A NEW APPROACH FOR PERMUTATION FLOW-SHOP SCHEDULING PROBLEM USING LEAGUE CHAMPIONSHIP ALGORITHM

Seyed Mojtaba Sajadi\textsuperscript{a}, Ali Husseinzadeh Kashan\textsuperscript{b}, and Siyavash Khaledan\textsuperscript{c}
\textsuperscript{a} University of Tehran, Tehran, Iran. 
\texttt{msajadi@ut.ac.ir}
\textsuperscript{b} Department of Industrial Engineering, Tarbiat Modares University, Tehran, Iran. 
\texttt{a.kashan@modares.ac.ir}
\textsuperscript{c} Department of Industrial Engineering, Islamic Azad University, Najafabad Branch, Isfahan, Iran. 
\texttt{siyavash.khaledan@gmail.com}

ABSTRACT

This paper presents a new algorithm, called league championship algorithm (LCA), for scheduling context. We consider the scheduling in a permutation flowshop system with makespan criterion. LCA is one of the latest algorithms introduced for optimization which tries to metaphorically model a league championship environment wherein artificial teams play in an artificial league for several weeks (iterations). Given the league schedule, a number of individuals, as sport teams, play in pairs and their game outcome is determined given known the playing strength (fitness value) along with the team formation (solution). Modelling an artificial match analysis, each team devises the required changes in its formation (a new solution) for the next week contest and the championship goes for a number of seasons. LCA works for continuous optimization and should be modified to work for discrete scheduling problems. For this sake, we allow LCA searches within the continuous space, but do evaluations in a discrete space via a heuristic rule to make a bridge between the continuous and discrete spaces. Results of LCA applied to well known benchmark suites are presented and compared to the well known approaches such as genetic algorithm, particle swarm optimization and differential evolution algorithms. On the adopted benchmark suite, LCA is able to beat all these rivals.

Keywords: Scheduling, flow shop system, metaheuristics, league championship algorithm (LCA)

* Corresponding Author: a.kashan@modares.ac.ir
Tel: +98912 6971214
1 INTRODUCTION

A production design in a manufacturing system is to put the machinery in a serial position, in a way that each one operates a single task in certain duration, so that the product exits from the system when it passes a series of freed machines after probable waits. This is called a flow-shop system. Permutation Flow-Shop Problem (PFSP) is a specified type of Flow-Shop Problem which determines the optimal permuting sequencing program of operations on \( n \) tasks by \( M \) machines. There are two assumptions in a permutation flow-shop sequencing problem; (i) to assign the processing of \( n \) jobs through \( m \) machines with an order on different machines being the same for all jobs, and (ii) fixed known processing time for each job on each machine is available [1]. A large body of literature has presented many algorithms of combinatorial optimization using classical methods of solving discrete problems and implicit counting methods like Branch and Bound (B&B), which are of a high level of computational complexity while guarantee the optimality in small-sized ones. Approximation methods are chosen in order to remove the defect of high level of complexity; meanwhile they do not necessarily guarantee the optimality.

Following the same pattern, numerous methods of solving scheduling problems in permutation flow-shop systems have been presented since the time of introducing the problem of scheduling in the flow-shop system [2]. Exact methods like B&B and mathematical programming methods [3] are applied only to small-sized problems due to the high level of computational complexity of large-sized ones. Hence, some heuristic methods to generate solutions and some metaheuristics to improve solutions, being titled as neighbourhood/local search Methods, have been developed [4]. Among these local search methods, we can point out to Simulated Annealing (SA) [5], Tabu Search (TS) [6,7], and Variable Neighbourhood Search (VNS) [8]. A hybrid method by Zheng and Wang [9] has put a Simulated-Annealing-based sub-algorithm as the mutation operator in Genetic Algorithm for several crossover operations on sub-populations. Metaheuristics such as Particle Swarm Optimization (PSO) or Differential Evolution (DE) algorithms or another hybrid algorithm, combining DE with a local search procedure based on tasks pairs exchanging [10], have been applied for flow-shop problem. A dramatic sketch to deal with discrete variables led to present a DE-based algorithm for the flow-shop problem, too [11]. Another hybrid model in accordance with DE, with blocking assumption, was presented to solve Permutation Flow-shop Problem [12]. A PSO algorithm was alleged for the total weight of completing durations minimization problem in a permutation flow-shop system [13]. A meta-heuristic approach called scatter search (SS) has also been employed for an NP-hard sequencing problem, which is used to find a processing order of \( n \) different jobs to be processed on \( M \) machines in the same sequence with minimizing the makespan [14]. Another hybrid approach was proposed by Lin and Ying [15], which draws on the advantages of simulated annealing and tabu search for the non-permutation flow-shop scheduling problem, in which the objective function is the minimization of the makespan. Débora et al. [16] examined the flow-shop scheduling problem with no storage constraints and with blocking in-process. In that environment, there were no buffers between successive machines; therefore, intermediate queues of jobs waiting in the system for their next operations were not allowed. Flow shop scheduling problems with sequence-dependent setup times and minimizing the number of tardy jobs has been considered in another research, in which a mixed-integer linear programming model is developed for the problem [17]. A multi-objective flexible flow shop scheduling problem with limited intermediate buffers whose objectives are consisted of minimizing the completion time of jobs and minimizing the total tardiness of jobs is also available; then a hybrid water flow algorithm for solving the problem is proposed [18].

The league championship algorithm (LCA) is a novel metaheuristic, designed based on the metaphor of sport competitions in leagues [19,20,21]. Besides the nature, culture, politics, human, etc. as the typical sources of inspiration behind various algorithms, the metaphor of sporting competitions is used, for the first time, in LCA. The methodology of LCA can be overviewed as follows. A number of individuals, making role as sport teams, compete in an
artificial league for several weeks (iterations). Based on the league schedule in each week, teams play in pairs and their game outcome is determined in terms of win or loss, given known their playing strengths (fitness values) along with the particular team formation/arrangement (solution) followed by each team. Keeping track of the previous week events, each team devises required changes in its formation (a new solution is generated) for the next week contest and the championship goes on for a number of seasons (stopping condition). The way, in which a new solution associated to a team is generated, is governed via imitating the match analysis process, followed by imaginary coaches, to design a suitable arrangement for their forthcoming match. In a typical match analysis, coaches will modify their arrangement on the basis of their own game experiences and their opponent’s style of playing.

LCA was, in its first utilization, used for finding the global minimum of unconstrained functions and later for constrained continuous ones [21]. Following these successful applications of LCA for continuous optimization, we are interested to investigate its effectiveness for discrete optimization, too. This paper proposes a new solution method for task sequencing and scheduling in a permutation flow-shop system, based on League Championship Algorithm (LCA). The performance of the new method is also investigated against well-established rivals.

The paper, first, shortly presents the fundamentals of LCA, and then, an LCA-based optimization algorithm is developed to solve the sequencing problem in a permutation flow-shop system. To find out how well LCA performs on the flow-shop scheduling problem, we compare it to benchmark suites from literature (GA, PSO and DE).

2 OVERVIEW OF LCA

LCA, a population-based metaheuristic, presented by Husseinzadeh Kashan [19,20], which offers an approximate method of global optimization in the continuous context, like all other population-based metaheuristics, rests on the assumption that a population of possible solutions should be moved to promising areas of the search space, in terms of the problem objective, in course of search for the optimum values. As a majority of population-based algorithms do, LCA deals with a set of $l$ (the population size) feasible solutions in the search space, which evolve gradually in a series of successive iterations. Let us, first, give a brief introduction to a number of related sport terminologies and their counterparts in the LCA.

- **Sport League:** A sport league is an organization to provide a regulated series of competitions for a number of individuals to compete in a team format specific sport. A championship league may be arranged in a way in which each team may play with every other teams for a number of times (legs) in a round-robin tournament and finally the team with the best record of scores, added up from wins/ties/losses resulting from its matches, will be recognized as the champion. (The algorithm ignores the tie results.)

- **Formation:** Each team normally has a style for playing which is recognizable via its formation. A team formation is a structure in which the players are distributed throughout the game field [21]. A coach tries to find and implement the best possible formation for his/her team based on his/her analyses of the previous matches.

- **Match Analysis:** The main aim of match analysis is to identify own strengths -to be further built upon, own weaknesses -to be improved- the opponent’s strengths (own threats) -to be defeated, and opponent’s weaknesses (own opportunities) -to be utilized. The feedbacks of such an analysis should be ready pre-match, post-match, or in the build-up to the next match [23]. Such an analysis is called strength (S)-weakness (W)-opportunity (O)-threat (T) analysis (SWOT) analysis, which explicitly joins the internal factors (S&W) to the external ones (O&T). There are four possible alternatives for the interrelationships of the four factors [24]:

- **S/T Strategy** shows strengths in the light of major threats. The aim is to defuse the threats using strengths.
S/O Strategy shows the strengths and opportunities. Essentially, the team should attempt to use its strengths to exploit opportunities. Such a strategy is generally attacking.

W/T Strategy shows the weaknesses against existing threats. Essentially, the team must attempt to minimize its weaknesses to avoid threats. Such a strategy is generally defensive.

W/O Strategy shows the weaknesses coupled with major opportunities. A team should do its best to use the opportunities and at the same time minimize the impacts of internal weaknesses.

The rationale of LCA is exactly in coordination with what happens in a sport league typically. However, there are some idealized rules which relax the real world situations and form the artificial championship environment of LCA.

- Idealized rule 1. It is more likely that a team with a better playing strength wins the game.
- Idealized rule 2. The outcome of a game is not predictable given the teams’ playing strengths completely.
- Idealized rule 3. The probability of the event that team \( i \) beats team \( j \) is assumed equal, from the both teams’ points of view.
- Idealized rule 4. The outcome of the game is only win or loss; there is no tie in the basic version of LCA.
- Idealized rule 5. When team \( i \) beats team \( j \), any strength helping team \( i \) to win, has a relative dual weakness causing team \( j \) to lose. In other words, any weakness is a lack of a particular strength.
- Idealized rule 6. Teams only focus on their upcoming match, regardless of the other future matches. Knowing the previous week results, each team sets its new formation based on the analysis of its current best formation.

Let \( f(X = (x_1, x_2, ..., x_n)) \) be an \( n \) variable numerical function that should be minimized over the decision space defined by \( l_d \leq x_d \leq u_d, d = 1, ..., n \). A team formation (a feasible solution to the problem) for team \( i \) at week \( t \) can be represented by \( X_i^t = (x_{i1}^t, x_{i2}^t, ..., x_{in}^t) \), with \( f(X_i^t) \) indicating the fitness value relevant to \( X_i^t \). This value is called the playing strength relevant to formation \( X_i^t \). \( B_i^t = (b_{i1}^t, b_{i2}^t, ..., b_{in}^t) \) denotes the best previously experienced formation of the team \( i \) until the week \( t \), yielding the best playing strength value (the best solution generated by the \( i^{th} \) agent in the population till iteration \( t \)). This is the best solution associated to the \( i^{th} \) member of the population that has been obtained so far. To determine \( B_i^t \), a greedy selection is employed between \( X_i^t \) and \( B_i^{t-1} \), based on the value of \( f(X_i^t) \) and \( f(B_i^{t-1}) \).

2.1 Generating the League Schedule

Like real sport games, LCA for the first step has the thread to set a weekly schedule for the teams to compete against each other in an artificial season. At the end of each season, all the \( L \) teams will have played \( L - 1 \) matches. In this way, each season would have \( L(L - 1)/2 \) matches. The algorithm terminates after passing a certain number of “seasons” \( S \).

2.2 Winner/Loser Recognition

It is more likely for a stronger team, having a better playing strength, to beat a weaker one (See Idealized Rule 1). Given an ideal league, away from the influences of uninvited effects, it is assumed a linear relationship between the playing strength of a team and the outcome of its game. This conclusion comes from the Idealized Rule 2.

The formation and the playing strength of the Team \( i \) in the Week \( t \) have been shown as \( X_i^t \) and \( f(X_i^t) \), respectively. Suppose that teams \( i \) and \( j \) would be competing together in the Week \( t \). Then \( p_i^t \) is the probability, with which Team \( i \) will win in the Week \( t \) (similarly \( p_j^t \) is defined). The rationale of such a simulation comes from the fact that \( p_i^t + p_j^t = 1 \) and is in
accordance with Idealized Rule 3. Suppose the ideal level of strength would be \( \hat{f} \) (the optimal value till the moment or a lower bound for it), then \( p^e_i \) is computed as follows [25]:

\[
p^e_i = \frac{f(x_i^j) - \hat{f}}{f(x_i^j) + \hat{f} - 2\hat{f}}
\]

Since the value of \( \hat{f} \) may be unavailable in advance, the best function value found so far (i.e., \( \hat{f} = \min_{i=1,...,L}\{f(B_i^j)\} \)) is used.

2.3 Setting up a New Formation

Therefore, there are two kinds of analysis according to the coaches’ points of view: Internal or External analyses, relatively if the coach considers the strengths and weaknesses of either the own or the opponent team, respectively (strength (S) and weakness (W) are considered as internal factors and opportunity (O) and threat (T) are considered as external factors).

![Artificial match analysis in LCA](Ref: [19])

The way, in which a new solution associated to an LCA’s team is generated, is governed via imitating the above match analysis process. In LCA, the artificial analysis of the team’s previous performance (at week \( t \)) is treated as internal evaluation (strengths/weaknesses) while analysis of the opponent’s previous performance is accounted as external evaluation (opportunities/threats). Figure 1 (left branch) shows such a hypothetical internal evaluation for team \( i \) in a flowchart format. Now, based on the league schedule at week \( t + 1 \), assume that the next match of team \( i \) is against team \( l \). If team \( l \) had won (lost) the game from (to) team \( k \) at week \( t \), then that success (loss) and the team formation behind it may be a direct threat (opportunity) for team \( i \). Apparently, such a success (loss) has been achieved by means of some strengths (weaknesses). Focusing on the strengths (weaknesses) of team \( l \), we gain an
An intuitive way to avoid from the possible threats (to receive benefits from the possible opportunities). Referring to idealized rule 5, we can focus on the weaknesses (strengths) of team \( k \) instead. Figure 1 (right branch) also demonstrates the hypothetical external evaluation followed by team \( i \).

Adopting a suitable strategy by all teams such as \( i (i = 1, ..., L) \), using Figure 1, all the teams should now attempt to fill the gaps in their formations for their next match at the end of week \( t+1 \). Let us take a quick glance at the indices again:

Index \( l \) denotes the opponent of team \( i \) in the week \( t + 1 \).

Index \( j \) denotes the opponent of team \( i \) in the week \( t \).

Index \( k \) denotes the opponent of team \( l \) in the week \( t \).

Now, let us assume that \( B_i^t, B_j^t, B_k^t \), and \( B_l^t \) are the best formations relative to the teams \( i, j, k \) and \( l \), respectively, till the week \( t \). As an example of match analysis running by the \( i^{th} \) team, let us assume that team \( l \) has lost its previous match to team \( k \). \( B_k^t - B_l^t \), addresses the gap between the playing styles of teams \( i \) and \( k \), sensed via the strengths of team \( k \) in week \( t \). The reason rests on the fact that the more team \( i \) arranges its formation in the week \( t + 1 \) closer to the formation of team \( k \) in week \( t \), the more probable it would also win the team \( l \) in the week \( t + 1 \). Similarly, it can be focused on the weaknesses of the team \( k \), by computing \( B_l^t - B_k^t \), when team \( i \) has lost to team \( l \) in week \( t \), in order to avoid a playing style similar to the one it had used. So are \( B_k^t - B_l^t \) and \( B_l^t - B_k^t \) when team \( i \) focuses on its previous result. Consequently, the new formation \( X_i^t = (x_{i1}^t, x_{i2}^t, ..., x_{in}^t) \) for team \( i (i = 1, ..., L) \), at week \( t+1 \) can be set up by one of the following equations.

If \( i \) won on \( j \) and \( l \) won on \( k \), too, the S/T strategy is adapted by:

\[
S/T \text{ equations: } x_{id}^{t+1} = b_{id} + y_{id}^t \left( \psi_1 r_1 (b_{id} - b_{kd}) + \psi_2 r_2 (b_{id} - b_{jd}) \right) \quad \forall d = 1, ..., n \tag{2}
\]

Else if \( i \) won on \( j \) but \( l \) lost to \( k \), the S/O strategy is adapted by:

\[
S/O \text{ equations: } x_{id}^{t+1} = b_{id} + y_{id}^t \left( \psi_2 r_1 (b_{kd} - b_{id}) + \psi_1 r_2 (b_{id} - b_{jd}) \right) \quad \forall d = 1, ..., n \tag{3}
\]

Else if \( i \) lost to \( j \) but \( l \) won on \( k \), the W/T strategy is adapted by:

\[
W/T \text{ equations: } x_{id}^{t+1} = b_{id} + y_{id}^t \left( \psi_1 r_2 (b_{kd} - b_{id}) + \psi_2 r_1 (b_{id} - b_{jd}) \right) \quad \forall d = 1, ..., n \tag{4}
\]

Else if \( i \) lost to \( j \) and \( l \) lost to \( k \), too, the W/O strategy is adapted by:

\[
W/O \text{ equations: } x_{id}^{t+1} = b_{id} + y_{id}^t \left( \psi_2 r_2 (b_{kd} - b_{id}) + \psi_1 r_1 (b_{id} - b_{jd}) \right) \quad \forall d = 1, ..., n \tag{5}
\]

End if

Where \( \psi_1 \) and \( \psi_2 \) are constant coefficients being used for scaling the contribution of “retreat” or “approach” components in comparison with another formation, respectively; \( r_1, r_2 \) are uniformly random numbers in \([0,1]\), \( d \) is the dimension or variable index, and the minus signs determine whether team \( i \) wants to approach toward the winner or to keep away from the loser. \( y_{id}^t \) in the equations (2) to (5) is a binary change variable which indicates whether or not the \( d^{th} \) element in the current best formation will change. Only \( y_{id}^t = 1 \) cause the value of \( b_{id}^t \) to be changed. Let us define \( y_{i1}^t = (y_{i1}^t, y_{i2}^t, ..., y_{in}^t) \) as the binary change array, in which the number of 1’s does not exceed \( q_i^t \). It is not usual for a coach to change all the aspects of the team. We can use a truncated geometric distribution to simulate the number of changes \( (q_i^t) \) [26]. This will help us choose the number of changes dynamically with more emphasis on the smaller/greater rate of changes. The following formula generates a random number for the changes made in \( B_i^t \) to get the new team formation \( X_i^{t+1} \).

\[
q_i^t = \left[ \frac{\ln(1-(1-(1-p_i^n)(n-q_0+1)r_i))}{\ln(1-p_i)} \right] + q_0 - 1; \quad q_i^t \in \{q_0, q_0 + 1, ..., n\} \tag{6}
\]
Where \( r \) is a random number in \([0,1]\), and \( p_c < 1 \), \( p_c \neq 0 \) is a control parameter. \( q_0 \) is the least number of changes found during the match analysis. We set \( q_0 = 1 \) according to Huseinzadeh Kashan [19]. The greater the positive (negative) value of \( p_c \) is, the smaller (greater) the number of changes more than \( q_0 \) is recommended. Using equations (6), we change the value of \( q_i^t \) elements of \( B_i^t \) randomly according to the equations (2) to (5). In order to keep feasibility, a random value is assigned within the range to the \( x_{id}^{t+1} \) if it lies outside of the feasible range (i.e. \( l_d > x_{id}^{t+1} \) or \( x_{id}^{t+1} > u_d \)).

3 OPTIMIZATION OF A FLOW-SHOP SYSTEM SCHEDULING BASED ON LCA

When we use LCA in a continuous context, every single solution is denoted by an \( n \)-dimension vector (\( n \) is the number of real valued variables). But the flow-shop system is naturally a discrete problem for which LCA is not directly applicable. In such a problem it is assumed there are \( n \) dimensions for \( n \) jobs. This way, each solution represents the sequence by which the jobs are introduced to the flow-shop system and is useful in arranging a Gant chart and priority assignment of operations. Our idea to tackle Flow-shop scheduling problem using LCA would be as to let LCA still searches in the continuous space, but do just the evaluations in the discrete space. In this way we can use LCA for scheduling problem with no extra adaptation cost. To this end we need a transformation function which converts any given \( 1 \times n \) continuous array to a \( 1 \times n \) sequence of integers. Such a transformation function can be obtained through using the notion of random keys [27]. The main idea of random keys is to use continuous numbers as sort keys for decoding a solution to show it as a sequence of integer numbers between 1 to \( n \), where each number is appeared just once. The continuous numbers will be ordered and decoded using a single passing rule to generate a permutation.

Let \( X_i^t = (x_{i1}^t, x_{i2}^t, ..., x_{in}^t) \) be the formation relevant to team \( i \) in week \( t \) which corresponds to the continuous values for \( n \) number of jobs in the flow-shop scheduling problem. To decode \( X_i^t \) as a sequence, we sort the keys in \( X_i^t \) in ascending order; now the ranking of each key in such a sequence, is replaced instead of the key itself.

For the sequence relevant to \( X_i^t \), once the jobs are accomplished by the machines in the flow-shop system, given their order in the sequence, the relevant makespan, i.e. the maximum of the completion times of all jobs, or the moment on which the last job leaves the system is recorded as the fitness value relevant to \( X_i^t \), that is \( f(X_i^t) \).

Team formations are initialized randomly in LCA. The following formula is used to construct the initial team formation relevant to team \( i \):

\[
x_{id}^0 = x_{min} + r(x_{max} - x_{min}), \quad \forall d = 1, ..., n
\]  

(7)

Where \( x_{min} = -n \) and \( x_{max} = n \) (\( n \) is the number of jobs) and \( r \) is a random number in \([0,1]\). During the generation of new solutions in LCA via equations (2) to (5), it is possible for \( x_{id}^t \) to get a value outside of the initial range of the search space. Whenever such infeasibilities occurred, we reinitialize the value of \( x_{id}^t \) using Equation (7).

4 COMPUTATIONAL EXPERIMENTS

In the previous sections we introduced LCA as a new method for solving permutation flow-shop scheduling problem and thus, measuring its pure performance and absolute potency is of particular interest. We know that the most successful algorithms for scheduling problems are based on the hybridization of evolutionary techniques with local search methods. But in a hybrid algorithm, this is the local search part that carries out the exploration of the search space and the performance of the evolutionary search is tangled with the local search effect. To evaluate the true performance of our approach, we abstain to employ further knowledge of the search landscape that could be acquired by any local search method.

The performance of an algorithm is generally subjected to the parameters setting which affects the search nature and convergence quality. Therefore, it would be demanded that
these parameters be set on a level through which high quality of solutions are obtained. A primitive study to recognize the affecting parameters showed that factors such as league size ($L$), $\psi_1$, $\psi_2$, and $p_c$ are of special consideration. To determine a proper level for the size of the league ($L$) two pairs are considered. The first pair investigates a league size equal to 20 and maximum number of weeks (iterations) equal to 50, i.e., $(20,50)$. The second pair is shown as $(10,100)$, where $n$ is the number of jobs. As can be seen, the total number of solutions generated and evaluated under each pair is the same for both pairs and is equal to 1000. We run LCA with both pairs. Primary results show that LCA performs significantly better with $(20,50)$. Since a larger population gives a better exploration ability to LCA. Given that the number of solutions generated in LCA is $S \times (L - 1)$, where $S$ is the number of seasons, we simply have chosen a suitable value for $S$, satisfying $S \times (L - 1) = 1000n$.

The parameter $p_c$ has a direct influence on preserving diversity among solutions generated in LCA. This parameter controls the number of elements to be changed in the population of vectors. Therefore, the less the positive it is, the more the current population is changed which in turn, gives a higher exploring ability to the algorithm. Decreasing the number of changes helps the algorithm converge to the best solution soon. However, it may be at the expense of a premature convergence. Increasing the number of changes makes the searching process of LCA more stable. Preliminary computations have shown that $p_c$ is better to be 0.7 among 0.1, 0.3, 0.5, 0.7, and 0.9. The studies show that 2 would be a suitable value for both $\psi_1$ and $\psi_2$.

A series of computational experiments are carried out, in which LCA is tested on a set of test problems. We adopt the benchmark test problem suite from Taillard [28], which is widely used in scheduling researches and can be found in [29]. Three sets of problems are adopted each of which consists of 10 problem instances. Each problem in the first set has 20 jobs to be scheduled on 5 machines in a flow-shop system. The problems in the second set have 50 jobs and 10 machines. These specifications for problems in the third set have 100 jobs and 20 machines, respectively. Each problem instance in each set is denoted with the run code $T_n - v$ with $v = 1, \ldots, 10$ and $n$ demonstrating the number jobs.

Each problem instance in each set is solved 25 times by LCA and the results are shown in the Tables 1-4. In Tables 1-3, columns with caption “$U$” shows the upper bound values for makespans provided by Taillard [28]. The “AVG” column demonstrates the average value among the best makespan values obtained by LCA at the end of each of 25 runs. Finally the “STD” column represents the standard deviation of the best makespan values obtained by LCA in each of 25 runs on each problem.

<table>
<thead>
<tr>
<th>Run code</th>
<th>$U$</th>
<th>AVG</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>T20-1</td>
<td>1278</td>
<td>1293.12</td>
<td>6.69</td>
</tr>
<tr>
<td>T20-2</td>
<td>1359</td>
<td>1365.44</td>
<td>1.98</td>
</tr>
<tr>
<td>T20-3</td>
<td>1081</td>
<td>1099.84</td>
<td>5.05</td>
</tr>
<tr>
<td>T20-4</td>
<td>1293</td>
<td>1308.16</td>
<td>4.86</td>
</tr>
<tr>
<td>T20-5</td>
<td>1236</td>
<td>1247.4</td>
<td>2.97</td>
</tr>
<tr>
<td>T20-6</td>
<td>1195</td>
<td>1205.48</td>
<td>6.48</td>
</tr>
<tr>
<td>T20-7</td>
<td>1239</td>
<td>1249.36</td>
<td>4.03</td>
</tr>
<tr>
<td>T20-8</td>
<td>1206</td>
<td>1215.52</td>
<td>4.77</td>
</tr>
<tr>
<td>T20-9</td>
<td>1230</td>
<td>1253.8</td>
<td>2.48</td>
</tr>
<tr>
<td>T20-10</td>
<td>1108</td>
<td>1121.76</td>
<td>5.53</td>
</tr>
</tbody>
</table>
Table 2: Results of LCA on problem instances with 50 jobs and 10 machines

<table>
<thead>
<tr>
<th>Run code</th>
<th>U</th>
<th>AVG</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>T50-1</td>
<td>3025</td>
<td>3130.16</td>
<td>11.74</td>
</tr>
<tr>
<td>T50-2</td>
<td>2892</td>
<td>3006.12</td>
<td>15.28</td>
</tr>
<tr>
<td>T50-3</td>
<td>2864</td>
<td>2998.6</td>
<td>13.13</td>
</tr>
<tr>
<td>T50-4</td>
<td>3064</td>
<td>3152.44</td>
<td>15.13</td>
</tr>
<tr>
<td>T50-5</td>
<td>2986</td>
<td>3126.84</td>
<td>19.58</td>
</tr>
<tr>
<td>T50-6</td>
<td>3006</td>
<td>3123.28</td>
<td>10.43</td>
</tr>
<tr>
<td>T50-7</td>
<td>3107</td>
<td>3220.16</td>
<td>19.53</td>
</tr>
<tr>
<td>T50-8</td>
<td>3039</td>
<td>3132.04</td>
<td>12.34</td>
</tr>
<tr>
<td>T50-9</td>
<td>2902</td>
<td>3018.84</td>
<td>13.96</td>
</tr>
<tr>
<td>T50-10</td>
<td>3091</td>
<td>3198.48</td>
<td>7.264</td>
</tr>
</tbody>
</table>

Table 3: Results of LCA on problem instances with 100 jobs and 20 machines

<table>
<thead>
<tr>
<th>Run code</th>
<th>U</th>
<th>AVG</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100-1</td>
<td>6286</td>
<td>6616.6</td>
<td>23.096</td>
</tr>
<tr>
<td>T100-2</td>
<td>6241</td>
<td>6566.96</td>
<td>32.449</td>
</tr>
<tr>
<td>T100-3</td>
<td>6329</td>
<td>6651.28</td>
<td>20.866</td>
</tr>
<tr>
<td>T100-4</td>
<td>6306</td>
<td>6606.88</td>
<td>29.532</td>
</tr>
<tr>
<td>T100-5</td>
<td>6377</td>
<td>6699.12</td>
<td>26.799</td>
</tr>
<tr>
<td>T100-6</td>
<td>6437</td>
<td>6733</td>
<td>26.58</td>
</tr>
<tr>
<td>T100-7</td>
<td>6346</td>
<td>6701</td>
<td>22.942</td>
</tr>
<tr>
<td>T100-8</td>
<td>6481</td>
<td>6822.52</td>
<td>26.588</td>
</tr>
<tr>
<td>T100-9</td>
<td>6358</td>
<td>6674.48</td>
<td>20.723</td>
</tr>
<tr>
<td>T100-10</td>
<td>6465</td>
<td>6764.24</td>
<td>23.085</td>
</tr>
</tbody>
</table>

Table 4: Comparisons of results between LCA, GA, PSO and DE

<table>
<thead>
<tr>
<th>Problem</th>
<th>LCA</th>
<th>GA</th>
<th>PSO</th>
<th>DE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ΔAVG</td>
<td>ΔSTD</td>
<td>ΔAVG</td>
<td>ΔSTD</td>
</tr>
<tr>
<td>T20</td>
<td>1.117</td>
<td>0.573</td>
<td>3.13</td>
<td>1.86</td>
</tr>
<tr>
<td>T50</td>
<td>3.783</td>
<td>0.745</td>
<td>5.61</td>
<td>1.41</td>
</tr>
<tr>
<td>T100</td>
<td>5.046</td>
<td>0.494</td>
<td>6.32</td>
<td>0.89</td>
</tr>
<tr>
<td>Average</td>
<td>3.315</td>
<td>0.604</td>
<td>5.02</td>
<td>1.38</td>
</tr>
</tbody>
</table>

In Table 5, the value of ΔAVG reported for each algorithm on each problem set shows the average relative deviation from the upper bound provided by Taillard [28], which is obtained by the following formula:

\[
\frac{\sum_{i=1}^{10} \sum_{j=1}^{25} (u_i - c_{max_{ij}})_{100}^{100}}{u_i}/250.
\]  

(8)

Where \(u_i\) is the upper bound of makespan provided by Taillard for problem instance \(i\) and \(c_{max_{ij}}\) is the best makespan value obtained by LCA on its \(j^{th}\) run on the \(i^{th}\) problem instance. The ΔSTD values show the standard deviations among \(\frac{(u_i - c_{max_{ij}})_{100}^{100}}{u_i}\) values.
Table 5: Statistics of the approximate two-sample t-tests between LCA versus GA, PSO, and DE

<table>
<thead>
<tr>
<th>Technique</th>
<th>Feature of t-test</th>
<th>$T_{20}$</th>
<th>$T_{50}$</th>
<th>$T_{100}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCA vs. GA</td>
<td>($t_0$, $df$)</td>
<td>$(-3.35,10)^* $</td>
<td>$(-1.44,11)$</td>
<td>$(-2.52,11)^* $</td>
</tr>
<tr>
<td>LCA vs. PSO</td>
<td>($t_0$, $df$)</td>
<td>$(-3.88,12)^* $</td>
<td>$(-2.64,13)^* $</td>
<td>$(-0.89,13)$</td>
</tr>
<tr>
<td>LCA vs. DE</td>
<td>($t_0$, $df$)</td>
<td>$(-4.27,12)^* $</td>
<td>$(-6.53,12)^* $</td>
<td>$(-4.71,11)^* $</td>
</tr>
</tbody>
</table>

As can be seen from resultant tables, the performance of LCA is acceptable and is the best in comparison to GA [30], PSO [30], and DE [31]. As illustrated in Table 5, the average deviation from upper bound ($\Delta_{AVG}$) for LCA is smaller than those given by the GA, PSO and DE. Such an outcome is true for all of the three sets of problems and hence it is independent from the size of the problem (the number of jobs and machines).

The same scenario is true for $\Delta_{STD}$. It means that the deviation in the performance of LCA, from one run to the other one, is the least. As can be seen from the Figure 2, the more the number of jobs are, the less the steep of the average makespan deviation line in LCA is, in comparison to that of PSO and DE, which means that as the size of problems get larger, the performance of LCA will deteriorate less than others. From this point of view, GA seems to be a good competitor for LCA in case of large problem instances. Figure 3 illustrates the changing trend in $\Delta_{STD}$ values for different algorithms.

As an important point, it should be noted that unlike GA, PSO and DE, which exhibit their results based on only 10 replications on each instance, the results reported by LCA are based on 25 replications.

![Figure 2: Value of $\Delta_{AVG}$ for different algorithms](image-url)
To find whether the performance of LCA differs significantly from GA, PSO, and DE we conduct an approximate two-sample t-test between the mean performance of the comparator algorithms and LCA according to the following statistic [32]:

\[
t_0 = \frac{\Delta_{LCA} - \Delta_{Comparator}}{\sqrt{\left(\frac{\Delta_{LCA}^{STD}}{n_1}\right)^2 + \left(\frac{\Delta_{Comparator}^{STD}}{n_2}\right)^2}}
\]

where $\Delta_{LCA}^{AVG}$, $\Delta_{Comparator}^{AVG}$, $\Delta_{LCA}^{STD}$, and $\Delta_{Comparator}^{STD}$ are obtained from Table 4. The superscript "Comparator" indicates one of GA, PSO or DE algorithms. $n_1$ and $n_2$ are the number of independent runs of two approaches being compared. More specifically we have $n_1=25$ and $n_2=10$. The degree of freedom is calculated as follows:

\[
df = \left[\frac{1}{\left(\frac{\Delta_{LCA}^{STD}}{n_1} + \frac{\Delta_{Comparator}^{STD}}{n_2}\right)^2}\left(\frac{\Delta_{LCA}^{STD}}{n_1}\right)^2 + \left(\frac{\Delta_{Comparator}^{STD}}{n_2}\right)^2 \right] \left(\frac{n_1 + n_2}{n_1 + n_2 - 2}\right)
\]

The values of these statistics have been listed in Table 5 for each pair-wise comparison. From the results of Table 5, we can see that on many cases, being conducted t-test, there is a significant difference between the mean values provided by LCA and other rivals. However, the statistical analysis fails to realize the superiority of LCA over PSO on $T_{50}$ and over DE on $T_{100}$. For both cases LCA reports smaller $\Delta_{AVG}$ and $\Delta_{STD}$ values.

In conclusion, from the results we can see that in the absence of any local information on the search landscape, our approach is more efficient than GA, PSO and DE algorithms and produces more competitive results.

5 CONCLUSION

In this study we used the recently proposed league championship algorithm (LCA) for scheduling jobs on a permutation flow-shop system to minimize the makespan. LCA is a population based stochastic search methodology that mimics the sporting competition in a sport league. At the heart of LCA is the artificial match analysis process where the new solutions are generated using a metaphorical SWOT analysis typically followed by coaches during the planning for their matches. While LCA do its search in continuous spaces, flow-shop scheduling problem is discrete in nature. We used a simple decoding mechanism to convert a continuous formation vector to a job permutation so that LCA can be used for solving all classes of scheduling problems.
We adopted a set of benchmark test problems and the results showed the better performance of LCA in comparison to those given by GA, PSO and DE. The deviation from the upper bound for LCA has an average value of 3.31% which is better than 5.02% for GA, 4.54% for PSO, and 4.38% for DE. The standard deviation of performance for the upper bound is 0.6 for LCA which is better than 1.38, 1.13, and 1.19 for GA, PSO, and DE, respectively. These results show the dominance of LCA over the competing algorithms. For future researches, we suggest one compare the performance of LCA in other areas of optimization such as project planning, facility location, etc. The studies also show that hybrid systems, consisted of an algorithm with a connection to a local search technique, have better performances in comparison to algorithms themselves. Hence, it is also a good suggestion to check the combination of LCA with a local search technique.

6 REFERENCES


