An Anarchic Society Optimization Algorithm for a Flow-Shop Scheduling Problem with Multiple Transporters between Successive Machines

Amir Ahmadi-Javid and Pedram Hooshangi-Tabrizi
Department of Industrial Engineering
Amirkabir University of Technology
Tehran, Iran

Abstract

This paper addresses a *Permutation Flow-shop Scheduling Problem* (PFSP) where there is a finite number of transporters between any two successive machines to carry jobs from one machine to the subsequent machine. In order to solve the problem, an effective *Anarchic Society Optimization* (ASO) algorithm is developed to find a sequence of jobs with nearly-minimum makespan. The proposed ASO algorithm is compared with a *Particle Swarm Optimization* (PSO) algorithm on a set of benchmark test problems. The results show that the ASO algorithm considerably outperforms the PSO algorithm. Finally, a sensitivity analysis is carried out to examine the impact of the number of transporters on the performance of the manufacturing system.

Keywords
Anarchic Society Optimization (ASO), Permutation Flow-Shop Scheduling Problem (PFSP); Particle Swarm Optimization (PSO); Transportation

1. Introduction
In today's competitive world, effective scheduling is one of the necessities of survival in the market. Scheduling of operations plays an important role in the manufacturing and service industries, especially in the production, transportation, distribution, processing of information and communication. In general, scheduling problems determine the priority of activities to achieve the specific goals considering some requirements and limitations. One of the standard assumptions of almost all scheduling problems is that the time needed for moving each job from one machine to another machine is negligible. Although this assumption is often justified, there are many practical situations that it has no longer held because transportation times are significantly large and there are a limited number of transporters. In fact, in most manufacturing and distribution systems, semi-finished jobs will be transferred from a processing facility to other ones by a number of material handling transporters including *Automated Guided Vehicles* (AGVs), robots and conveyor belts.

In the literature, there are several papers that consider simultaneous scheduling of jobs and material handling equipment like AGVs (Raman et al. 1986, Langston 1987, Anwar et al. 1998, Jawahar et al. 1998, Khayat et al. 2006, Kumar et al. 2011). Kise (1991) proved that the makespan minimization in a flow-shop scheduling problem with two machines and constant transportation times and one transporter is an NP-Hard problem. Several authors (Bilge and Ulusoy 1995, Ulusoy et al. 1997, Naderi et al. 2009) studied an off-line integrated production and material handling scheduling problem with the objective of minimizing the makespan in a flexible manufacturing system under various assumptions. Other papers that consider related problems in job-shop or flow-shop problems are (Hurink and Knust 2001, Hurink and Knust 2002, Hurink and Knust 2005, Boudhar and Haned 2009). This paper studies a permutation flow-shop considering the transportation times where there are a finite number of transporters between any two successive machines. This generalizes the problem considered in Naderi et al. (2010) where there is either one or infinite number of machines between any two successive machines.

The remainder of the paper is organized as follows. Section 2 presents a formal description of the problem. Section 3 establishes algorithms based on Anarchic Society Optimization (ASO) and Particle Swarm Optimization

(PSO) to solve the proposed problem. Section 4 presents numerical study, and Section 5 concludes the paper.

2. Problem description

A *Permutation Flow-shop Scheduling Problem* (PFSP) consists of scheduling of *n* jobs with given processing times on *m* machines where all jobs have the same processing routes on machines and the sequences of processing jobs on all machines are identical. To be more precise, in a PSFP, a set of *n* jobs, $J = \{J_1, J_2, ..., J_n\}$, must be done on a set of *m* machines, $M = \{1, 2, ..., m\}$. Each job J_i requires a set of operations, $O_i = \{O_{i,1}, O_{i,2}, ..., O_{i,m}\}$, to complete its manufacturing process. Operation $O_{i,j}$ must be processed on machine *j* and incurs $P_{i,j}$ units of time. A job can be started on machine *j* if its process on machine *j*-1 is finished, and machine *j* is free. If a job is in the *k*th position in a sequence on the first machine, then it will be in the *k*th position on all the other machines. Moreover, we consider the following assumptions:

- A set of *n* jobs $J = \{J_1, J_2, ..., J_n\}$ is available at time zero,
- Each job needs *m* (number of machines) different operations and each operation is only performed on a specific machine,
- Machines have sufficient buffers for processing all jobs,
- At a time, each job can be processed only on one machine, and each machine can process only one job,
- For each operation, the setup time is independent of the job sequence, and it is included in the processing time of the job,
- Characteristics of jobs are specific and the jobs are predefined,
- All processing times are assumed to be deterministic,
- Interruption of operations due to preemption is not allowed,
- The transportation times of jobs between machines are not dispensable,
- Empty-moving times for transporters are taken into account,
- There are a finite number of transporters between any two successive machines that carry jobs from one to the other.

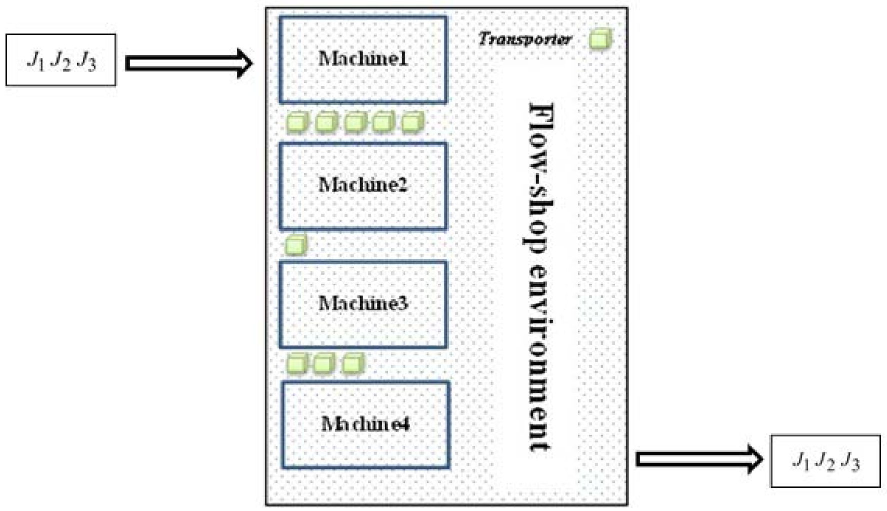

Figure 1: Layout of a flow-shop environment with transporters

The main feature of the proposed PFSP is that here, transportation times between machines are considered where a finite number of transporters exist between any two successive machines to move semi-finished jobs. Figure 1 shows that the layout of a flow-shop environment.

In this environment, there are 3 jobs that must be processed on 4 machines with the order $J_1\, J_2\, J_3$. There are 5, 1 and 3 transporters between machines 1 and 2, machines 2 and 3, and machines 3 and 4, respectively. Figure 2 depicts the Gantt chart for specific transportation and processing times. The makespan of this example is the completion time of the last job J_3 on the last machine M_4.

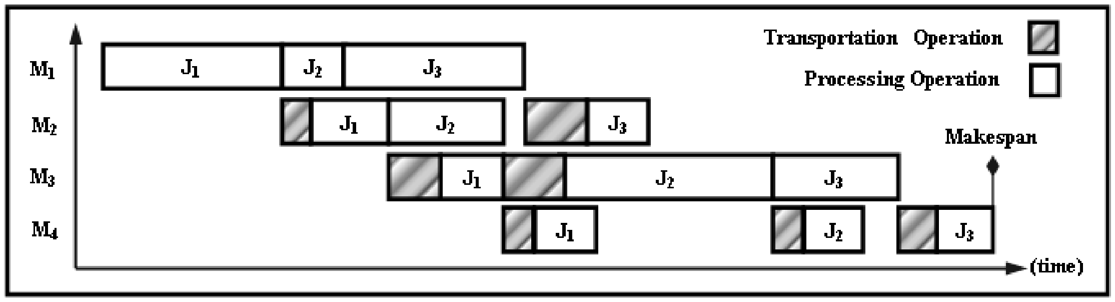

Figure 2: Gantt chart for the example given in Figure 1

3. Problem description

To solve the PFSP explained in the previous section, we design two heuristic algorithms based on Anarchic Society Optimization (ASO) and Particle Swarm Optimization (PSO). The following subsections briefly explain these algorithms.

3.1 Anarchic Society Optimization (ASO)

The algorithm proposed in this section is based on ASO that is a human-inspired optimization method introduced by Ahmadi-Javid (2011). ASO is inspired by a human society whose members behave anarchically and adventurously to find much better situations. The members become more nervous and greedier as the differences among people intensify. Using such members, ASO explores the solution space perfectly and avoids falling into local optimums. The mathematical description of ASO is given in the following.

Given the problem of minimizing a function $f(x)$ over the set $\Omega \subset R^d$, an ASO algorithm tries to solve the problem by using a society of members exploring the solution space Ω to seek the global optimal solution. In the *k*th iteration of the algorithm, each member of society *i* (*i* = 1, 2,…, N) has three characteristics as three *d*-dimensional vectors:

- $Sit_i(k)$: The situation of the *i*th member,
- $Dir_i(k)$: The movement direction associated with the selected movement of the *i*th member,
- $Best_i(k)$: The best personal previously experienced situation,

and the society has a characteristic expressed as a *d*-dimensional vector:

- $GBest(k)$: The situation of the best member in the society.

Steps of the proposed ASO algorithm are presented below in a nutshell:

Step 1: Initialize the members' situations and movement directions, $Sit_i(1), Dir_i(1)$, randomly, and compute the objective functions for all society members.

Step 2: For member *i* in iteration *k*:

- Define the fickleness index $FI^k(i)$, and then determine a movement policy $MP^k_{Current}(i)$ based on vectors $Sit_i(k), Dir_i(k-1)$, and the fickleness index,
- Define the external irregularity index $EI^k(i)$, and then determine a movement policy $MP^k_{Society}(i)$ based on the other society members' situations $Sit_j(k),. j \neq i,$ and the external irregularity index,
- Define the internal irregularity index $II^k(i)$, and then determine a movement policy $MP^k_{Past}(i)$ based on the member's past situations $Best_i(1),..., Best_i(k-1)$ and the internal irregularity index,
- Determine a movement policy based on the three movement policies $MP^k_{Current}(i)$, $MP^k_{Society}(i)$ and $MP^k_{Past}(i)$.
- Update the current situation of member *i* by the determined movement policy.

Step 3: If the termination condition is met, then stop; otherwise, repeat steps (2)–(3).

The elements of the proposed ASO including movement policies and associated parameters are described in the following. For more technical details we refer to Ahmadi-Javid (2011).

- **Movement policy based on current situation:**

For member *i*, the fickleness index $FI^k(i)$ is defined as

$$FI^k(i) = \frac{f(Sit_i(k)) - f(Gbest(k))}{f(Gbest(k))},$$

where the objective function $f(y)$ is the makespan of the decoded solution associated with the real vector $y \in R^d$ (see Figure 3) and the movement policy $MP^k_{Current}(i)$ as follows:

$$MP^k_{Current}(i) = \begin{cases} \text{Move in a } \varepsilon-\text{neighborhood of its past movement direction} & FI^k(i) \leq \alpha \\ \text{Move toward the situation of a randomly selected member (say } j\text{)} & \text{otherwise} \end{cases}$$

where α is a given threshold. The direction associated with this movement policy is

$$Dir_i(k) = \begin{cases} D & FI^k(i) \leq \alpha \\ Sit_j(k) - Sit_i(k) & \text{otherwise} \end{cases}$$

where ε is a given positive number and D is a vector uniformly taken from the box $[(1-\varepsilon) \times Dir_i(k-1), (1+\varepsilon) \times Dir_i(k-1)]$.

- **Movement policy based on the other society members' situations:**

For member *i*, we define the external irregularity index $EI^k(i)$ as follows:

$$EI^k(i) = \frac{\underset{i}{Max}\{f(Sit_i(k))\} - f(Gbest(k))}{f(Gbest(k))},$$

and the movement policy $MP^k_{Society}(i)$ as follows:

$$MP^k_{Society}(i) = \begin{cases} \text{Move toward Gbest member's situation} & EI^k(i) \leq \beta \\ \text{Move toward the situation of a randomly selected member (say } j\text{)} & \text{otherwise} \end{cases}$$

where β is a given threshold. The direction associated with this movement policy is

$$Dir_i(k) = \begin{cases} R \times (Gbest(k) - Sit_i(k)) & EI^k(i) \le \beta \\ Sit_j(k) - Sit_i(k) & \text{otherwise} \end{cases}$$

where *R* is a uniform random number in the interval (0, 1).

```
Take permutation $x$
1- $Ct_j = 0$    $j = 1,2,3,...,n$
2- $Ctm_i = 0$    $i = 1,2,3,...,m$
3- $Ctt_{i,k} = 0$    $i = 1,2,3,...,m$    $k = 1,2,3,...,k_i$
4- For $i = 1$ to $m$ do
5-     For $j = 1$ to $n$ do
6-     $j$ = Job in the $j^{\text{th}}$ position of permutation $x$
7-         If $(i \ne 1)$
8-             $k = \arg\min_l \{Ctt_{i,l}\}$
9-             If $(Ctt_{i,k} > Ct_j)$
10-                $Ct_j = Ctt_{i,k}$
11-            Else
12-                $Ctt_{i,k} = Ct_j$
13-            End If
14-                $Ctt_{i,k} = Ct_j + T_{j,i}$ + empty moving time
15-                $Ct_j = Ct_j + T_{j,i}$
16-        End If
17-        If $(Ctm_i > Ct_j)$
18-            $Ctm_i = Ctm_i + P_{j,i}$
19-        Else
20-            $Ctm_i = Ct_j + P_{j,i}$
21-        End If
22-        $Ct_j = Ctm_i$
23-    End For
24- End For
25- Find $C_{\max}$ = makespan
```

Figure 3: Procedure of calculating makespan of a decoded solution *x* in ASO and PSO algorithms

- **Movement policy based on the member's past situations:**

For member *i*, the internal irregularity index $II^k(i)$ and $MP^k_{Society}(i)$ are defined as follows:

$$II^k(i) = \frac{f(Best_i(k)) - f(Gbest(k))}{f(Gbest(k))}$$

$$MP^{k}_{Past}(i) = \begin{cases} \text{Move toward best personal situation experienced by member } (Best_i) & II^{k}(i) \le \gamma \\ \text{Move toward the situation of a randomly selected member (say } j\text{)} & \text{otherwise} \end{cases}$$

where γ is a given threshold. The direction associated with this movement policy is

$$Dir_i(k) = \begin{cases} R \times (Best_i(k) - Sit_i(k)) & II^{k}(i) \le \gamma \\ Sit_j(k) - Sit_i(k) & \text{otherwise} \end{cases}$$

where R is a uniform random number in the interval (0, 1).

After determining the three movement policies, moving to a new situation is done by updating $Sit_i(k)$ as follows:

$$Sit_i(k+1) = Sit_i(k) + Dir_i(k)$$

where $Dir_i(k)$ is the direction associated with the final movement policy obtained based on three movements policies $MP^{k}_{Current}(i)$, $MP^{k}_{Society}(i)$ and $MP^{k}_{Past}(i)$. The elitism, sequential and other combination rules (see Ahmadi-Javid [1]) can be used to obtain the final movement policy based on these movement policies.

3.2 Particle Swarm Optimization (PSO)

PSO is an evolutionary algorithm introduced by Kennedy and Eberhart (1995). The PSO algorithm is based on the social behavior of animals, such as bird flocking or fish schooling. The algorithm works by initializing swarm of particles like bird or fish randomly over the search space**.** Each particle has two characteristics: velocity and position. These particles move with a certain velocity and try to find the global best position after some iterations. Each particle iteratively adjusts its velocity vector, based on its tendency toward best personal previously visited position *pbest* and tendency toward the best position of its neighbors *gbest*, then compute a new position that the particle decide to fly there. Supposing $Pos_i(k)$ is the position vector and $Vel_i(k)$ is the velocity vector of particle i, a PSO algorithm updates the vectors as follows:

$$Vel_i(k+1) = wVel_i(k) + c_1 r_1 (pbest_i(k) - Pos_i(k)) + c_2 r_2 (gbest(k) - Pos_i(k))$$

$$Pos_i(k+1) = Pos_i(k) + Vel_i(k+1)$$

where k denotes the iteration number. Two constants c_1 and c_2 are called cognitive and social parameters, respectively, r_1 and r_2 are two random numbers in interval (0, 1) and w is the inertia weight factor. Ahmadi-Javid [1] showed that PSO is a special case of ASO framework.

For both proposed ASO and PSO algorithms, Figure 3 shows the pseudo code used to compute the makespan of the decoded solution x which is a permutation vector whose elements are the rank of corresponding element in the vector $Sit_i(k)$ for the ASO algorithm, and $Pos_i(k)$ for the PSO algorithm.

4. Numerical Study

In this section, we study the performance of the proposed ASO and PSO algorithms. The algorithms are coded in C++ and run with a PC with 2.66 GHz Intel Core i5 and 4 GB of RAM memory under a Windows operating system. Both algorithms are tested ten times on 57 instances including 45 small-sized and medium-sized and 12 large-sized instances which are considered in Taillard's benchmark (Taillard 1993). For each instance, we run both algorithms and then compute the *Relative Percentage Deviation* (*RPD*) which is computed as follows:

$$RPD = \frac{Alg - Best}{Best}$$

where *Alg* is the objective value of the solution obtained by the algorithm, and *Best* is the best objective value obtained by the ASO and PSO algorithms in ten runs. The processing times are taken from Taillard's instances, which are uniformly distributed over (1, 99), and the transportation times are uniformly taken from (1, 35). The empty-moving times equal 6 units of time, and the numbers of transporters between each two successive machines are uniformly distributed over the set {1,…, n}.

The parameters of the ASO algorithm are set as follows $\alpha = 5\%, \beta = 15\%, \gamma = 2.5\%$ and $\varepsilon = 10\%$. The blended crossover (BLX-α) (Settles and Soule 2005) is used for the Gbest member in the ASO algorithm. The elitism rule is considered as the combination rule. The parameters of the PSO algorithm are set as follows $c_1 = 2, c_2 = 2, w = 1, Pos_{\max} = 4, Pos_{\min} = 0$ and $Vel_{\max} = 4$. Both algorithms are terminated after 400 iterations.

Average RPDs for both algorithms are plotted in Figure 4. This figure shows that the ASO algorithm is significantly effective than the PSO algorithm in almost all instances.

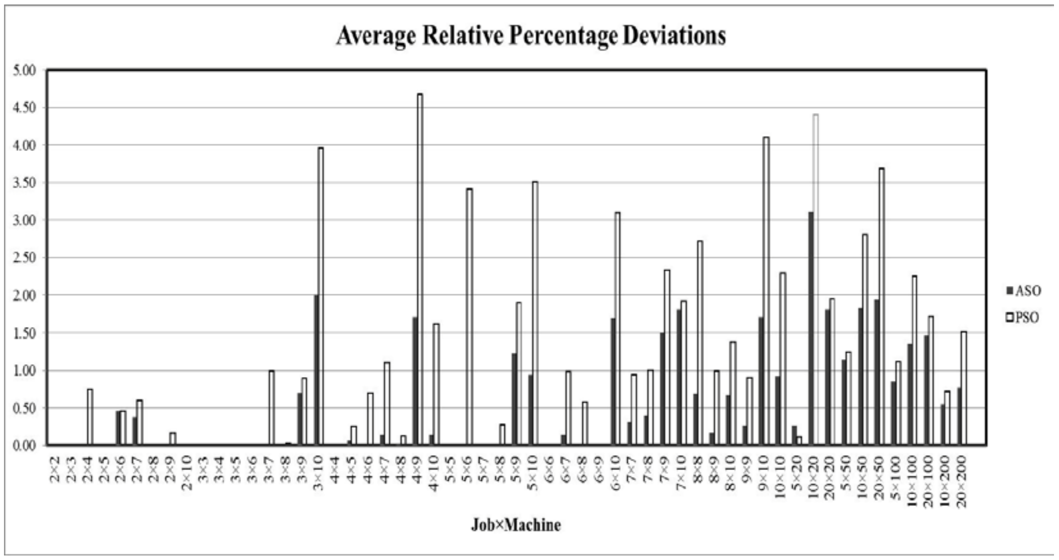

Figure 4: Comparison of proposed ASO and PSO algorithms

We now carry out a sensitivity analysis for an instance of the PFSP with 4 machines and 8 jobs, m = 4, n = 8, to assess the impact of the numbers of transports on the improvement of makesapn. Consider the coefficient C:

$$C = \frac{MT(k) - MT(8)}{MT(8)} \times 100$$

where $MT(k)$ is the makespan obtained by the ASO algorithm when there are k transporters between any two successive machines. As the coefficient C tends to zero, the performance of the manufacturing system increases. For $k = 8$ we have $C = 0$, since $MT(8)$ is the best makespan for this problem considering that we have only 8 jobs in this example.

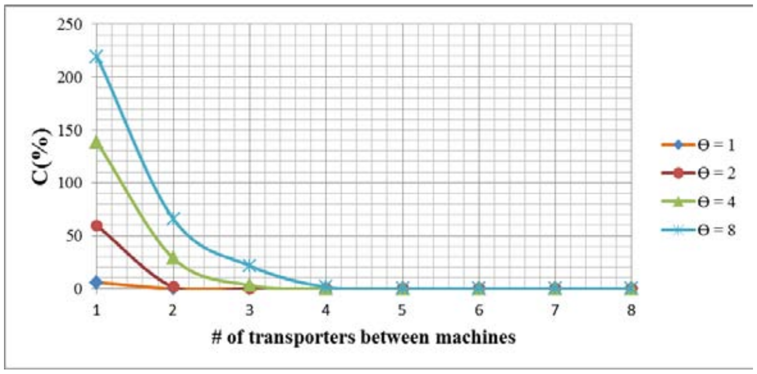

Figure 5: Impact of number of transporters on performance of a manufacturing system with four machines and eight jobs

Assume that the processing times are taken uniformly from (1, 99), and the transportation and return times are taken uniformly from (1, θ×50) and (1, θ×25), respectively, for θ = 1, 2, 4, 8. Figure 5, depicts C for different values of θ = 1, 2, 4, 8 and k = 1, 2,…, 8. From this figure, one can see that, for example, for θ = 2, having two transporters between any two successive machines is the same as the case of having eight transporters. Even for the extreme case of θ = 8, where transportation times are much greater than processing times, the performance of the manufacturing system with four transporters between each two successive machines is the same as the case of eight transporters. This analysis indicates that by exploiting a few of transporters between any two successive machines we can approximately achieve the full performance, that is, $C \approx 0$.

5. Concluding Remarks

This paper deals with a permutation flow-shop scheduling problem with a finite number of transporters between any two successive machines. The objective is to minimize the makespan, the maximum completion time of jobs. To solve large-sized instances of the problem, an effective Anarchic Society Optimization (ASO) algorithm is developed. The ASO algorithm is compared with a Particle Swarm Optimization (PSO) algorithm. The results show that the ASO algorithm significantly outperforms the PSO algorithm on a set of benchmark test problems. Lastly, a sensitivity analysis is carried out to assess the effect of the number of transporters between successive machines on the makespan. This shows that a small number of transporters, compared to the number of jobs, are sufficient to nearly achieve the best possible makesapn.

Future research needs to examine more complicated systems such as job-shop environment with different transportation systems. Integration of scheduling and designing the transportation system is another fruitful but challenging avenue for research study.

References

Ahmadi-Javid, A., Anarchic society optimization: A Human-inspired method, *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, June 5-8, New Orleans, LA, pp. 2586-2592, 2011.

Anwar, M.F., Nagi, R., Integrated scheduling of material handling and manufacturing activities for just-in-time production of complex assemblies, *International Journal of Production Research*, vol. 36, no. 3, pp 653-681, 1998.

Bilge, U., Ulusoy, G., A time window approach to simultaneous scheduling of machines and material handling system in an FMS, *Operations Research*, vol. 43, no. 6, pp. 1058-1070, 1995.

Boudhar, M., Haned, A., Preemptive scheduling in the presence of transportation times, *Computers & Operations Research*, vol. 36, no. 8, pp. 2387-2393, 2009.

Hurink, J., Knust, S., Makespan minimization for flowshop problems with transportation times and a single robot, *Discrete Applied Mathematics*, vol. 112, no. 1-3, pp. 199-216, 2001.

Hurink, J., Knust, S., A Tabu Search Algorithm for Scheduling a Single Robot in a Job-shop Environment, *Discrete Applied Mathematics*, vol. 119, no. 1-2, pp. 181-203, 2002.

Hurink, J., Knust, S., Tabu search algorithms for job-shop problems with a single transport robot, *European Journal of Operation Research*, vol. 162, no. 1, pp. 99-111, 2005.

Jawahar, N., Aravindan, P., Ponnambalam, S.G., and Suresh, R.K., AGV schedule integrated with production in flexible manufacturing systems, *International Journal of Advanced Manufacturing Technology*, vol. 14, no. 6, pp. 428-440, 1998.

Kennedy, J., Eberhart, R.C., Particle swarm optimization, *Proceedings of IEEE International Conference on Neural Networks*. *Piscataway: IEEE Service Center,* November/December, Perth, Australia, vol. 4, pp. 1942–1948, 1995.

Khayat, G.E., Langevin, A., and Riopel, D., Integrated production and material handling scheduling using mathematical programming and constraint programming, *European Journal of Operation Research*, vol 175, no. 3, pp 1818-1832, 2006.

Kise, H., On an automated two-machine flowshop scheduling problem with infinite buffer, *Journal of the Operations Research Society of Japan*, vol. 34, no. 3, pp. 354-361, 1991.

Kumar, M.V.S., Janardhana, R., and Rao, C., Simultaneous scheduling of machines and vehicles in an FMS environment with alternative routing, *The International Journal of Advanced Manufacturing Technology*, vol. 53, no. 1, pp.. 339-351, 2011.

Langston, M.A., Interstage transportation planning in the deterministic flow-shop environment, *Operations* Research, vol. 35, no. 4, pp. 556-564, 1987.

Settles, M., Soule, T., Breeding Swarms: A GA/PSO Hybrid, *GECCO'05*, June 25–29, Washington, DC, USA, 2005.

Naderi, B., Ahmadi-Javid, A., and Jolai, F., Permutation flowshops with transportation times: mathematical models and solution methods, *International Journal of Advanced Manufacturing Technology*, vol. 46, no. 5-8, pp. 631-647, 2010.

Naderi, B., Zandieh, M., Khaleghi Ghoshe Balagh, A., and Roshanaei, A., An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness, E*xpert Systems with Applications*, vol. 36, no. 6, pp. 9625-9633, 2009.

Raman, N., Talbot, F.B., and Rachamadugu, R.V., Simultaneous scheduling of machines and material handling devices in automated manufacturing, *Proceedings of 2nd ORSA/TIMS Conference on FMS: OR Models and Applications*. Elsevier Science Publishers B.V, Amsterdam, pp. 321-332, 1986.

Taillard, E., benchmarks for basic scheduling problems, *European Journal of Operational Research*, vol. 64, no. 2, pp. 278-285, 1993.

Ulusoy, G., Sivrikaya-Serifoglu, F. and Bilge, Ü., A genetic algorithm approach to the simultaneous scheduling of machines and automated guided vehicles, *Computers & Operations Research*, vol. 24, no. 4, pp. 335-351, 1997.