jobs, is completed.

Before providing more details on the generation of the new solution \(x_{i+1}^{t+1}\), it is important to have a brief discussion about the way whereby the groups ordered. In a typical solution to a batch-scheduling problem, the ordering of groups has no impact on the objective value. However, when computing the value of \(R\), it is reasonable to properly join the groups in pairs, and then calculate the value of “distance” between each pair of the groups. Because, when the groups of the solution \(b_{id}^{t}\) are paired with the groups of the solution \(b_{jd}^{t}\) and \(b_{kd}^{t}\) in an arbitrary manner, distances may take large values and cause the algorithm shows a divergent behavior. For instance, if we compare two identical solutions that their groups are ordered in an arbitrary manner, we may obtain biased results for distances, which cannot represent the real distance between them. Therefore, we re-index the groups of \(b_{id}^{t}\) and \(b_{jd}^{t}\) (and \(b_{kd}^{t}\)) in such a manner that the most similar groups become paired together. To do so, we use the concept of “maximum weight bipartite matching”, and propose a general ordering procedure. To propose the maximum weight bipartite matching (MWM) ordering rule, first, we should construct a complete bipartite similarity graph – a graph whose nodes can be partitioned into two subsets (source and destination) such that each node in the first subset is adjacent to every node in the second subset and there is no edge between the nodes of the same subset. Each source node is related to one group in \(b_{id}^{t}\) and each destination node is related to one group in \(b_{jd}^{t}\) (and \(b_{kd}^{t}\)). Each edge is weighted by the value of Jaccard’s coefficient of similarity between its two end nodes. If \(Y_b\) represents such a graph, a matching in \(Y_b\) is a subset of edges without common nodes. Actually, our aim is to first find a matching with the greatest sum of weights (maximum weight matching), and then re-index the groups of \(b_{jd}^{t}\) (and \(b_{kd}^{t}\)) in such a manner that both end
nodes of an edge which is in the maximum weight matching, have the same index. Husseinzadeh Kashan et al. [14] proved that this procedure can be done in $O(n)$ and can comply with our goal. We use this procedure for all the groups of the accepted jobs.

Finally, based on any logic, the missing items must be assigned to an existing or a new group in such a way that the feasibility of the generated grouping is guaranteed. A good strategy to reassign orphaned jobs is to use the best-fit decreasing strategy. In this strategy, among the unassigned jobs, first we pick the one with the maximum processing time. We then place the selected job in a feasible batch whose batch processing time is larger than the processing time of the chosen job and also has the smallest residual capacity. Note that if the batch processing time of all the feasible batches are individually smaller than the processing time of the chosen job, place the job in a batch which has the maximum batch processing time among the feasible batches. If more than one feasible batch has the greatest batch processing time, select the one with the smallest residual capacity. Finally, if the chosen job fits in no existing batches, generate a new batch and place the job in it.

The general framework of the proposed algorithm is described as Algorithm 1:

---

**Algorithm 1.** General framework of the proposed algorithm.

1. **Initialize:** the number of seasons ($NS$), the number of teams ($NT$), $c_1$, $c_2$ and $FS$; $t = 1$;
2. Generate the league schedule;
3. **For** $i = 1$ to $NT$
   3.1. Generate a random feasible solution $x^t_i$;
   3.2. Evaluate the generated solution;
   3.3. $b^t_i \leftarrow x^t_i$
4. **End for**
5. **While** $t \leq NS \ast (NT - 1)$
   5.1. Use the playing strength and determine the winner and loser of each match, based on the league schedule at week $t$;
   5.2. $t = t + 1$;
   5.3. **For** $i = 1$ to $NT$
      5.3.1. Based on the events of the previous week determine the appropriate updating equation;
      5.3.2. Except for the first group (group of the rejected jobs), Using the MWM ordering rule, pair each group of the solution $b^t_{id}$ with the most similar group in $b^t_{jd}$ (and $b^t_{kd}$);
      5.3.3. For each group of the solution $b^t_{id}$, calculate the value of distances and similarities that are needed in the right-hand side of the related updating equation using (13);
      5.3.4. Obtain the value of $n^{t+1}_{id}$ by (24);
      5.3.5. Select $n^{t+1}_{id}$ items from group $b^t_{id}$ based on any logic and put them in the new group $x^{t+1}_{id}$
      5.3.6. Modify the rejection capacity;
      5.3.7. Determine the rejected jobs;
      5.3.8. Based on any logic, insert the missing items in an existing or a new group;
      5.3.9. Evaluate the playing strength of the newly generated formation ($x^{t+1}_{id}$);
      5.3.10. **If** $f(x^{t+1}_{id}) \leq b^t_{id}$
            5.3.10.1. $b^{t+1}_{id} \leftarrow x^{t+1}_{id}$;
      5.3.11. **Else**
            5.3.11.1. $b^{t+1}_{id} \leftarrow b^t_{id}$;
      5.3.12. **End if**
   5.4. **End for**
5.5. **If** $mod(t, nt - 1) = 0$
   5.5.1. Generate the league schedule;
5.6. **End if**
6. **End while**

---

The time complexity of GLCA is linear in the size of population and the number of iterations. The effort to generate a new solution within the search process is also bounded by $O(n \log n)$, because a best-fit type algorithm
Table 1. Information of the jobs for the numerical example.

<table>
<thead>
<tr>
<th>Job number</th>
<th>Processing time</th>
<th>Size</th>
<th>Rejection penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29</td>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>41</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>42</td>
<td>10</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>49</td>
<td>7</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>18</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>43</td>
<td>6</td>
<td>18</td>
</tr>
</tbody>
</table>

is used to complete any partial solution. Therefore the time complexity of GLCA is $O(NT \times NI \times n \log n)$. Given that $NI = NS \times (NT - 1)$.

6.5. A numerical example

In this subsection, a numerical example is used to illustrate the process of new solution generation in the proposed algorithm. There are 8 jobs to be processed on an SBPM. The characteristics of the numerical problem are as the same as those of the research problem, presented in Section 3. Table 1 shows the processing time, size, and rejection penalty of each job. The capacity of the batches of the accepted jobs is equal to 18.

Suppose that we want to generate a new solution $x_{td}^{t+1}$ from the solution $b_{td}^t$. The indices $j$, $k$, and $l$ have the same meaning as in Section 5.3. To calculate the distances and similarities, suppose that for all groups $c_1 = c_2 = 0.6$, $r_1 = 0.7$ and $r_2 = 0.4$. We also assume that in the last week, $i$ was winner and $l$ was loser. Therefore, we have to use equation (19). First, we re-index the groups of $b_{jd}^t$ (and $b_{kd}^t$) by MWM ordering rule. The re-indexing process is illustrated in Figure 6.

Then, in the inheritance phase, we must make a decision about those parts of $b_{td}^t$ which $x_{td}^{t+1}$ should inherit. Figure 7 demonstrates the process of the inheritance phase of the numerical example.

Finally, suppose that the minimum and the maximum value of the rejection costs in the population are 30 and 90, respectively. We want to calculate the rejection capacity using the first method described in Section 6.4. We generate a random number in an interval which is 10% wider than the interval between the minimum and the maximum value of the rejection costs (5% in each end). Therefore, to determine the rejection capacity, we generate a random number in an interval between 27 and 93. Assume that the random number generation created a value of 50 for the rejection capacity of $x_{td}^{t+1}$. According to Table 1, the current rejection penalty is equal to $17 + 20 = 37$. The process of post-assignment is illustrated in Figure 8.

7. Computational experiments

To assess the performance of the developed LCA (GLCA), a series of experiments are carried out. For the small size instances, the performance of GLCA is compared with GAMS. For the large scale instances, as there is no metaheuristic algorithm for the single batch-scheduling problem with job rejection, a genetic algorithm is proposed based on the algorithm presented by Husseinzadeh Kashan et al. [13], and the computational results of the proposed LCA are compared with those of the genetic algorithm.

This section is presented in five subsequent sections. In the first section, the characteristics of the modified GA are represented. In the second section, the specification of the randomly generated test instances is described. In the third section, Parameter tuning of the proposed algorithm is discussed. The quality of the proposed LB is shown in the next section, and finally, the results of the computational experiments are presented in the last section.
Both algorithms are coded in MATLAB 2017b. The CPLEX solver in GAMS v.24.1.2 is used to solve the mathematical programming model of the research problem. A Core i7, 3.6 GHz computer with 8 GB RAM is used to run all the computational experiments.

### 7.1. Modified batch-based hybrid genetic algorithm (MBHGA)

In this section, we adopt batch-based hybrid genetic algorithm (BHGA) introduced by Husseinzadeh Kashan et al. [13], for the research problem. In the sequel, a discussion about the chromosome representation and the operators of the genetic algorithm is presented.

Similar to GLCA, the time complexity of MBHGA algorithm is linear in the population size and the number of iterations. The effort to generate a new solution within the search process is bounded by $O(n \log n)$, because a best-fit type algorithm is used to complete any partial solution. Therefore the time complexity of MBHGA is $O(Popsize \times NI \times n \log n)$. Given that $NI$ is the number of iterations and $Popsize$ is the size of the population.
Figure 7. The inheritance phase for the numerical example.

<table>
<thead>
<tr>
<th>$x_{id}^{t+1}$</th>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>job 2, job 7</td>
<td>job 3</td>
<td>job 5</td>
<td></td>
</tr>
</tbody>
</table>

Distance($b_{k1}^t$, $b_{i1}^t$) = 0.6  
Distance($b_{k2}^t$, $b_{i2}^t$) = 0.5  
Distance($b_{k3}^t$, $b_{i3}^t$) = 0.66  
Similarity($b_{k1}^t$, $b_{j1}^t$) = 0  
Similarity($b_{k2}^t$, $b_{j2}^t$) = 0.33  
Similarity($b_{k3}^t$, $b_{j3}^t$) = 0.25

Calculate the right-hand sides  
$R_{i1}^t = 0.25$  
$R_{i2}^t = 0.29$  
$R_{i3}^t = 0.34$

Calculate the number of inherited items for each group

$n_{i1}^t = 2$  
$n_{i2}^t = 1$  
$n_{i3}^t = 1$

Figure 8. The post-assignment phase of the numerical example.

<table>
<thead>
<tr>
<th>$x_{id}^{t+1}$</th>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>job 2, job 7, job 6, job 7</td>
<td>job 3</td>
<td>job 5</td>
<td></td>
</tr>
</tbody>
</table>

Missing items: 1, 4, 6  
Rejection cost= 37

Reject the missed items one by one until the rejection cost become close enough to the rejection capacity.

<table>
<thead>
<tr>
<th>$x_{id}^{t+1}$</th>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>job 2, job 6, job 7</td>
<td>job 1, job 3</td>
<td>job 4, job 5</td>
<td></td>
</tr>
</tbody>
</table>

Missing items: 1, 4  
Rejection cost= 50

Accept the missing items by assigning them to an existing or a new group.
7.1.1. Chromosome representation

In the representation used by Husseinzadeh Kashan et al. [13] chromosome of BHGA, each gene belongs to one of the $n$ jobs. Each gene indicates the batch (group) to which the job is assigned. We assume that the jobs in the first batch, are the rejected ones. Figure 9 shows an instance chromosome related to a batch-scheduling problem with rejection including 6 jobs. It is possible to reject all the jobs; therefore, when creating the initial population, we suppose that the first batch has an unlimited capacity (as the same as what was mentioned for the proposed LCA in Sect. 6.1).

7.1.2. Selection

As in BHGA, the roulette wheel selection technique is used for selecting parents. The fitness of each parent is $R_{\text{tot}} + C_{\text{max}}$. The probability of selecting a certain parent is proportional to its fitness.

7.1.3. Crossover

As in BHGA, the parameterized uniform crossover suggested by Bean [4] is applied as the crossover strategy. The method involves using a biased coin for each gene to determine which parent passes its gene to the relevant offspring solution. Using parameterized uniform crossover allows the algorithm to direct the search more quickly towards the components of better solutions, which accelerates the convergence [34]. Figure 10 shows the crossover procedure of BHGA. After applying the parameterized uniform crossover, it may occur to lose some of the batches in the newly generated offspring. As it is shown in Figure 10, in such a case we rename the batches in the offspring. Now we have an initial plan for the batching that can be infeasible. In order to ensure the feasibility we use the following MRBP to reach a feasible offspring.

7.1.3.1. MRBP In this subsection, we modify the random batch procedure (RBP), which is suggested by Husseinzadeh Kashan et al. [13] to get a feasible solution, in such a way that it becomes applicable to the batch-scheduling problem with job rejection. The modified random batch procedure (MRBP) is as follows:

Step 1. Similar to what was done for the proposed LCA, to determine the rejection capacity, we generate a random number in an interval between a number smaller than the minimum rejection cost of the population and a number greater than the maximum rejection cost of the population. This issue is depicted in Figure 5. After determining the rejection capacity, if the current rejection cost of the offspring is greater than the rejection capacity, the rejected jobs are inserted one by one in the other groups until the rejection cost become close enough to the rejection capacity. Else, if the current rejection cost of the offspring is lower than the rejection capacity, we reject some jobs from the batches with capacity violation, until the rejection
cost become close enough to the rejection capacity. Now, we set the capacity of the first group in such a way that the capacity violation of the first group becomes equal to zero. Now, the operation on the first group, which is the group of the rejected jobs, is completed.

Step 2. Among the batches with the capacity violation, select the batch with highest processing time, and among the jobs within the selected batch, choose ones with the longest processing time. Among the batches that are feasible for the selected job, select the first one that has the smallest residual capacity, and its processing time is greater than the processing time of the selected job. Insert the selected job into the selected batch, then go to Step 3. If the processing time of all the feasible batches is smaller than the processing time of the selected job, insert the selected job in the feasible batch with the largest processing time and then go to Step 3. Finally, if the selected job cannot be fitted in any of the existing batches, generate a new batch and insert the job in it.

Step 3. Do the second step while getting a feasible batching scheme.

7.1.4. Mutation

The random swapping mutation, which randomly changes the positions of two genes of a selected chromosome, is applied as the mutation operator for the MBHGA. Figure 11 illustrates the mutation procedure of MBHGA, which is the same as the BHGA. The feasibility of the mutated solution is kept through using the MRBT.

7.2. Test problem specification

The testing instances are randomly generated in a manner similar to that used by Ghazvini and Dupont [8]. There are four categories of problems, which are differed by job sizes. The first category, denoted by $S_1$, has a combination of small and large jobs, randomly generated from a discrete uniform distribution over the range of [1, 10]. The second one, denoted by $S_2$, has no small and large jobs, randomly generated from a discrete uniform distribution over the range of [3, 8]. The third one, denoted by $S_3$, only has large jobs, randomly generated from a discrete uniform distribution over the range of [4, 10]. Finally, the fourth one, denoted by $S_4$, only has small jobs, randomly generated from a discrete uniform distribution over the range of [1, 5]. For each category, six levels for the number of jobs are considered, i.e., $n \in \{10, 20, 50, 100, 150, 200\}$, respectively designated by $J_1$–$J_6$. The processing time ($p_j$) of each job is generated from a discrete uniform distribution between 10 and 50. The rejection penalty of each job ($e_j$) is drawn from a discrete uniform distribution in the interval [1, 20]. The capacity of the machine ($B$) is set equal to 10 units. The experimental settings are represented in Table 2. Based on Table 2, there are 24 combinations of the problem categories and number of jobs. For each combination, we randomly generate 10 instances and report the average results.

7.3. Parameter tuning

Through a number of experiments, it is found that different parameter setting affects the performance of both algorithms. For example, the number of teams is one of the key parameters in GLCA. Broadly speaking, although more teams can lead to a better solution, it can increase the computational costs. Therefore, for both algorithms, it is essential to set proper values for the parameters. The parameters of GLCA are the number of seasons ($NS$), the number of teams ($NT$), $c_1$, $c_2$, and the constant rate for determining the method by which the rejection capacity is produced ($FS$). As mentioned before, two of the most crucial mechanisms of any metaheuristic algorithm are intensification and diversification, and making a proper balance between these
Table 2. Factors and levels of problem.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>J1 = 10, J2 = 0, J3 = 50, J4 = 100, J5 = 150, J6 = 200</td>
</tr>
<tr>
<td>s_j</td>
<td>S1 = DU [1, 10], S2 = DU [3, 8], S3 = DU [4, 10], S4 = DU [1, 5]</td>
</tr>
<tr>
<td>p_j</td>
<td>DU [10, 50]</td>
</tr>
<tr>
<td>e_j</td>
<td>DU [1, 20]</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3. Evaluation of the lower bound.

<table>
<thead>
<tr>
<th>Number of jobs</th>
<th>Category</th>
<th>Objective function value (C*)</th>
<th>Lower bound (LB)</th>
<th>LB/C*</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>S1</td>
<td>116</td>
<td>116</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>S2</td>
<td>125</td>
<td>125</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>S3</td>
<td>125</td>
<td>125</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>S4</td>
<td>111</td>
<td>111</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>S1</td>
<td>212</td>
<td>210</td>
<td>0.9906</td>
</tr>
<tr>
<td></td>
<td>S2</td>
<td>188</td>
<td>180</td>
<td>0.9574</td>
</tr>
<tr>
<td></td>
<td>S3</td>
<td>219</td>
<td>219</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>S4</td>
<td>168</td>
<td>168</td>
<td>1</td>
</tr>
</tbody>
</table>

two parameters is very important. Diversification increases by increasing $c_1$ and $c_2$. Intensification, on the other hand, increases by decreasing $c_1$ and $c_2$. We want the algorithm to gradually reduce the diversification while, simultaneously increase the intensification. At first, $c_1 = c_2 = 1$. In each iteration, the value of $c_1$ and $c_2$ are reduced with a reduction rate ($RR$). $FS$, $NT$, and $NS$ are set to be 0.975, 5, and 70, respectively. The parameters of the modified BHGA, including the crossover rate ($P_c$) and the mutation rate ($P_m$), are set to be 0.9 and 0.1, respectively. Parameters of both algorithms are set based on the results of considerable experiments.

7.4. Evaluation of the lower bound

A tighter lower bound leads to a better performance assessment of the algorithms. Therefore, the quality of the proposed lower bound is important. In this section, a small computational experiment is conducted to compare the tightness of the proposed LB relative to optimal objective function values. We consider small instances with 10 and 20 jobs, and for each of them, a random instance in each category is generated. The value of the LB obtained from the proposed DPA is compared with the objective function value achieved by solving the mathematical model (presented in Sect. 3.2). The results are summarized in Table 3.

Column 1 in Table 3 presents the number of jobs. Column 2 shows the category of job sizes. Column 3 displays the optimal objective function value obtained from solving the mathematical model using GAMS software. Column 4 presents the value of the LB obtained from the proposed DPA. Column 5 reports the ratio of the LB to the optimal objective function value.

It can be seen from Table 3 that in all instances the ratio of the LB to the optimal objective function value lies between 0.9574 and 1 and in six out of eight instances the ratio is exactly equal to one. The results verify the tightness of the proposed lower bound.

7.5. Experimental result

In this section, the performance of GLCA is assessed by comparing its computational results with those of GAMS and MBHGA. As the research problem is NP-hard, the exact solution for medium and large-sized instances cannot be found in polynomial time. Thus, in the first experiment, small problems with 10 and 20 jobs
Table 4. Number of variables and constraints of the mathematical model.

<table>
<thead>
<tr>
<th>No. jobs</th>
<th>No. variables</th>
<th>No. constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>20</td>
<td>440</td>
<td>440</td>
</tr>
<tr>
<td>(n)</td>
<td>(n^2 + 2n)</td>
<td>(n^2 + 2n)</td>
</tr>
</tbody>
</table>

Table 5. Comparison the result of proposed algorithm and GAMS for instances with 10 jobs.

<table>
<thead>
<tr>
<th>10 jobs</th>
<th>Proposed algorithm</th>
<th>GAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best cost</td>
<td>Ave. cost</td>
</tr>
<tr>
<td>S1</td>
<td>116</td>
<td>116</td>
</tr>
<tr>
<td>S2</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>S3</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>S4</td>
<td>111</td>
<td>111</td>
</tr>
</tbody>
</table>

Table 6. Comparison the result of proposed algorithm and GAMS for instances with 20 jobs.

<table>
<thead>
<tr>
<th>20 jobs</th>
<th>Proposed algorithm</th>
<th>GAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best cost</td>
<td>Ave. cost</td>
</tr>
<tr>
<td>S1</td>
<td>212</td>
<td>212</td>
</tr>
<tr>
<td>S2</td>
<td>188</td>
<td>188</td>
</tr>
<tr>
<td>S3</td>
<td>219</td>
<td>219</td>
</tr>
<tr>
<td>S4</td>
<td>168</td>
<td>168</td>
</tr>
</tbody>
</table>

are considered, and for each of them, a random instance in each category is generated. For each of the generated instances, the mathematical model (presented in Sect. 3.2) is solved with GAMS software. The number of variables and constraints in the mixed integer model for 10 and 20 jobs are presented in Table 4.

The proposed algorithm is not a deterministic method; therefore, it is more accurate to run it several times instead of once. For each instance, GLCA is run ten times independently. For the small-sized instances, the best, worst, and average results are reported. The results obtained from GLCA and GAMS for 10 and 20 jobs are shown in Tables 5 and 6. In these tables, the first column shows the category of job sizes. The best, average, and the worst cost function value among the ten runs of GLCA is reported in the second to fourth columns of the tables, respectively. The fifth column gives the average run time taken by GLCA. The sixth column reports the objective function value, obtained from solving mathematical programming model by GAMS. The seventh column is the optimality gap, and the eighth column is the CPU time of GAMS.

The computational results show that GLCA finds the optimal solution in small instances with 10 and 20 jobs. It is also clear from Tables 5 to 6 that the proposed algorithm finds the solution in a time less than the computational time taken by GAMS.

In the second experiment, in order to assess the performance of GLCA in solving the medium and large scale problems, its performance is compared with MBHGA (which presented in detail in Sect. 7.1). To achieve this aim, for each instance, GLCA is run ten times independently with the parameters specified in Section 7.3. In each experiment, the number of function evaluation in which the algorithm converged is considered, and finally, the average of the number of function evaluations in each instance is recorded. Then for each instance, MBHGA is run ten times. The number of objective function evaluation of the proposed algorithm is used as the stopping criteria of the MBHGA. In each instance, the number of generations (\(NG\)) and the size of population (\(npop\)),...
are set in such a way that the number of objective function evaluation of MBHGA becomes almost equal to the number of function evaluation of GLCA. Now, we can compare the solution obtained by each algorithm in the same number of objective function evaluation. The computational results obtained from the proposed algorithm and MBHGA for the problem instances with 50, 100, 150, and 200 jobs are summarized in Tables 7–10, respectively. In these tables, the first column displays the number of jobs. Category of job sizes is shown in the second column. The calculated lower bound is given in the third column. The fourth column denotes the average number of objective function evaluation of both algorithms. Columns 5 and 7 show the best and worst objective function value among the 10 runs of GLCA. Column 6 shows the average objective function value.

As can be seen from Tables 7 to 10, almost in all categories of the problem, the overall performance of GLCA is better than that of MBHGA, which demonstrates the performance of GLCA. In all instances of S1, S2, and
Figure 12. The convergence behavior of the algorithms for a problem instance in category $S_1$ with 200 jobs.

Figure 13. The convergence behavior of the algorithms for a problem instance in category $S_2$ with 200 jobs.

$S_3$, GLCA is superior to MBHGA. In the fourth category ($S_4$), when the number of jobs is equal to 50 and 100, MBHGA has a better performance than the proposed algorithm. As the number of jobs increases, the average performance of GLCA improves and for the instances with 150 and 200 jobs, GLCA is superior to MBHGA.

Moreover, experimental results indicate that GLCA truly converges toward the global optimum. For large-scale test problems with 200 jobs, the gap between the best-found value by GLCA and the lower bound value is less than 6%, 5%, 3%, and 14%, for problem categories $S_1$, $S_2$, $S_3$, and $S_4$, respectively. Recall that these are gap values relative to the lower bound values and not the true optimal values. Hence, the true gap with the optimal value is even smaller.

In conclusion, experimental results exhibit that the convergence of GLCA is better than that of a powerful existing MBHGA algorithm taken from the literature. We have depicted the evolution/convergence plot of GLCA
Figure 14. The convergence behavior of the algorithms for a problem instance in category $S_3$ with 200 jobs.

Figure 15. The convergence behavior of the algorithms for a problem instance in category $S_4$ with 200 jobs.

and MBHGA for the large-scale problems with 200 jobs in Figures 12–15. As can be seen, both algorithms are converging. However, GLCA exhibits a more decent property.

The comparison of the average performance of GLCA and MBHGA for different categories of job sizes as well as the number of jobs are illustrated in Figures 16–19. The results demonstrate that almost in all instances, the average performance of GLCA is better than that of MBHGA.
8. Conclusion and future research

In this paper, we have discussed the scheduling problem of a set of non-identical size jobs on a single parallel-BPM wherein the job rejection is allowed, $1|\text{p-batch, rej, } \sum s_j \leq B|C_{\text{max}} + R_{\text{tot}}$. First, the research problem was modeled as a 0-1 mixed integer linear programming (0–1 MILP) model. Then, in order to solve the problem, since the single batch-scheduling problem with rejection can be considered as a grouping problem, and it was also shown to be a NP-hard problem, a grouping metaheuristic algorithm based on the league championship algorithm (LCA) was developed to exploit the structural information of the problem. Adopting the specific
grouping representation, the grouping version of LCA (GLCA) was developed. We developed updating equations similar to the original LCA equations which use the Jaccard coefficient of dissimilarity instead of the arithmetic subtraction operator. By doing so we can use the groups rather than the isolated items, and as we know, in the grouping problems these are groups that are meaningful building blocks of the problem. The new updating equations work in continuous space while their consequences are used in discrete space of a grouping problem. To demonstrate the effectiveness of the proposed GLCA, plenty of computational experiments have been done using randomly generated test instances. For the small-sized instances, the performance of GLCA was compared with GAMS. For the large-sized instances, since there was no metaheuristic algorithm in the literature for the problem on hand, a genetic algorithm based on the approach proposed in Husseinzadeh Kashan et al. \cite{13} was adapted for the research problem, and the computational results of GLCA were compared with those of genetic

Figure 18. Comparison of the average performance of the algorithms for the problem instances in category $S3$.

Figure 19. Comparison of the average performance of the algorithms for the problem instances in category $S4$. 

- Figure 18.

- Figure 19.
algorithm. The results of the experiments verified the effectiveness of GLCA. For future research, other new metaheuristic algorithms such as Find-Fix-Finish-Exploit-Analyze algorithm [18] can be modified as was done in this research, or more complex manufacturing environments, such as flexible manufacturing [15] or the presence of parallel or flow shops of batch processing machines [12] can be considered. Moreover, evaluation of GLCA on other grouping problems such as vehicle routing and scheduling [1] can be an interesting research direction.

REFERENCES


Please help to maintain this journal in open access!

This journal is currently published in open access under the Subscribe to Open model (S2O). We are thankful to our subscribers and supporters for making it possible to publish this journal in open access in the current year, free of charge for authors and readers.

Check with your library that it subscribes to the journal, or consider making a personal donation to the S2O programme by contacting subscribers@edpsciences.org.

More information, including a list of supporters and financial transparency reports, is available at https://edpsciences.org/en/subscribe-to-open-s2o.