Modelling and Scheduling Lot Streaming Flexible Flow Lines

Bahman Naderi[a,*], Mehdi Yazdani[a]

[a] *Assistant Professor, Department of Industrial Engineering, Faculty of Industrial and Mechanical Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran*

Received 08 July, 2014; Revised 02 November, 2014; Accepted 28 February, 2015

Abstract

Although lot streaming scheduling is an active research field, lot streaming flexible flow lines problems have received far less attention than classical flow shops. This paper deals with scheduling jobs in lot streaming flexible flow line problems. The paper mathematically formulates the problem by a mixed integer linear programming model. This model solves small instances to optimality. Moreover, a novel artificial bee colony optimization is developed. This algorithm utilizes five effective mechanisms to solve the problem. To evaluate the algorithm, it is compared with adaptation of four available algorithms. The statistical analyses showed that the proposed algorithm significantly outperformed the other tested algorithms.

Keywords: Lot streaming, Flexible flow line scheduling, Mixed integer linear programming model, Artificial bee colony optimization

1. Introduction

Among the different complex combinatorial optimization problems, shop scheduling problems are the most active fields of research. Their applications could be found in a wide variety of situations, from information services to manufacturing systems. Scheduling can be viewed as an allocation of limited resources to perform a set of jobs. The resources could be machines in a workshop, runways at an airport, crews at a construction site, or processing units in a computing environment, and the jobs could be operations in a production process, take-offs and landings at an airport, stages in a construction project, or executions of computer programs. Effective scheduling increases efficiency and capacity utilization since it decreases task times and increases profitability.

Among different shop scheduling systems, a flow line problem is a multi-stage manufacturing process in which all jobs have the same generic recipe. That is, jobs are processed at stage 1, then stage 2, until the last stage. Traditionally, it is assumed that each stage has a single processor and a job is indivisible; as a result, the entire job must be completed before being transferred to the next stage. These might not be the case in many realistic situations.

The indivisibility of jobs leads to low machine utilization and long completion times. In practical scheduling, a job is actually a lot composed of many identical items. A job could, therefore, be split into a number of smaller sublots where each can be treated individually. When a job sublot is completed, it can be immediately moved to the next machine. Different sublots of the same job can thus be processed simultaneously at different stages. The process of splitting jobs into sublots is usually called lot streaming.

In real world cases, a shop with a single processor at each stage rarely exists. Commonly, processors are duplicated in parallel at stages. The purpose is to balance the capacity of stages, increase the overall shop floor capacity, reduce, if not eliminate, the impact of bottleneck stages and so on (Naderi et al. (2010)). The shop with multi processors at its stages (or at least one stage with more than one processor) is called a flexible flow lines. Najafi et al. (2012) develop a mathematical model for hybrid flow shop scheduling problems. Rezaeian et al. (2013) propose a hybrid approach based on genetic and imperialist competitive algorithm for flexible flow shops with multiprocessor tasks. Naderi and Sadeghi (2012) propose a multi-objective simulated annealing algorithm for hybrid no-wait flow shop scheduling problems with transportation times. More recently, Bożejko et al. (2013) present a parallel tabu search algorithm for hybrid flow shops, while Li et al. (2014) propose a hybrid variable neighbourhood search for the same problem.

The lot streaming technique has been receiving increasing attention after the pioneer work of Reiter (1966). Some papers discuss that lot streaming can significantly improve the schedule performance with respect to the makespan (Potts & Baker (1989); Kalir and Sarin (2000)). An extensive survey can be presented by Chang and Chiu (2005). The lot streaming flow shop has been commonly solved by meta-heuristics. Among all,

* Corresponding author Email address: bahman.naderi@aut.ac.ir

one can refer the reader to genetic and simulated annealing by Marimuthu and Ponnambalam (2005), simulated annealing algorithm and a tabu search algorithm by Marimuthu et al. (2007), discrete particle swarm optimization algorithm by Tseng and Liao [18], genetic and hybrid evolutionary algorithms by Marimuthu et al. (2008), ant-colony optimization and threshold accepting algorithms by Marimuthu (2009). More recent meta-heuristics are a local-best harmony search algorithm by Pan et al. (2011), discrete artificial bee colony algorithm by Pan et al. (2011), shuffled frog leaping algorithm by Pan et al. (2011), estimation of distribution algorithm by Pan and Ruiz (2012), and differential evolution algorithm and particle swarm optimization algorithms by Vijay et al. (2013).

Although there are several papers in the literature of lot streaming, they are mostly limited to classical flow shops. Despite the applicability of flexible flow lines in different industrial settings, the problem of lot streaming flexible flow lines is not well studied. This paper considers the problem of lot streaming flexible flow lines. It first formulates the problem by a mixed integer linear programming model. Moreover, a solution algorithm based on artificial bee colony optimization with many advance operators is developed. The algorithm is first comprehensively tuned. Then, it is evaluated against the optimal solutions obtained by the model on small problems. On large problems, the algorithm is compared with adaptations of our recently proposed algorithms, shuffled frog leaping algorithm (Pan et al., 2011), estimation of distribution algorithm (Pan and Ruiz, 2012), differential evolution algorithm (Vijay et al., 2013) and discrete artificial bee colony (Pan et al., 2012).

The rest of the paper is organized as follows. Section 2 reviews the literature. Section 3 formally defines and formulates the problem. Section 4 develops the algorithm for the problem under consideration. Section 5evaluates the model and algorithms for performance.

2. Problem Definition and Formulation

This section describes the flexible flow line problem. There is a set N of n jobs and a set M of m stages. At stage i, there are a set of $m_i$identical machines. Every job j is required to follow the exact same processing sequence across all stages. The processing route starts from stage 1, then stage 2 until stage m. Each job j is split into n_j sublot where all sublots of a job must be processed by exactly one machine $l$among machine available at each stage. Let$p_{j,k,i}$denote the processing time of kth sublot of job j at stage i.

Each machine can process no more than one sublot simultaneously while each sublot can be processed by no more than one machine at a time. The setup and transportation are negligible or included into processing times. There is no machine failure; hence, machines are continuously available for processing. Finally, all jobs are available at time 0 and the process of a sublot on a machine can be never interrupted; therefore, once the process starts, it continues until it finishes. The objective is to sequence jobs and schedule sublots so as to minimize makespan.

This paper develops a mathematical model for this problem. The application of integer programming models in solving scheduling problems starts with the pioneer model of Wagner (1959). Yet, regarding the limitation of computer capacity and the lack of specified software, the progress of research on this field is not as active as the other solution approaches. Due to recent advances obtained in computer capacity and advent of efficient specialized software, the MILP model development is each time becoming more and more interesting. Even if one accepts this idea that mathematical models cannot be efficient solution algorithms, they are the first natural way to approach scheduling problems by Pan (1997). They can explicitly describe all the characteristics of a scheduling problem. Furthermore, mathematical models are used in many solution methods such as branch and bound, dynamic programming and branch and price. More efficient MILP models would result in more effective solution methods. The parameters and indexes used in these models are:

n	The number of jobs
g	The number of stages
b, j	Index for jobs, $l, j \in \{1, 2, \dots, n\}$
n_j	The number of sub-lots of job j
k	Index for sub-lots, $k \in \{1,2, \dots, n_j\}$
i	Index for stages, $i \in \{1, 2, \dots, m\}$
m_i	The number of machines at stage i
l	Index for machines, $l \in \{1, 2, \dots, m_i\}$
$p_{j,k,i}$	The processing time of k-th sub-lot of job $j$at stage i
M	A large positive number

This model determines the relative precedence of jobs in pairs. The following variables are defined.

$X_{j,i,b}$	Binary variable taking value 1 if job j is processed after job b at stage i, and 0 otherwise.
$Y_{j,i,l}$	Binary variable taking value 1 if job j is processed at stage i on machine l, and 0 otherwise.
$C_{j,k,i}$	Continuous variable for the completion time of kth sub-lot of job j at stage i

The model is as follows.

$$Minimize\ C_{max}$$
$$Subject\ to:$$

$$\sum_{l=1}^{m_i} Y_{j,i,l} = 1 \qquad \forall_{j,i} \qquad (1)$$
$$C_{j,1,1} \geq p_{j,1,1} \qquad \forall_{j} \qquad (2)$$
$$C_{j,k,i} \geq C_{j,k,i-1} + p_{j,k,i} \qquad \forall_{j,k,i>1} \qquad (3)$$
$$C_{j,k,i} \geq C_{j,k-1,i} + p_{j,k,i} \qquad \forall_{j,k>1,i} \qquad (4)$$
$$C_{j,1,i} \geq C_{b,n_b,i} + p_{j,1,i} - M \cdot \left(3 - X_{j,i,b} - Y_{j,i,l} - Y_{b,i,l}\right) \qquad \forall_{i,l,j<n,b>j} \qquad (5)$$
$$C_{b,1,i} \geq C_{j,n_j,i} + p_{b,1,i} - M \cdot X_{j,i,b} - M \cdot \left(2 - Y_{j,i,l} - Y_{b,i,l}\right) \qquad \forall_{i,l,j<n,b>j} \qquad (6)$$
$$C_{max} \geq C_{j,n_j,m} \qquad \forall_{j} \qquad (7)$$
$$C_{j,k,i} \geq 0 \qquad \forall_{j,k,i} \qquad (8)$$
$$X_{j,i,b} \in \{0,1\} \qquad \forall_{j<n,b>j,i} \qquad (9)$$
$$Y_{j,i,l} \in \{0,1\} \qquad \forall_{j,i,l} \qquad (10)$$

Constraint set (1) specifies the job assignment of jobs to one machine among the machines available at each stage. Constraint set (2) assures that the first sublot of jobs at the first stage is greater than its processing time. Constraint sets (3) and (4) determine the minimum completion of each sublot at each stage regarding the completion time of that sublot in the previous stage and the completion time of the previous sublot of the same job in that stage. Constraint sets (5) and (6) determine the minimum completion time of the first sublot of each job at each stage regarding the completion time of the last sublot of the previous jobs processed before the job at the same machine. Constraint set (7) calculates makespan. Finally, constraint sets (8), (9) and (10) define the decision variables.

3. Artificial Bee Colony Optimization

Artificial bee colony optimization (ABCO) algorithm is one of the most recent population-based meta-heuristics proposed by Karaboga (2005). ABCO is motivated by the intelligent and well organized behaviour of honeybees supplying food. They colonially make use of current food sources and also search to discover the positions of new sources. ABCO has shown high performance in other problems such as flow shop scheduling problem by Tasgetiren et al. (2011) and Liu and Liu (2013).

Honeybees colonially perform the food search. That is, bees coordinate their activities to find food sources. Their colony includes three groups of specialized bees: employed bees, onlookers and scouts. The employed bees are those bees exploiting the current food sources. They bring loads of nectar from the food sources to the hive. In the hive, there is a dance floor where each employed bee dances to share information about its food source. The dance shows the quality of food source being exploited by the bee. Onlooker bees stay in hive and watch dances of all employed bees. Then, each onlooker selects one food source for further exploiting based on dances (i.e., the quality of food sources). The probability of each food source for being selected is the proportion of its quality. Thus, the low quality food sources attract less onlooker bees while the good one does more. After appropriately exploiting a food source (i.e., the nectar amount of the food source is exhausted), all the employed bees associated with this food source abandon it. In this case, the employed bee of a left food source becomes a scout bee and performs random search for a new food source. Once a scout/onlooker bee finds a food source, it again becomes an employed bee.

In ABC algorithm, the food source corresponds to a solution of the problem and its nectar amount of a food source the objective value. In fact, employed and onlooker bees perform exploitation search whereas scout bees carry out exploration search. The structure of ABCO algorithm is as follows. It starts from a population of pop food sources (each is an encoded solution). The nectar amount (objective value) of each food source is determined. Then, these food sources are iteratively exploited using two mechanisms: employed and onlooker bee. Meanwhile, the new area is explored for new food source using scout bee mechanism. Another enhancing feature is the diversifying mechanism by which the food sources are further explored.

In the employed bee mechanism, a new food source is generated from each food source. In the onlooker bee mechanism, positions around food sources are searched. In the scout bee mechanism, a scout bee searches for a new food source to replace an abandoned food source. In diversifying mechanism, the food source is further explored. Figure 1 shows the general procedure of the proposed ABCO algorithm.

Procedure: *Artificial bee colony optimization*
Initialization
While the stopping criterion is not met **do**
    Employed bee mechanism
    Onlooker bee mechanism
    Scout bee mechanism
    Intensifying mechanism
End for

Fig. 1. The procedure of the proposed ABCO

3.1 Encoding and initialization

To design a meta-heuristic the first two steps are to develop schemes to encode a solution of the original problem and decode an encoded solution to calculate the objective function value. The better these schemes are, the more effective the meta-heuristic becomes. To develop these schemes, the problem must be structurally analysed. The problem under consideration includes two decisions of assigning and sequencing.

The following encoding scheme is used. A solution is represented by a string including the permutation of job numbers from 1 to n. For example, consider a problem with n = 8. One possible solution can be

4	6	3	1	2	8	5	7

In this scheme, each job number represents its corresponding job. By scanning from left to right, the sequence of jobs at the first stage is determined. To specify the assigning decision, the following rule is used. The job is assigned to the first available machine. To determine the sequence of jobs at subsequent stages, the following rule is applied. The jobs are sorted at ascending order of completion time of the previous stage. That is, if a job is completed sooner, it is processed at the next stage sooner. The assignment of subsequent stages, the algorithm also uses the first available machine rule. To have initial solutions, pop solutions are generated as follows. Jobs are randomly sorted. The first job is scheduled. Then, jobs, one by one, are put into all possible positions among scheduled jobs. The best position is selected. Figure 2 shows the pseudo code of initialization mechanism.

Procedure: Initialization mechanism
For $k = 1$ to pop
    Sort jobs randomly in set J and set $D = \emptyset$.
    For $j = 1$ to n
        Take the jth job out of set J
        Test all possible positions in set D.
        Insert the job in the best position in set D
    Endfor
Endfor

Fig. 2. Initialization mechanism

3.2. Employed bee mechanism

The purpose of employed bee mechanism is to exploit the current food source. To implement this idea, a new solution is generated from vicinity of the current food source. For each solution i, one element j (real number) is randomly selected and combined with the corresponding element of another randomly selected food source k. Note that $k \neq i$. To do that, the following operators are used:

Step 1: Select two cut points between [1, n] randomly.
Step 2: Copy the job numbers among these two cut point from food source i to the new food source.
Step 3: Copy the remaining job numbers according to food source k to new food.

Let us illustrate the procedure by an example with n=8. Suppose food sources i and $k$are as follows:

Food source i:

4	6	3	1	2	8	5	7

Food source k:

5	1	2	8	6	7	4	3

If the two randomly selected points are 3 and 7 in the first step, there is the following incomplete food source.

New food source:

		3	1	2	8	5	

In the third step, the rest of job numbers are copied from food source k. The complete food source becomes:

New food source:

6	7	3	1	2	8	5	4

If this new food source is better than food source i, it is accepted and food source i is deleted. Otherwise, the new food source is rejected.

3.3. Onlooker bee mechanism

After searching current food sources, employed bees start to share the information about their food sources with onlooker bees. Then, they evaluate the nectar information obtained from all employed bees. According to the nectar amount of food sources, each onlooker bee chooses one food source. To determine the probability of each food source for being selected, the probability function, shown in Eq. (11) is used:

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \quad (11)$$

Where fit_i is the nectar amount (objective value) of food source i (solution i).

The selected food source undergoes the following local search by the onlooker bee. The jobs (one by one, without repetition and at random order) are taken out of permutation and put into another randomly selected position. If for a job a better position is found, the search restarts. If all jobs are tested and no improvement is made, the local search ends.

3.4. Scout bee mechanism

The employed and onlooker bees leave a food source already exhausted and search for a new food source. In this case, the bee is called a scout. A food source is exhausted if it is not improved for a number of consecutive iterations. In this case, the employed bee of that food source abandons the current food source and discovers a new food source randomly.
In the standard format, scout bees replace an exhausted food source with a new randomly selected food source. In this paper, another alternative is proposed. The aim is to make use of current knowledge of food sources. 20 new food sources from the best food source by a procedure are generated as follows. Three randomly selected jobs are randomly relocated into three new positions. Among these new food sources, the best one is selected. The criterion to determine if a food source is exhausted or not is as follows. After finding no improvement for a number of five iterations in row, a food source undergoes scout bee mechanism.

3.5. Intensifying mechanism

To further give capability of intensification to ABCO, the following mechanism is developed. At each iteration, all new food sources generated by scout bee mechanism as well as two other randomly selected food sources undergo the intensifying mechanism. This mechanism is a simulated annealing-like procedure to further explore the solution space. In this mechanism, the initial solution, say θ, is the selected food source. In this solution θ, one randomly selected job is relocated into a new random position. If this new solution, say θ', is better than solution θ, it is accepted. Otherwise, it is accepted with probability of

$$P_{\theta'} = 0.1 - \frac{f(\theta') - f(\theta)}{f(\theta)} = \frac{1.1f(\theta) - f(\theta')}{f(\theta)} \quad (12)$$

Where f(θ) is the makespan of solution θ. The maximum probability of acceptance for a worse solution is 0.1. The worse, a solution is, the less its probability becomes. The procedure of generating new solution continues until no better solution is found in 30 consecutive moves. At the end, the final solution is compared with the initial food source of intensifying mechanism. If it is better than the initial one, it is accepted. Otherwise, it is rejected. Figure 3 shows the pseudo code of intensifying mechanism.

Procedure: Intensifying mechanism
Take food source i as solution θ and Set $k = 1$;
While $k <= 1$
  Generate a new solution θ' by relocating one randomly selected job from solution θ into a new random position.
  If $f(\theta') < f(\theta)$ **then**
    Replace solution θ with solution θ'
    If $f(\theta) < f(i)$
      Replace food source i with solution θ and set $k = 0$;
    End if
  Else
    Probably replace solution θ with solution θ'
  End if
  Set $k = k + 1$;
End while

Fig. 3. The pseudo code of intensifying mechanism

4. Experimental Evaluations

This section evaluates the proposed model and algorithm, called ABCO, for performance. This experimental evaluation includes three different parts. First the parameters of the proposed algorithm are tuned. Then, using a set of small instances, the model's capability in solving the problem is evaluated. The algorithm is evaluated against the optimal solution obtained by the model. Finally, the algorithm is further evaluated by comparing its performance with adaptation of four recent algorithms in the literature of lot streaming flow shops: shuffled frog leaping algorithm (SFL) by Pan et al. (2011), estimation of distribution algorithm (EDA) by Pan and Ruiz (2012), differential evolution algorithm (DEA) by Vijay et al. (2013), discrete artificial bee colony (DABC) by Pan et al. (2011).

The proposed ABCO and the four algorithms brought from the literature are compared coded using Borland C++. The model is also coded into CPLEX 12.1. The model and algorithms are run on a PC with 2.40 GHz Intel Core i3 Duo and 4 GB of RAM memory. The stopping criterion for the meta-heuristics is set to n^2m milliseconds elapsed CPU time. For the model, the stopping criterion is set to 1000 seconds of computation time limit. The relative percentage deviation (RPD) is used as the performance measure. RPD is calculated as follows.

$$RPD = 100\left(\frac{C_{max} - LM}{LM}\right) \quad (13)$$

Where C_{max} and LM are makespan of solution found by the algorithm and the lowest makespan found by any of algorithms for an instance.

4.1. Experiment for parameter tuning

The correct choice of parameters significantly impact on the performance of meta-heuristics. One advantage of the

proposed ABCO is having only one parameter of population size (*pop*) of bees. To do the experiment, there are 48 instances by generating two instances for each of the following 24 combination sizes.

$n = \{20,50,80,120\}, m = \{2,4,8\}, m_i = \{2, U(1,4)\},$

Processing times are generated from a uniform distribution between (1, 99). The number of sublots for each job comes from a uniform distribution between (2, 4).5 levels for *pop*= {5,20,40,70,100} are considered. All 24 instances are solved by four different ABCs. The results are analysed by analysis of variance (ANOVA) and least significant deviation (LSD) statistical tests. Figure 4 shows the average RPD obtained by ABC of each level as well as the LSD intervals. As it can be seen, pop of 20 is the best level with average RPD of 0.94% although it is statistically similar to *pop* of 40.

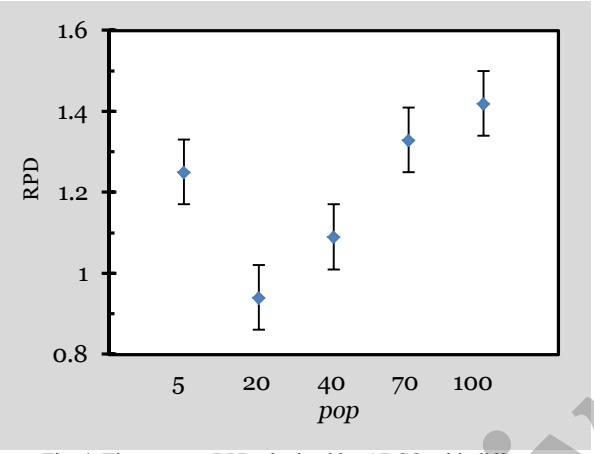

Fig. 4. The average RPD obtained by ABCO with different *pop*

4.2. Experiment with small-sized instances

This subsection evaluates the proposed model and general performance of the tested algorithms on a set of small sized instances. Note that the tested meta-heuristics do not guarantee the optimality. In this regard, there are12 instances, two instances for each of the following combination.

$$n = \{6,8,10\}, m = \{2,4\}, m_i = \{2\}.$$

The number of sublots and processing times are generated from uniform distributions over (2, 4) and (1, 99), respectively.

Table 1 shows the results obtained by the model and algorithms. In this table, for the model, there are two columns. The column "time" shows the computational time of the model elapsed to optimally solve the corresponding instance. The column "gap" shows the optimality gap of model for those unsolved instances. The model optimally solves all the instances up $n = 10$ and $m = 2$ in less than 35 seconds. It also solves one of two instances with $n = 10$ and $m = 4$ in 294 seconds. In sum, the model solves 11 instances out of 12 ones. Regarding the algorithms, ABCO and EDA solve 9 instances out of the 11 instances to optimality. SLF presents the worst performance with average optimality gap of 1.1%.

4.3. Experiment with large sized instances

This section further compares the five tested algorithms (SFL, EDA, DEA, DABC and ABCO) on large-sized instances. As large-sized instances, there are120 instances, five instances for each of the following 24combination sizes.

$n = \{20,50,80,120\}, m = \{2,4,8\}, m_i = \{2, U(1,4)\},$

Processing times and the number of sublots are generated from uniform distributions between (1, 99) and (2, 4), respectively.

Table 2 shows the results, averaged for each combination of n and m (10 data per average). As it can be seen, the proposed ABCO outperforms the other algorithms with average RPD of 0.59%. The second best is EDA with RPD of 1.07%. Among the remaining algorithms, DEA and DABC provide the average RPD of 2.32% and 2.5%, respectively. The worst performing algorithm is SFL with average RPD of 3.07%. The paper conducts the ANOVA and LSD tests to compare the algorithms. Figure 5 shows the means plot with LSD intervals. As can be seen from the table, the proposed ABCO provides statistically better results among the tested algorithms.

To further evaluate the algorithms, the relation between the performance of the algorithms and the problem size is analysed. First, the influence of n over the different algorithms is assessed. Figure 6 presents the average RPD obtained by any of algorithms in different sizes of n. There is a clear trend that the proposed ABCO provides better results in larger sizes of n.

Table 1
The optimality gap of the model and the tested algorithms

N	*m*	Model		Algorithms (Opt. gap %)				
		Time (sec)	Opt. gap	SFL	EDA	DEA	ABCO	DABC
6	2	<1	0% (2)	0.0	0.0	0.0	0.0	0.0
	4	<1	0% (2)	0.0	0.0	0.0	0.0	0.0
8	2	5.73	0% (2)	0.9	0.0	0.0	0.0	1.4
	4	20.18	0% (2)	1.3	0.0	1.6	0.0	0.0
10	2	34.07	0% (2)	2.5	1.7	1.4	2.1	3.1
	4	294.41	7.54% (1)	2.7	1.5	1.5	1.5	2.1
Average				1.10	0.44	0.68	0.52	1.01

Table 2
The average RPD of the tested algorithms

N	*M*	Algorithms				
		SFL	EDA	DEA	ABCO	DABC
10	2	3.11	0.55	2.28	0.56	2.22
	4	2.98	0.83	2.24	0.85	2.46
	8	3.27	0.77	1.97	0.48	2.1
40	2	3.13	0.89	1.56	0.52	2.61
	4	2.97	0.99	2.26	0.68	2.14
	8	2.52	1.01	2.51	0.73	2.27
70	2	2.59	0.93	2.62	0.69	2.55
	4	2.84	1.25	2.45	0.44	2.45
	8	3.24	1.33	1.98	0.52	2.81
100	2	3.22	1.47	2.96	0.62	2.87
	4	3.4	1.29	2.46	0.41	2.35
	8	3.59	1.53	2.5	0.56	3.17
Ave.		3.07	1.07	2.32	0.59	2.5

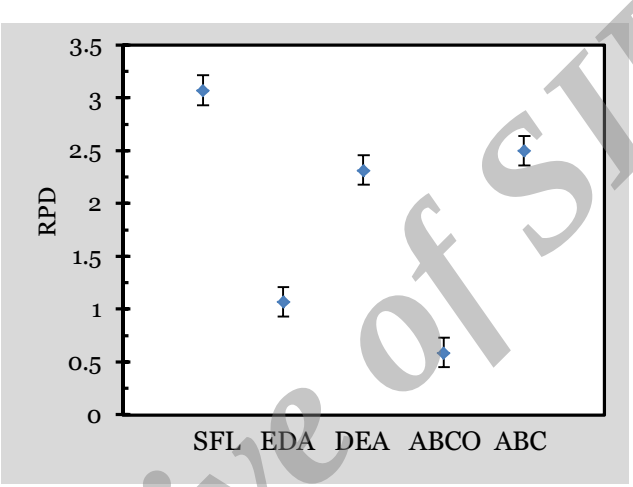

Fig. 5. Means plot with LSD intervals for the different algorithms

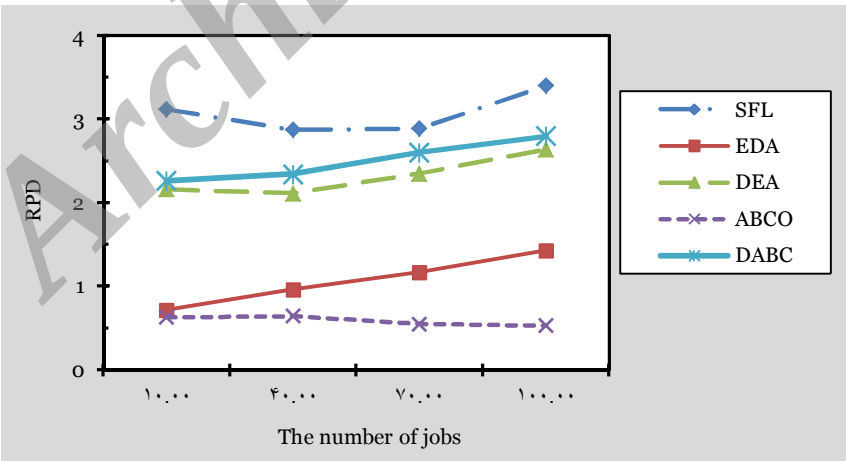

Fig. 6. Means plot of algorithms versus the number of jobs

5. Conclusion

The current literature on lot streaming scheduling has focused on flow shop problems. In flow shops, it is assumed that there is only one processor at each working stage. Yet, in practice, shops duplicate machines in parallel at each stage. This paper extended the problem of lot streaming flow lines with parallel machines. The problem has been mathematically formulated by a mixed integer linear programming model. The model is based on the concept of relative-sequence of jobs. Using commercial software CPLEX, the model solved instances

up to 8 jobs to optimality in less than 35 seconds and instances with 10 jobs in less than 294 seconds.

In addition to the model, a novel artificial bee colony algorithm was developed to solve larger instances. This algorithm was developed with 5 mechanisms of initialization, employed bee, onlooker bee, scout bee and intensifying. The proposed algorithms were compared with 4 available algorithms (shuffled frog leaping, estimation and distribution, differential evolution and particle swarm optimization algorithms). Among the tested algorithms, the artificial bee colony and estimation and distribution algorithms outperform the others with average RPD of 0.58% and 1.07%, respectively, over 120 large instances. In small instances the proposed algorithm yields the best results with RPD of 0.52%. The statistical tests showed that the proposed algorithm significantly obtains better results than the others.

As an interesting future research direction, one can study the more realistic version of the problem with other assumptions like setup times. Since there are two or more objectives to optimize simultaneously in practical industrial settings, the problem can also be extended to the multi objective case. Regarding the proposed solution method, the artificial bee colony can be applied to solve other scheduling related problems.

References

[1] Bożejko, W., Pempera, J., Smutnicki, C., (2013). Parallel tabu search algorithm for the hybrid flow shop problem, Computers and Industrial Engineering, 65(3), 466-474

[2] Chang, J.H., Chiu, H.N. (2005). A comprehensive review of lot streaming. International Journal of Production Research, 43(8), 1515–1536.

[3] Liu, Y.F., Liu, S.Y. (2013). A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. Applied Soft Computing, 13(3), 1459-1463.

[4] Li, J.Q., Pan, Q.K., Wang, F.T. (2014). A hybrid variable neighborhood search for solving the hybrid flow shop scheduling problem, Applied Soft Computing, 24, 63-77.

[5] Kalir, A.A., Sarin, S.C. (2000) Evaluation of the potential benefits of lot streaming in flow-shop systems. International Journal of Production Economics, 66, 131-142.

[6] Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department.

[7] Marimuthu, S., Ponnambalam, S.G. (2005).Heuristic search algorithms for lot streaming in a two-machine flowshop. International Journal of Advanced Manufacturing Technology, 27, 174–180.

[8] Marimuthu, S., Ponnambalam, S.G., Jawahar, N. (2007). Tabu search and simulated annealing algorithms for scheduling in flow shops with lot streaming. Proceedings of the Institution of Mechanical Engineers Part B—Journal of Engineering Manufacture, 221(2), 317–331.

[9] Marimuthu, S., Ponnambalam, S.G., Jawahar, N. (2008). Evolutionary algorithms for scheduling m-machine flow shop with lot streaming. Robotics and Computer-Integrated Manufacturing, 24(1), 125–139.

[10] Marimuthu, S., Ponnambalam, S.G., Jawahar, N. (2009). Threshold accepting and Ant-colony optimization algorithms for scheduling m-machine flow shops with lot streaming. Journal of Materials Processing Technology, 209, 1026–1041.

[11] Naderi, B., Ruiz, R., Zandieh M. (2010). Algorithms for a realistic variant of flowshop scheduling. Computers and Operations Research, 37,236–246.

[12] Naderi, B., Sadeghi, H. (2012). A Multi-objective simulated annealing algorithm to solving flexible no-wait flowshop scheduling problems with transportation times, Journal of Optimization in Industrial Engineering, 5(11), 33-41.

[13] Najafi, E., Naderi, B., Sadeghi, H.,Yazdani, M. (2012). A mathematical model and a solution method for hybrid flow shop scheduling, Journal of Optimization in Industrial Engineering, 5(10), 65-72.

[14] Pan, C.H. (1997). A study of integer programming formulations for scheduling problems. International Journal of Systems Science, 28, 33–41.

[15] Pan, Q.K., Wang, L., Gao, L., Li, J. (2011). An effective shuffled frog-leaping algorithm for lot-streaming flow shop scheduling problem. International Journal of Advanced Manufacturing Technology, 52,699–713.

[16] Pan, Q.K., Tasgetiren, M.F., Suganthan, P.N., Chua, T.J. (2011). A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Information Sciences, 181, 2455–2468.

[17] Pan, Q.K., Suganthan, P.N., Liang, J.J., Tasgetiren, M.F. (2011). A local-best harmony search algorithm with dynamic sub-harmony memories for lot-streaming flow shop scheduling problem. Expert Systems with Applications, 38, 3252–3259.

[18] Pan, Q.K., Ruiz, R. (2012). An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. Omega, 40, 166–180.

[19] Potts, C.N., Baker, K.R. (1989). Flow-shop scheduling with lot streaming. Operations Research Letters, 8(6), 297–303.

[20] Reiter, S. (1966). System for managing job-shop production. Journal of Business, 39(3), 371–393.

[21] Rezaeian, J., Seidgar, H., Kiani, M. (2013). Scheduling of a flexible flow shop with multiprocessor task by a hybrid approach based on genetic and imperialist competitive algorithms, Journal of Optimization in Industrial Engineering, 6(13), 1-13.

[22] Tasgetiren, M.F., Pan, Q.K., Suganthan, P.N., Oner, A. (2011).A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. Information Sciences, 181(16), 3459-3475.

[23] Tseng, C.T., Liao, C.J. (2008). A discrete particle swarm optimization for lot-streaming flowshop scheduling problem. European Journal of Operational Research, 191, 360–373.

[24] Yoon, S.H., Ventura, J.A. (2002). An application of genetic algorithms to lot streaming flow shop scheduling. IIE Transactions, 34, 779–787.

[25] Vijay Chakaravarthy, G., Marimuthu, S., Sait, A.N.(2013).Performance evaluation of proposed Differential Evolution and Particle Swarm Optimization algorithms for scheduling m-machine flow shops with lot streaming. Journal of Intelligent Manufacturing, 24, 175–191.

[26] Wagner, H.M. (1959). An integer linear-programming model for machine scheduling. Naval Research Logistics Quarterly, 6, 131–140.