Journal of Optimization in Industrial Engineering 10 (2012) 65-72

A Mathematical Model and a Solution Method for Hybrid Flow Shop Scheduling

Esmaeil Najafi[a], Bahman Naderi[b], Hasan Sadeghi[c], Mehdi Yazdani[d,*]

[a] *Assistant Professor, Department of Industrial Engineering, Science & Research Branch, Islamic Azad University, , Tehran, Iran*

[b] *Assistant Professor, Department of Industrial Engineering, Faculty of Engineering, University of Kharazmi, Karaj, Iran*

[c] *BSc, Young Researchers Club, Qazvin Branch, Islamic Azad University, Qazvin, Iran*

[d] *Instructor, Department of Industrial Engineering, Qazvin Branch, Islamic Azad University, , Qazvin, Iran*

Received 12 September, 2011; Revised 9 February, 2012; Accepted 27 March, 2012

Abstract

This paper studies the hybrid flow shop scheduling where the optimization criterion is the minimization of total tardiness. First, the problem is formulated as a mixed integer linear programming model. Then, to solve large problem sizes, an artificial immune algorithm hybridized with a simple local search in form of simulated annealing is proposed. Two experiments are carried out to evaluate the model and the algorithm. In the first one, the general performance of the model and the proposed algorithm is experimented. In the next one, the presented algorithm is compared against some other algorithms. The results support high performance of the proposed algorithm.
Keyword: Scheduling, Hybrid flow shop, Mathematical model, Mixed integer linear program, Artificial immune algorith

1. Introduction

Scheduling problems are the allocation of resources to perform a set of activities in a period of time (Pinedo, 2008). Flow shop scheduling is one of the well-recognized scheduling systems in which a set of n jobs need to be processed in a set of m stages where each one has one machine. The jobs visit stages in the same route, starting from stage 1 to stage m. Many assumptions are considered by researchers to actualize the problem of flow shops. They hybridize the flow shop with another classical scheduling problem called parallel machines. In this case, each stage has a certain number of identical machines in parallel, and each job is processed at only one machine among machines at each stage. By duplicating machines in parallel, the capacity of stages is balanced; the overall capacities of the shop floor is increased; or the impact of bottleneck stages on the overall shop floor capacities is either eliminated or reduced (Naderi et al., 2010a; Behnamian and Fatemi Ghomi, 2011).

The hybrid flow shop scheduling (HFS) problem has several applications in real industrial settings including automobile manufacture (Kurz and Askin, 2006) and printed circuit board manufacture (Zandieh et al., 2006). Each job j requires a fixed and pre-determined amount of processing time at each stage i. This amount is denoted by pji. Moreover, all jobs are independent and available for process at time 0. At each stage i, there is a number of m_i identical machines. All machines are continuously available. Each machine l can at most process a job j at a time. Each job j is processed by exactly one machine. The process of a job j at stage i cannot be interrupted. There is no transportation time between stages. There are incapacitated buffers between stages. Therefore, if a job needs a machine that is busy processing another job; it can wait indefinitely until it is available. (Naderi and Ruiz, 2010).

It is common in the scheduling literature to look for a sequence of jobs that minimizes the maximum completion time (or makespan) which coincides with the time at which the last job in the sequence finishes at the last machine. Other frequently considered criteria are total tardiness (TT) minimization, denoted as $\sum T_j$, where T_j is the tardiness of job j at the shop. TT is more realistic case of makespan (Ruiz and Allahverdi, 2007) and calculated as follows.

$$T_j = \max(0, C_j - d_j)$$

where C_j and d_j are completion time and due date of job j.

The HFS problem is classified as an NP-hard problem (Jin et al., 2006). Due to the inherent complexity of hybrid flow shop scheduling, no effective exact method has been

* Corresponding author E-mail address: M_Yazdani@qiau.ac.ir

developed so far to tackle these problems within a reasonable amount of time. Hence, a variety of algorithms divided into two main groups, heuristics and meta-heuristics, have been applied to solve these problems to find optimal or near optimal schedule (Kurz and Askin, 2003; Kurz and Askin, 2004). Kurz and Askin (2003) examined scheduling rules for sequence dependent setup time hybrid flow shops. They explored three classes of heuristics. The first class of heuristics (cyclic heuristics) is based on simplistic assignment of jobs to machines with little or no regard for the setup times. The second class of heuristics is based on the insertion heuristic for the traveling salesman problem (TSP). The third class of heuristics is based on Johnson's rule. They proposed eight heuristics (CH, RCH, SPTCH, FTMIH, CTMIH, MMIH, 1,g Johnson's rule, g/2, g/2 Johnson's rule) and compared the performance of those on a set of test problems.

Moreover, Kurz and Askin (2004) formulated the sequence dependent setup time hybrid flow shops as an integer programming model. Because of the difficulty in solving the IP model directly, they developed a random keys genetic algorithm (RKGA). Problem data was generated to evaluate the RKGA with other scheduling heuristics rules, which they proposed aforetime. Zandieh et al. (2006) studied the same problem and proposed an artificial immune algorithm. This algorithm's structure is similar to RKGA, yet it employs affinity procedure. A complete survey of scheduling problems with setup times is given by Allahverdi et al (2008).

With respect to the corresponding explanation, in this paper we explore HFS problems to minimize total tardiness. A mathematical model is proposed, and then a Hybrid artificial immune algorithm is presented to solve the problem.

The rest of the paper is organized as follows. Section 2 proposes the mixed integer linear program. Section3 develops an artificial immune algorithm. Section 4 evaluates the model and algorithm. Section 5 concludes the paper and proposes some future search directions.

2. Mathematical Model for the Hybrid Flow Shop Problem

The purpose of using mathematical models is to explicitly define all the characteristics of a scheduling problem (Pan, 2007). Furthermore, mathematical models could be utilized in many solution methods such as branch and bound, dynamic programming and branch and price. Therefore, effective model presentation is of interest (Naderi and Ruiz, 2010; Matta, 2009). This paper develops one mixed integer linear programming model for HFS problems. As mentioned earlier, in HFS, solving the problem is equivalent to sequencing jobs on stages as well as assigning jobs to machines inside each stage. Therefore, the model finds both job sequence and job assignment. The following parameters and indices are used in the model.

n	Number of jobs
m	Number of machines
j,k,t	Index for jobs/positions where $\{1,2,....,n\}$
d_j	Due date of job j
i	Index for stages where $\{1,2,....,m\}$
m_i	Number of machines in stage i
l	Index for machines at stage i where $\{1,2,....,m_i\}$
$O_{j,i}$	Operation of job j at stage i
$p_{j,i}$	Processing time of job j at stage i
M	A large positive number

The following binary variables are defined.

$X_{j,i,k}$	Binary variable taking value 1 if job j occupies position k at stage i, and 0 otherwise.
$Y_{i,k,l}$	Binary variable taking value 1 if the job in position k of stage i is processed on machine l, and 0 otherwise.
$S_{i,k}$	The starting time of the job in position k at stage i
$F_{j,i}$	The starting time of the job j at stage i
T_j	The tardiness of the job j

The model formulates the problem as follows.

$$\text{Minimize} \quad \sum_{j=1}^{n} T_j \tag{1}$$

Subject to:

$$\sum_{k=1}^{n} X_{j,i,k} = 1 \qquad \forall j,i \tag{2}$$

$$\sum_{j=1}^{n} X_{j,i,k} = 1 \qquad \forall i,k \tag{3}$$

$$\sum_{l=1}^{m_i} X_{i,k,l} = 1 \qquad \forall i,k \tag{4}$$

$$F_{j,i+1} \geq F_{j,i} + p_{j,i} \qquad \forall j, i \leq m \tag{5}$$

$$S_{i,k} \geq S_{i,t} + \sum_{j=1}^{n} X_{j,i,t}.p_{j,i} \qquad \forall i,l,k>1,t<k \tag{6}$$

$$S_{i,k} \geq F_{j,i} - M(1 - X_{j,i,k}) \qquad \forall i,k,j \qquad (7)$$

$$S_{i,k} \geq F_{j,i} + M(1 - X_{j,i,k}) \qquad \forall i,k,j \qquad (8)$$

$$T_j \geq \left(F_{j,\mathrm{m}} + \sum_{k=1}^{n} X_{j,m,k} . p_{j,i} \right) \qquad \forall j \qquad (9)$$

$$S_{i,k}, \mathrm{F}_{j,i}, T_j \geq 0 \qquad (10)$$

$$X_{j,i,k}, Y_{i,k,l} \in \{0,1\} \qquad (11)$$

In (1), total tardiness is calculated. Constraint set (2) states that every job occupies exactly one position at each stage. Constraint set (3) ensures that every position at each stage is occupied exactly once. Constraint set (4) enforces that each job is assigned to one machine, among available machines at each stage. Constraint set (5) ensures that $O_{j,i}$ starts after the completion of $O_{j,i-1}$. Constraint set (6) assures that if the job in position k at stage i cannot begin before the completion of its previous jobs at the same machine. In other words, it holds the assumption that one machine can process at most one job at a time. Constraint sets (7) and (8) make the relation between each job and its position at each stage. Constraint set (9) calculates the tardiness of jobs. Finally, Constraint sets (10) and (11) define the decision variables.

To formulate a problem with n jobs and m stages, the model needs $nm(n+\sum m_i)$ binary variables, $(2\mathrm{nm}+\mathrm{n})$ continuous variables and $(3\mathrm{nm}-\mathrm{m}+\mathrm{n}^2\mathrm{m}+\mathrm{m}^2\mathrm{n}+\mathrm{n})$ constraints.

3. Artificial Immune Algorithm

The artificial immune algorithm (AIA) is an approximation algorithms and classified as population-based meta-heuristic (Prakash et al., 2008). The original intention is inspired by the simulation of the physiological immune system of natural living organisms defending the body from foreign pathogens (bacteria or virus). The mechanisms work by first recognizing foreign substances known as antigens. The immune systems then generate a set of antibodies to eliminate the antigens. The mechanisms are able to recognize which antibodies are better at eliminating the antigens and produce more variations of those antibodies in the next generation of antibodies. Each antibody is assigned a value called Affinity showing the capability of that antibody to eliminate antigens. The antigen, affinity and antibody in the AIA are equivalent to the problem to be solved, objective function and feasible solution for a conventional optimization method, respectively.

Three commonly applied types of AIAs are cloning selection algorithm, immune network algorithm and negative selection algorithm. It is known that cloning selection algorithms are more suitable in tackling scheduling problems (Naderi et al., 2009b). The main operators are cloning selection and affinity maturation. When an antigen is detected, those antibodies that best recognize this antigen will proliferate by cloning. This process is called cloning selection principle. Affinity maturation consists of two basic concepts: hypermutation and receptor editing. Hypermutation is similar to mutation in genetic algorithms with the following distinctive specifications. Mutation corresponds to the creation of new solutions which are structurally and behaviorally similar to their creators but not exactly the same with a fixed rate while in hypermutation; inferior antibodies are mutated at higher rate than the good antibodies suffer. The procedure of hypermutation is handled by receptor editing.

3.1. The proposed immune system algorithm

The proposed AIA searches a problem space with a population of antibodies each of which is an encoded solution. An affinity value is assigned to each antibody according to its performance. The more desirable the antibody, the higher this value becomes. The population evolves by a set of operators until some stopping criterion is met. A typical iteration of AIA, generation, proceeds according to an author-defined affinity function and the number of the clones that would be proliferated from each antibody is calculated. All the proliferated clones are put in a mutating pool. A selection mechanism chooses the clones in the current mutating pool in such a way that clones with lower TT have higher chances of being selected. The selected clones hypermutate and generate new antibodies, namely offspring. Afterwards, the new population is evaluated again and the whole process repeats.

In the following subsection, we introduce the specifications of the proposed AIA. To this end, we first present our antibody representation (encoding scheme) which makes a solution recognizable for the algorithm. Then, we present the operators that we use in the proposed algorithm.

3.2. Antibody representation and initialization

In AIA, each antibody represents a solution. We use job-based representation to encode a solution. In job-based representation, the permutation of jobs in the first stage is determined, and then by a dispatching rule the jobs are assigned to the machines. In the proposed algorithm, the job is assigned to the first available machine at each stage. In HFS problems, the first available machine results in the earliest completion time. The sequence of jobs in subsequent stages is according to their completion time in the previous stage. The proposed algorithm starts form NEH algorithm (Nawaz et al., 1983), and random solutions form feasible region.

3.3. Cloning selection procedure

The capability of each antibody to fight against antigens is measured through affinity. Antibodies with higher affinity value are better at eliminating antigens. In our case, it was necessary to consider the following issues: 1- The antigen is our optimization problem. 2- An antibody is an encoded solution. 3- Since the objective is the minimization of TT, better antibodies (solutions) are those results in lower TT (i.e. they can fight against the antigen better). Since the higher affinity value means the better antibody, we define the following function to calculate each antibody's affinity:

$$Affinity\ (t) = 1/\ TT\ (t) \qquad (12)$$

The lower the TT, the higher the affinity value becomes. The probability of the cloning of each antibody to transfer into the mutating pool is directly proportional to its affinity value. Therefore, the antibodies with lower TT more likely have a lot of clones. The mutating pool has a fixed size of pop-size clones. One of them is fulfilled by the best antibody. To fulfill the rest, we apply a selection mechanism (popsize-1) times, and each time, an antibody is copied into the pool. We use ranking selection (Goldberg, 1989).

3.4. Affinity maturing procedure

In Affinity maturing procedure, all of the clones existing in the pool undergo an operator which makes a random change in the clones. This operator is called hypermutation. Dependent on the affinity value, each clone suffer different rate of change. The inferior clones undergo high rate of hypermutation whereas better clones suffer a slight change. As a low rate hypermutation, we utilize SHIFT mutation. In SHIFT mutation, a randomly selected operation is randomly relocated in the sequence. As high rate hypermutation, we use an operator working as follows: The positions of two randomly selected operations are swapped. In addition to the operators we just defined, we need a criterion to determine the condition under which we exploit one of the operators. We define a criterion as such: each clone (t) undergoes the low rate hypermutation

$$\text{If } \frac{TT(t) - TT(\text{best antibody})}{TT(\text{best antibody})} < 0.1 \qquad (13)$$

Otherwise, it undergoes the high rate hypermutation. Contrary to previous work (Engin et al., 2004) in which the offspring are accepted only if they have lower TT than their creators, we use simple simulated annealing-like acceptance criterion. Besides the acceptance of better offspring, inferior offspring might be accepted by the following random mechanism:

$$\text{if random} < \exp\left(\frac{TT(\text{Offespring}) - TT(\text{Creator})}{20}\right) \qquad (14)$$

where random is a random number between (0, 1). This allows the algorithm easily to avoid getting trapped into local optima. It is necessary to indicate that we apply the elite strategy meaning that the best clone is directly copied into the next generation. Figure 1 presents the general outline of the proposed AIA.

```
Procedure AIA

Initialize (population and Best)
while the stopping criterion is not met do
    for t = 1 to pop-size do
        Apply ranking selection
        Put the selected antibody into the mutating pool
    Endfor
    for t = 1 to pop-size do
        Take the t-th antibody of the pool
        if (TT(t)-Best)/Best < 0.1 do
            Apply low rate hypermutation
        elseif
            Apply high rate hypermutation
    endfor
    Update Best
Endwhile
```

Fig. 1. Pseudo code of the proposed AIA

3.5. The hybridization of the proposed artificial immune algorithm

Many researchers concluded that hybrid approaches for job shops could end up with high quality results (Zhang et al., 2008; Heinonen and Pettersson, 2007). The purpose of the hybridization is to overcome shortcomings of each algorithm. AIA is known to be a meta-heuristic powerful in diversification. Preliminary tests demonstrated that the performance of AIA is significantly influenced if we hybridize it with another algorithm which is powerful in intensification. We examined several meta-heuristics; it finally turned out that hybridization with a local search-based meta-heuristic can obtain convincing results. Among the local search-based meta-heuristic, simulated annealing (SA) is known to be a very fast algorithm possessing intensifying operators. Therefore, we have been galvanized into the hybridization of the AIA and SA. In this case, our hybrid algorithms have a diversification technique to investigate new and unknown areas in the search space, and intensification technique to make use of knowledge obtained at points previously visited to assist to find promising solutions. After all, the combination of the flexible and effective population-based algorithm to search for the optimal solution and the convergent characteristics of simulated annealing provides the rationale for developing a hybrid artificial immune algorithm (HAIA) strategy to solve job shops with sequence dependent setup times and flexible machine availability constraints to minimize total tardiness.

Our strategy to hybridize AIA with SA is as follows. After cloning, we apply the SA to the best clone in the

mutating pool, not all the clones because applying the SA to all the clones would result in a very algorithm. In the following subsection, we briefly introduce the simulated annealing that we use.

Simulated Annealing (SA) is a local search-based meta-heuristic to solve combinational optimization problems. SA got its existence from the physical annealing of solid metal. In annealing, a metal is first heated to a high temperature and then cooled down with a very slow rate to the room temperature. The fundamental idea of SAs is to generate a new job sequence s by a random rule from the neighborhood of present sequence x. This new sequence is accepted or rejected by another random rule. A parameter t, called the temperature, controls the acceptance rule. The variation between objective values of two candidate solutions is computed $\Delta C = TT(s) - TT(x)$. If $\Delta C \leq 0$, sequence s is accepted. Otherwise, sequence s is accepted with probability equal to exp (ΔC / ti). The algorithm proceeds by trying a fixed number of neighborhood moves at each temperature ti, while temperature is gradually decreased. The procedure is repeated until a stopping criterion is met. In our algorithm, SA proceeds until in five consecutive temperatures, no improvement is made.

Simulated annealing starts from an initial solution, and a series of moves are made according to a user-define annealing schedule. In this research, the initial solution is one clone in the current mutating pool. The algorithm checks 20 neighbors at temperature ti. The moving operator generates a neighbor (solution) from the current candidate solution by making a slight change in it. These operators must work in such a way that infeasible solutions are avoided. Since many researchers concluded that in SAs, SHIFT operator is superior to other operators like SWAP and INVESION (Naderi et al., 2009a), we generate new solution using SHIFT operator. In Shift operator, one of the job numbers is randomly selected and it is randomly relocated into a new position in sequence. For operation assignment, 10 random assignment is generated and the best is selected. Here, we use exponential cooling schedule, $ti = \alpha .ti\text{-}1$ (where $\alpha \in (0, 1)$ is temperature decrease rate), which is often believed to be an excellent cooling recipe (Schneider et al., 1998). Figure 2 shows the general pseudo code of the proposed SA.

```
Procedure SA

Initialization (the one clone in the mutating pool)
counter = 0
while counter <= 5 do
    for i = 1 to 20 do
        Generate a new neighbor from current solution
        Acceptance criterion
        Update the best solution so far found
    end for
    if the best solution is improved in this temperature do
        counter = 0
    else
        counter = counter + 1
    endif
    Temperature reduction
end while
```

Fig. 2. Pseudo code of the proposed SA

4. Experimental Evaluation

In this section, the performance of the proposed HAIA is evaluated by comparing with RKGA proposed by (Kurz and Askin, 2004), and NEH (Nawaz et al., 1983), artificial immune algorithm (AIA) by (Zandieh et al., 2006). We conduct an experimental evaluation. The algorithms are implemented in Borland C++ and run on a PC with 2.0 GHz Intel Core 2 Duo and 2 GB of RAM memory. We use Relative Percentage Deviation (RPD) as performance measure to compare the methods. When the TT of each algorithm has been obtained for its examples, the best solution obtained for each example (named Minsol) by any of the algorithms is calculated. RPD is obtained by the given formula below (Naderi et al., 2010b):

$$RPD = \frac{Alg_{\text{sol}} - Min_{\text{sol}}}{Min_{\text{sol}}}.100 \qquad (15)$$

where Alg_{sol} is the TT obtained for a given algorithm and instance. Obviously, lower values of RPI are preferred. The stopping criterion is n2×g×1.5 milliseconds computation time for the algorithms. This stopping criterion not only permits for more time as the number of jobs or machines increases, but also is more sensitive toward a rise in number of jobs than number of stages.

4.1. Data generation

We need three sets of instances, one for parameter tuning section, another for general performance evaluation of model and algorithm, and the other one for relative performance evaluation of the algorithm. Data required for a problem consists of the number of jobs (n), range of processing times (pji), number of stages (m) and due dates (dj) a stage for each job. Each stage requires data defining how many machines exist at that stage (mi).

For the first set, we have

$$n = \{20, 60, 100\} \text{ and } m = \{2, 4, 8\}$$

The number of machines at each stage could be a fixed value of 2 machines or uniformly distributed over (1, 4). The processing times are generated by the uniform distribution over range (1, 99). The due dates are generated as follows.

$$d_j = (1 + rand)\left(\sum_{j=1}^{m} p_{j,i}\right)$$

where rand is a random number uniformly distributed over (0, 1). Therefore, there are 18 combinations of n, m, mi and pji. Table 1 shows the problem size factors and their levels.

For the second set, we have

$$n = \{5, 6, 7\} \text{ and } m = \{2, 3\}$$

The number of machines at each stage could be a fixed value of 2 machines or uniformly distributed over (1, 3). The processing times are generated by the uniform distribution

over range (1, 99). Therefore, there are 12 combinations of n, m, mi and pji. One instance is generated in each problem size. Since the sizes are small, the set is called small-sized instances.

For the third set, we consider the levels presented in Table 1 and for each combination we generate 10 instances. Therefore, it sums up to 180 instances. This set is called large-sized instances.

Table 1
Factors and their levels

Factors	Levels
Number of Jobs	{20, 60, 100}
Number of stages	{2, 4, 8}
Machine distribution	{2, U(1, 4)}
Processing Time	U (1, 99)

4.2. Parameter tuning

It is known that the great choice of parameters of an algorithm can influence the performance of that algorithm. Our proposed HAIA has only one parameter, pop-size. The first set of instances is used in this section. We consider the values of $d = \{10, 20, 30, 40\}$. Therefore, we have 4 different HAIAs.

All the instances in the first set are solved by the HAIAs. To analyze the results, we carried out an analysis of variance (ANOVA) and the least significant deference (LSD) at the 95% confidence level (Montgomery, 2000). Figure 3 shows the means plot and LSD intervals. Regarding the results, we concluded that the value of pop-size = 20 is the best value for this parameter.

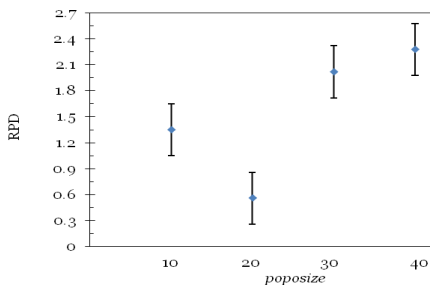

Fig. 3. Means plot and LSD intervals of different levels of pop-size

4.3. Numerical evaluation

In this section, we assess the model and the algorithm. To this end, we first test the capability of the model to solve different problem sizes. Then, the algorithm is compared with the optimal solutions of the model. In the second experiment, we compare the algorithm with some other available algorithms.

4.3.1. Small-sized instances

This section evaluates the general performance of model and algorithm with 12 small-sized instances generated in section 4.1. The model is given a time limit of 2000 sec. Table 2 shows the results of the experiment. In this table, the first three columns present the problem size. The fourth one shows the computational time for the model to solve an instance with the corresponding size and the last one reports the optimality gap of the presented algorithm against the solution of the model.

The MILP model is capable of solving 11 out of 12 instances. Now, we compare HAIA algorithm against the optimal solutions of the MILP model. HAIA optimally solves 10 of 12 instances (83%).

4.3.2. Large-sized instances

We evaluated the algorithms with the 180 large-sized instances generated in section 4.1. Table 3 shows the results of the experiments, averaged for each combination of n and m. HAIA outperforms the other algorithms with RPD of 1.50%. The second best is AIA with RPD of 2.82%. The worst performing algorithms are NEH and RKGA with RPD of 9.21% and 6.83%.

For further precise analysis of the results, we carried out ANOVA. Means plot and LSD intervals at the 95% confidence level for the type of methods factor are shown in Figure 4. As it could be seen, HAIA statistically supersedes the other algorithms. To analyze the possible effects of the number of jobs factor on the algorithms, we computed the performance of the algorithms in the different values of jobs. Figure 5 depicts the interaction between factors type of algorithm and the number of jobs. The proposed HAIA keeps a robust performance on various range of jobs, while NEH outperforms RKGA in the case of n = 100.

Table 2
The results of models evaluation

No.	Problem size			MILP model: Computational time (in sec.)	HAIA (RPD)
	n	m	m_i		
1	5	2	{2,2}	1	0%
2		2	{1,2}	1	0%
3		3	{2,2,2}	2	0%
4		3	{1,3,2}	2	0%
5	6	2	{2,2}	159	0%
6		2	{2,1}	84	0%
7		3	{2,2,2}	110	0%
8		3	{3,3,2}	730	4%
9	7	2	{2,2}	857	0%
10		2	{1,3}	174	0%
11		3	{2,2,2}	241	0%
12		3	{2,2,3}	2000	3%

5. Conclusion and Future Research

In this paper, we investigated hybrid flow shop scheduling problems. Our optimization criterion was total tardiness. First, the problem under consideration was formulated as a mixed integer linear programming model. This model is capable of solving instances up to 10 jobs. Then, an effective immune system algorithm was applied to tackle the problem. In this algorithm, some novel operators based on insertion operators are developed.

Table 3
Average relative percentage deviation for the algorithms grouped by n and m

Instances	Algorithms			
	NEH	RKGA	AIA	HAIA
20×2	10.71	4.81	1.45	1.05
20×4	9.66	5.41	1.67	1.01
20×8	10.12	4.6	1.9	1.09
60×2	9.04	7.23	3.87	1.28
60×4	8.07	6.56	3.21	2.19
60×8	9.58	6.15	3.77	2.01
100×2	7.92	8.17	2.96	1.79
100×4	9.4	8.63	3.35	2.08
100×8	8.41	9.9	3.16	0.99
Average	9.21	6.83	2.82	1.50

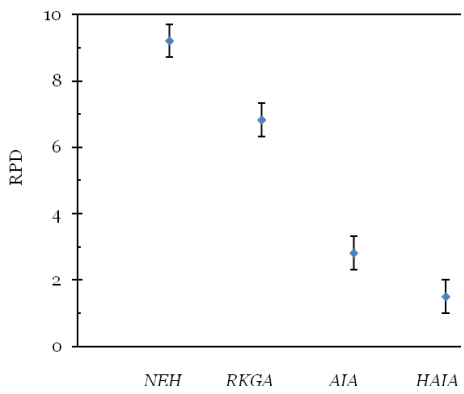

Fig. 4. Means plot and LSD intervals for the type algorithm

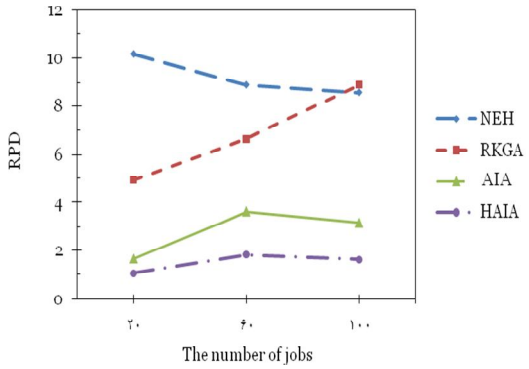

Fig. 5. Means plot for the interaction between factors type algorithm and number of jobs

The algorithm is also enhanced by hybridizing it with a simple local search. To evaluate the performance of the algorithm, we conducted two experiments. We first compared the algorithm with optimal solutions of the model on small-sized instances. Then, we compared the algorithm with some existing algorithms in the literature. The results supported the effectiveness of both the proposed model and the algorithm.

As future research, it could be interesting to work on a multi-population immune system algorithm for the problem and compare its performances with the other algorithms. The evaluation of novel meta-heuristics like electromagnetism-like algorithm for the problem under consideration could also be revealing. Another direction is to apply the immune system algorithm to other scheduling problems like job shops and open shops. The multi-objective case of hybrid flow shop could be considered as an interesting topic for future research.

6. References

[1] Allahverdi A., C. T. Ng, Cheng, T. C. E. and Kovalyov, Y. M., (2008). A survey of scheduling problems with setup times or costs. European Journal of Operational Research. 187(3): 985–1032.

[2] Behnamian, J. and Fatemi Ghomi, S. M. T., (2011). Hybrid flow shop scheduling with machine and resource-dependent processing times, Applied Mathematical Modelling, 35(3), 1107-1123.

[3] Heinonen J. and Pettersson, F., (2007). Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. Applied Mathematics and Computation, 187, 989–998.

[4] Kurz M. E. and Askin, R. G., (2003). Comparing scheduling rules for flexible flow lines. International Journal of Production Economics, 85: 371–388.

[5] Kurz M. E. and Askin, R. G., (2004). Scheduling flexible flow lines with sequence-dependent setup times. European Journal of Operational Research. 159(1): 66–82.

[6] Jin Z., Yang, Z. and Ito, T., (2006). Metaheuristic algorithms for the multistage hybrid flow shop scheduling problem. International Journal of Production Economics. 100: 322–334.

[7] Goldberg, D. E., (1989). Genetic algorithms in search, optimization and machine learning. Reading. USA: Addison-Wesley.

[8] Matta, M. E., (2009). A genetic algorithm for the proportionate multiprocessor open shop. Computers and Operations Research, 36, 2601–2618.

[9] Naderi, Ruiz, B. R., Zandieh, M., (2010a). Algorithms for a realistic variant of flow shop scheduling Computers & Operations Research, 37, 236 – 246.

[10] Naderi, B., Zandieh, M. and Roshanaei, V., (2009a). Scheduling hybrid flow shops with sequence dependent setup times to minimize makespan and maximum tardiness. International Journal of Advanced Manufacturing Technology, 41, 1186–1198.

[11] Naderi B., Khalili, M., Tavakkoli-Moghaddam, R., (2009b). A hybrid artificial immune algorithm for a realistic variant of job shops to minimize the total completion time. Computers and Industrial Engineering, 56, 1494–1501.

[12] Naderi, B. and Ruiz, R., (2010). The distributed permutation flow shop scheduling problem, Computers and Operations Research, 37, 754–768.

[13] Naderi, B., FatemiGhomi, S. M. T., Aminnayeri, M., Zandieh, M., (2010b). A contribution and new heuristics for open shop scheduling. Computers and Operations Research, 37, 213–221.

[14] Nawaz, M., Enscore Jr, E. E. and Ham, I., (1983). A heuristic algorithm for the m-machine, *n*-job flow shop sequencing problem, Omega, International Journal of Management Sciences. 11(1): 91–95.

[15] Schneider, J., Morgensten, I. and Singer, J. M., (1998). Bouncing towards the optimum: Improving the results of Mento Carlo optimization algorithms. Physical Review E, 58(4), 5085–5095.

[16] Pan, C. H., (2007). A study of integer programming formulations for scheduling problems. International Journal of Systems Sciences, 28(1), 33–41.

[17] Prakash, A., Khilwani, N., Tiwari, M. K. and Cohen, Y., (2008). Modified immune algorithm for job selection and operation allocation problem in flexible manufacturing systems. Advances in Engineering Software, 39, 219–232.

[18] Pinedo, M. L., (2008). Scheduling: Theory, Algorithms, and Systems. 3th edition, Springer Science+Business Media, New York.

[19] Ruiz R. and Allahverdi, A., (2007). Some effective heuristics for no-wait flow shops with setup times to minimize total completion time. Annals of Operations Research. 156: 143-171.

[20] Zandieh, M., Fatemi Ghomi, S. M. T. and Moattar Husseini, S. M., (2006). An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. Applied Mathematics and Computations. 180: 111–127.

[21] Zhang, C. Y., Li, P., Rao, Y. and Guan, Z., (2008). A very fast TS/SA algorithm for the job shop scheduling problem, Computers and Operations Research, 35, 282–294.