A New Hybrid Genetic Algorithm to Deal with the Flow Shop Scheduling Problem for Makespan Minimization

Fatima Boumediene, Yamina Houbad, Ahmed Hassam, Latéfa Ghomri

A new Hybrid Genetic Algorithm to deal with the Flow Shop Scheduling Problem for makespan minimization

Abstract. In the last years, many hybrid metaheuristics and heuristics combine one or more algorithmic ideas from different metaheuristics or even other techniques. This paper addresses the hybridization of a primitive ant colony algorithm inspired from the Pachycondyla apicalis behavior to search prey with the Genetic Algorithm to find near optimal solutions to solve the Flow Shop Scheduling Problem with makespan minimization. The developed algorithm is applied on different flow shop examples with diverse number of jobs. A sensitivity analysis was performed to define a good parameter choice for both the hybrid metaheuristic and the classical Genetic Algorithm. Computational results are given and show that the developed metaheuristic yields to a good quality solutions.

Keywords: Scheduling, Flow Shop, Pachycondyla apicalis algorithm , hybridization, optimization, metaheuristic.

1 Introduction

Flow Shop Scheduling Problem (FSSP) can be found in many real world problems because of the multitude of its application and its strong industrial background. A flow shop is a processing system characterized by a unidirectional flow of jobs being processed sequentially. Only small scale problems of Flow Shop Scheduling Problem can be solved by exact techniques, while others are classified NP-Hard problems and their resolution require dedicated methods to solve them in a reasonable computational time. Metaheuristics are among search techniques that offer near optimal solutions in generally low computational time. A metaheuristic can be seen as a search technique which can be applied to a wide set of different optimization problems with relatively few modifications to make them adapted to a specific problem. Genetic Algorithm [1], Ant Colony Optimization (ACO) [2, 3], Particle Swarm Optimization (PSO) [4], Artificial Bee Colony (ABC) [5, 6], Firefly Algorithm (FA) [7, 8, 9], Cuckoo-Search (CS) [10], just to name a few, are nature inspired metaheuristics and were largely used to solve many combinatorial optimization problems such as scheduling problems [11, 12, 13].

Genetic Algorithm is among the well known metaheuristics and has been widely used to solve combinatorial optimization problems such as Flow Shop Scheduling Problems and performed well. The pioneering research on ants inspired metaheuristics was the Ant Colony Optimization (ACO) algorithm done by Dorigo (1992) [14], and an introduction to the ACO algorithm has been provided by Dorigo,

Maniezzo and Colorni (1996) [15]. The ACO has been used to solve many combinatorial optimization problems ([16, 17, 18, and 19]).

An API algorithm is another ants inspired metaheuristic. It is a population-based, cooperative search procedure derived from the Pachycondyla apicalis foraging behavior ants, which are primitive ants that have been studied in the Mexican tropical forest near the Guatemala border [20]. It makes use of ant's colony, which iteratively construct and explore solutions to a combinatorial optimization problem.
Since Monmarché works ([20], [21]) the API algorithm has been used to solve different optimization problems ([22], [23], [24]).

The Pachycondyla apicalis are primitive ants living in colonies comprise from 20 to 100 individuals and use particular procedure to search their preys. The procedure begins by creating a number of hunting sites distributed uniformly around their nest at approximately 10 meters. Starting from the nest, they cover globally a given surface and partition it into many hunting sites and each ant has to perform an exploration of given sites. Because of prey impoverishment or when the nest becomes less comfortable overtime, ants change the nest location periodically.

Using the Pachycondyla apicalis strategy to find their preys an algorithm named the API algorithm has been developed [20]. The API algorithm is based on many parallel local searches collaborating to find a near optimal solution for an objective function.

An effective search technique must not focus the search on only one region of the solution space. It needs to enable a balance between intensification and diversification. In last years, many researches have produced a large number of algorithms that do not fit a single metaheuristic, but they combine various algorithmic ideas, generally originating from artificial intelligence, computer science and operations research. However, to obtain better results, researchers have recently focused on the use of hybrid metaheuristics.

Hybrid metaheuristics are high level procedures that coordinate, generally, two or more metaheuristics to find solutions that are of better quality than those found by only one of them. Hybrid genetic algorithm (HGA), are proposed to enhance the performance of original genetic algorithm.

The present paper deals with the Flow Shop Scheduling Problem with makespan minimization using a hybrid metaheuristic. We propose the adaptation of the hybrid genetic algorithm (HGA) to solve scheduling problem in a flow shop with Cmax minimization with a sensitivity analysis.

The remainder of this paper is organized as follows: section 2 describes the flow shop problem formulation, section 3 illustrate the GA principle, section 4 gives the API and the API Fourragement algorithms principles, section 5 describes the Hybrid Genetic Algorithm developed, section 6 gives the sensitivity analysis of the developed algorithms and discusses obtained results and section 7 draws some conclusions and discusses future works.

2 Problem description

In many manufacturing systems, a number of operations have to be done on all jobs to be processed. When these operations have to be done in the same order and assuming that all machines are set in series the environment is referred to a flow shop.

A Flow Shop Scheduling Problem which can be defined as follows:

There is a number of n independent jobs $(J_1, \ldots, J_n)$ to be processed on m different machines $(m_1, \ldots, m_m)$. Each job J_i $(i=1, \ldots, n)$ has to be processed on all m_j $(j=1, \ldots, m)$ machines for P_{ij} time units (processing time) and the processing of any job can begin only after the completion of the preceding job.

As the problem size increases, exact methods are not computationally practical. For this reason, many researchers have focused on developing heuristics and metaheuristics for NP-Hard problems.

FSSP with makespan minimization can be stated as the problem of finding the minimal processing time such that all the jobs are processed in all the machines.

The makespan can be defined as the time between the beginning of the execution of the first job on the first machine and the completion of the execution of the last job on the last machine.

The assumptions and constraints made for the Flow Shop Scheduling Problem are summarized as:

- There are no precedence constraints among tasks of different jobs;
- Each machine mj can process only one job J_i at a time;
- Each job J_i can be performed only on one machine m_j at a time;
- For all jobs the processing time on each machine is previously known and deterministic;
- The machines sequence is the same for all the jobs, the problem is to find the job sequences on the machines which minimize the makespan ;
- Traveling time between consecutive machines is negligible.

The following notations are needed to define the mathematical formulation of the problem:

n: number of jobs
m: number of machines
S_{ij}: starting time of job i in machine j
P_{ij}: processing time of job i in machine j
C_{ij}: completion time of job i in machine j
C_{max}: completion time of the last job in the last machine

The problem can now be formulated as:

$$f=minC_{max} \quad (1)$$

$$Cm_{ax}=max\{ C_{ij} \} \text{ for all } i=1,\ldots,n;\ j=1,\ldots,m \quad (2)$$

$$C_{ij} = S_{ij} + P_{ij} \qquad \text{for all } i=1,\ldots,n;\ j=1,\ldots,m \quad (3)$$

In our study we consider a 30 machines system with different number of jobs (20, 30 and 50 jobs), that makes we have three different classes of systems.

3 The GA principle

GA explores a problem space with a population of chromosomes (individuals) and selects chromosomes to be explored iteratively based on their performance. Holland (1975) [1] presented a basic GA in his studies described as follows:

```
Generate initial population randomly
Calculate the fitness value of chromosomes
While termination condition not satisfied
      Process crossover and mutation at chromosomes
      Calculate the fitness value of chromosomes
      Select the offspring to next generation
```

In this basic form of the genetic algorithm three operators are used to create offspring: selection, crossover and mutation.

- Selection: selects chromosomes (solutions) in the population for reproduction.
- Crossover: The crossover operator is applied on each selected parent and uses usually one cut point and subsequences before and after that point are exchanges between each two selected chromosomes named parents.
- Mutation: The mutation operator is used to diversify the population in the genetic algorithm and prevent the premature convergence of the algorithm. The mutation operator exchanges some genes chosen randomly in the considered chromosome.

4 The Pachycondyla apicalis based metaheuristic (API)

The API algorithm can be viewed as an optimization algorithm performing parallel searches to find best solutions starting from an initial solution (named the nest) and varying its initial solution iteratively.

The API algorithm can be given as follows:

(1) Choose randomly the initial location of the nest *N*

(2) For each ant a_i, i= 1,2, …, $ants_{nbr}$

Perform the API-Fourragement

(3) If P_N is reached Then Change the nest location and Reset the memories of all ants

(4) Go to (2) or Stop if a stopping criterion is satisfied.

The API-Fourragement is given as follows:

If a_i has less than p hunting sites in memory Then Create a new hunting site in the neighborhood of *N* ants Explore this created site

Else

If the previous site exploration was successful Then Explore again the same site

Else Explore a randomly selected site (among the p sites in memory)

Remove from the ants memories all sites which have been explored unsuccessfully more than $P_{local}(a_i)$ consecutive times

Parameters setting

In this section we present the API algorithm parameters and the signification of each one in our adaptation.

- a_i: Ant
- $ants_{nbr}$: the number of ants used to explore the search space.
- N: The nest. This corresponds, in optimization, to the initial solution of the problem. The random choice of the nest location means a random generation of an initial solution (jobs sequence) of the problem. Hunting sites are also problem solutions initially generated from the nest.
- p: Number of hunting sites in ants memories
- A hunting site: Represents a possible solution of the problem. The number of hunting sites is the number of solutions to be explored by an ant.
- Site exploration: An exploration of a site is to find a new solution in the neighborhood of the considered solution. Then the objective function is evaluated. The exploration will give a new hunting site.
- P_{local}: Represents a parameter which calculates the number of consecutive failures encountered on the same site (this corresponds to calculate the number of unsuccessful explorations of the same solution).
- P_N: Represents the parameter by which the nest changes location. In optimization, this change is equivalent to a reset of the initial condition in the algorithm. P_N is given as follows:

$$P_N = 2\times(P_{local}+1)\times p \quad (4)$$

The API algorithm can be viewed as a restarted parallel algorithm which carries out a number of parallel local searches starting from the same point.
Generally, in an optimization problem, the nest and the hunting sites represent admissible solutions.

5 The hybrid genetic algorithm developed

The HGA proposed hybridizes the GA with the API-Fourragement search procedure. The algorithm begins by generating an initial population randomly. The GA operators are performed exactly like in the basic GA until the mutation operator. In a basic GA selection, crossover and mutation operators are applied to parents in order to generate offspring. Therefore, good performers propagate through the population from generation to the next. Thus, a good exploration of each generation could yield to best solutions. We propose to explore solutions found by the GA (offsprings) using the API-Fourragement algorithm. The main idea is to apply the API-Fourragement on offsprings found by the GA in order to improve them before returning to be evaluated. This can offer to the HAG a good balance between the intensification and diversification search procedures because of its complementary operations.

Before giving the HGA, the solutions encoding, crossover and mutation operators have to be explained.

Solution encoding: In a flow shop scheduling problem, the aim is, generally, to find the best jobs sequence which permits to minimize Cmax. A feasible solution in our study could be viewed as a jobs sequence. Fig. 1. shows an example of a possible solution in our study for 10 jobs having to be processed.

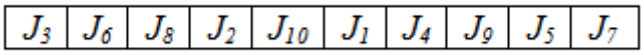

Fig. 1. An example of solution encoding

- Crossover operator: The crossover operator is applied on each two selected parents from the population. It can be done by choosing a crossover point in the chromosome at hand, keeping the left part of the first chromosome and finding in the second chromosome the remaindering genes to complete the new constructed chromosome (offspring) and doing the same procedure on the second parent. Fig. 2. gives an illustrating example of the crossover operator applied on two parents.

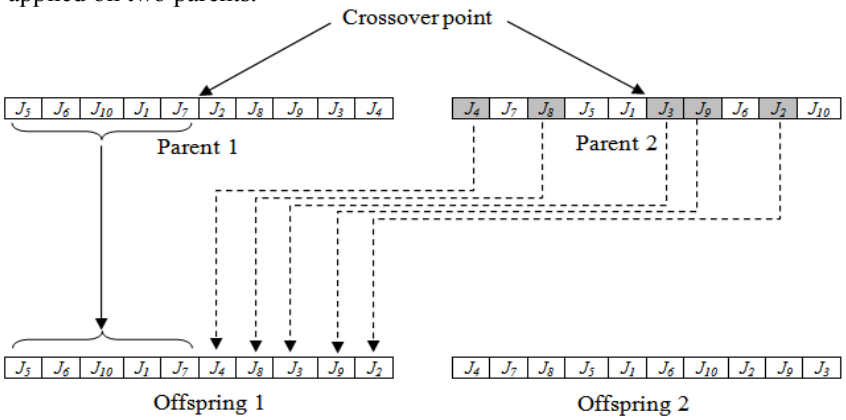

Fig. 2. Example of the crossover operator applied on two selected parents.

- Mutation operator: The mutation operator is done by selecting randomly a pair of jobs and swaps them to obtain a new chromosome. An illustrating example of the mutation operator is given in Fig. 3.

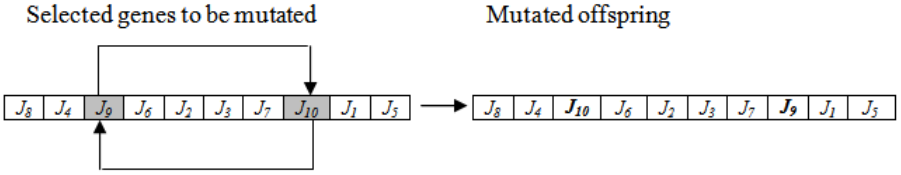

Fig. 3. Example of the mutation operator.

The HGA Algorithm is given by the following steps:

```
Generate an initial population of PopSize solutions
Evaluate each individual in the population by
calculating its fitness function (Cmax)
While stopping criterion is not reached
Select two individuals (named parents)
    Perform the crossover operator on the two selected
individuals
    Perform the mutation operator on each obtained
solution (offspring) from the selection operator
Choose randomly one solution among obtained solutions
to be the nest for the API-Fourragement
Consider all other obtained solutions (mutated
offsprings) as hunting sites for the API-Fourragement
For each ant ai
Perform the API-Fourragement
Replace parents by obtained offsprings
Output the best solution
```

6 Results and discussion

Based on the implementation of the HGA described in section 5, a number of simulation experiments are carried out in order to show the performance of the developed algorithm. For this aim we consider three different sized classes of problems on the basis of the number of jobs, j = (20, 30, 50). In each class 5 instances generated for a production system characterized by 30 machines. The processing times are generated in the range [0, 30].

Firstly, we have adapted and simulated the basic genetic algorithm to solve our problem and then we have developed and simulated the HGA. For both GA and HGA a sensitivity analysis has been done in order to determine the best choice of parameters values for each algorithm.

6.1 Sensitivity analysis of the GA

The number of examined solutions play a main role in population based metaheuristics. In a GA the population size determines the number of solutions to be examined at each iteration. In this section a number of computational results are given for three different sized classes of the considered FSSP according to different population size values.

Table 1. Sensitivity analysis on the population size of the GA

		C_{max}		
	Pop_{Size}	10 jobs 20 machines	20 jobs 20 machines	50 jobs 20 machines
	10	738	1078	1745
	20	727	1054	1730
Example 1	30	728	1044	1743
	40	**722**	**1032**	**1720**
	50	724	1037	1729
	10	787	1071	1726
	20	777	1069	1723
Example 2	30	774	1057	1696
	40	**770**	**1052**	**1673**
	50	773	1062	1694
	10	778	1062	1737
	20	765	1034	1725
Example 3	30	739	1024	1714
	40	**732**	**1008**	**1694**
	50	752	1013	1704
	10	777	1040	1723
	20	786	1040	1715
Example 4	30	783	1034	1708
	40	**768**	**1030**	**1687**
	50	772	1036	1688
	10	820	1085	1734
	20	809	1068	1724
Example 5	30	802	1062	1721
	40	**799**	**1041**	**1718**
	50	802	1052	1729

Table 1 illustrates computational results of the sensitivity analysis of the Genetic Algorithm according to the population size. Computational results show that the GA

performs better for a population size equals to 40. This can be explained by the fact that more the explored search space (i.e : the number of examined solutions) is large better the obtained solutions quality is. But if the population size is higher we may have saturation.

6.2 Sensitivity analysis of the HGA

To determine a good choice of the HGA parameters, a number of simulations have been done according to the population size.

Table 2. Sensitivity analysis of the HGA

	Number of sites (p)	Pop_{Size}	C_{max}		
			10 jobs 20 machines	20 jobs 20 machines	50 jobs 20 machines
Example 1	3	10	735	1052	1745
	5	16	731	1043	1734
	7	22	7.30	1039	1726
	9	**28**	**719**	**1021**	**1718**
	11	34	725	1036	1723
	13	40	726	1047	1738
Example 2	3	10	766	1061	1720
	5	16	765	1052	1711
	7	22	760	1052	1696
	9	**28**	**758**	**1043**	**1657**
	11	34	765	1055	1668
	13	40	770	1057	1685
Example 3	3	10	759	1023	1726
	5	16	755	1006	1688
	7	22	741	1001	1689
	9	**28**	**727**	**995**	**1678**
	11	34	744	1002	1696
	13	40	749	1027	1704
Example 4	3	10	785	1021	1705
	5	16	782	1020	1693
	7	22	772	1001	1679
	9	**28**	**761**	**999**	**1671**
	11	34	771	1018	1677
	13	40	773	1022	1679
Example 5	3	10	817	1066	1727
	5	16	806	1065	1717
	7	22	801	1044	1738
	9	**28**	**789**	**1035**	**1694**
	11	34	806	1039	1704
	13	40	809	1057	1710

However, solutions obtained by the GA will be explored by the API-Fourragement the population size of the GA must depend on the number of hunting sites explored by each ant of the API-Fourrragement. For this reason we developed the equation (5) which gives the population size (Pop_{Size}) according to the number of ants and the number of hunting sites.

$$Pop_{Size}=(p\times ants_{nbr})+1. \quad (5)$$

Equation (5) shows that the number of all solutions (Pop_{Size}) to be explored is the number of solutions explored by each ant (p) plus one solution which is the nest. That means the HGA population is built by solutions classified as:

The nest: which is a solution chosen initially in a random way

Hunting sites: which are all other solutions subdivided into $ants_{nbr}$ subpopulations.

The sensitivity analysis results according to the number of hunting sites and consequently to different population sizes is given in table 2. The number of hunting sites varies from 3 to 13 and therefore the population size varies too.

Computational results show that the solutions quality is better for a number of hunting sites equals to 9.

6.3 Comparison results of the HGA and the basic GA

Table 3 shows computational results of both the GA and HGA for best obtained parameters yielding to best C_{max} for each algorithm (Pop_{Size} =40 for the GA and p=9 for the HGA).

Table 3. Comparison results of the HGA and the basic GA

	10 jobs &10 machines		10 jobs & 20 machines		10 jobs & 50 machines	
	GA	HGA	GA	HGA	GA	HGA
Example 1	722	719	1032	1021	1720	1718
Example 2	770	758	1052	1043	1673	1657
Example 3	732	727	1008	995	1694	1678
Example 4	768	761	1030	999	1687	1671
Example 5	799	789	1041	1035	1718	1694

Computational results show that the HGA performs better than the basic GA for the different sized classes of the FSSP considered in term of solutions quality because of its complementary procedures even for relatively high instances.

7 Conclusion

In this paper, a hybrid algorithm (HGA) combining GA and API-Fourragement procedure is proposed to solve the flow shop scheduling problem with makespan minimization. The most challenging problem of the GA is to prevent early

convergence. The HGA deals with that by the use of the AI-Fourragement search procedure. A sensitivity analysis has been done to determine the best parameters of the algorithm.

The good choice of the algorithm parameters plays a significant role in metaheuristics. In future study, a thorough investigation may be done on this issue. Experimental results show that induction of the API-Fourragement permits to the HGA algorithm to outperform regular GA in term of solutions quality. Also, this method can be easily extended to solve other FSSP with other criteria which are also hard problems, such as maximum tardiness, total tardiness, etc.

References

1. Holland, J.H.: Adaptation in natural and artificial systems, University of Michigan Press, Ann Arbor, (1975)
2. Dorigo, M., DiCaro, G.: The ant colony optimization meta-heuristic, in: D.Corne, M.Dorigo, F.Glover (Eds.), New Ideas in Optimization, Mc Graw Hill, London,U, pp.11–32.
3. Korošec, P., Šilc, J., Filipič, B.: The differential ant-stigmergy algorithm, Information Sciences 192, pp.82–97, http://dx.doi.org/10.1016/j. ins.2010.05.002, (1999)
4. Kennedy, J., Eberhart, R.: The particle swarm optimization : social adaptation in information processing, in: Corne, D., Dorigo, M., Glover, F., (Eds.), New Ideas in Optimization, McGrawHill, London, UK, pp.379–387, (1999)
5. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization vol. 39(3), pp. 459–471, (2007)
6. Fister, I., Jr., Brest, J., Žumer, V.: Memetic artificial bee colony algorithm for large-scale global optimization, in: IEEE Congress on Evolutionary Computation, pp.1–8 (2012)
7. Gandomi, A., Yang, X.-S., Talatahari, S., Alavi, A.: Metaheuristic in modeling and optimization, in: Gandomi, A., Yang, X.-S., Talatahari, S., Alavi, A., (Eds.), Metaheuristic Application in Structures and Infrastructures, Elsevier, Wal-tham, pp.1–24, (2013)
8. Yang, X., S.: Optimization and metaheuristic algorithms in engineering, in: X.-S. Yangetal. (Ed.), Metaheuristic in Water Geotechnical and Transport Engineering, Elsevier, Waltham, pp.1–23, (2013)
9. Sahab, M., Toropov, V., Gandomi, A.: Traditional and modern optimization techniques – theory and application, in: Gandomi, A.H.et *al.* (Ed.), Metaheuristic Applications in Structures and Infrastructures, Elsevier, Waltham, pp. 26–47, (2013)
10. Yang, X., S., Deb, S.: Cuckoo search via levy flights, in: World Congress on Nature & Biologically Inspired Computing (NaBIC2009), IEEE Publications, pp.210–214, (2009)
11. Zhang, Z., Juig, Z.: An improved Ant Colony for permutation Flow Shop scheduling to minimize makespan, 13 th International Conference on parallel and distributed computing, Applications and Technologies, (2013)
12. Zhang, J., Tongand, J., Ma, Y.: An effective Hybrid Ant Colony Optimization for permutation Flow Shop Scheduling, The Open Automation and Control Systems Journal, pp. 62-68, (2014)
13. Ahmadizar, F.: A new ant colony algorithm for makespan minimization in permutation Flow Shop, Computer and Industrial Engineering, vol. 63, pp. 35;5-361, (2012)
14. Li, J.Q., Pan, Q. K.,&Gao, K. Z. Pareto-based discrete artificial bee colony algorithm formulti-objective flexible job shop scheduling problems. International Journal of Advanced Manufacturing Technology, vol. 55(9–12), 1159–1169, (2011)

15. Allaoua, H., Osmane, I.: Variable parameters lengths genetic algorithm for minimizing earliness-tardiness penalties of single machine scheduling with a common due date, Electronic Notes in Discrete Mathematics 36, 471–478, (2010)
16. Dorigo, M.: Optimization, learning and natural algorithm, in Italian. Ph.D. thesis, DEI, Politecnico di Milano, Italy, (1992)
17. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybern. B vol. 26, 29–41, (1996)
18. Holthaus, O., Rajendran, C.: A fast ant-colony algorithm for single-machine scheduling to minimize the sum of weighted tardiness of jobs. J. Oper. Res. Soc. 56, 947–953, (2005)
19. Marimuthu, S., Ponnambalam, S., G., Jawahar, N.: Threshold accepting and ant-colony optimization algorithms for scheduling m-machine flow shops with lot streaming. J.Mater. Process.Technol. vol. 209, pp. 1026–1041, (2009)
20. Lin, B. M. T., Lu, C. Y., Shyu, S. J., Tsai, C. Y.: Development of new features of ant colony optimization for flowshop scheduling. International Journal of Production Economics, vol. 112, pp. 742–755, (2008)
21. Rajendran, C., Ziegler, H.: Ant-colony algorithms for permutation flow shop scheduling to minimize makespan/total flow time of jobs. European Journal of Operational Research, vol.155, 426–438, (2004)
22. Monmarché N.: Algorithmes de Fourmis Artificielles: Application à la Classification et l'Optimisation, University of François Rabelais, Tours, (2000)
23. Monmarché, N., Venturini, G., Slimane, M.: On how Pachycondyla apicalis ants suggest a new search algorithm, Future Generation Computer Systems 16, pp. 937–946, (2000)
24. Debbat, F., Bendimerad, F.T.: Radiation pattern optimization by Apicalis Ant algorithm for smart arry antennas, International Journal of Scientific & Engineering Research, Vol. 3, Issue 11, (2012)
25. Houbad, Y., Souier, M., Hassam, A., Sari, Z.: Algorithme API hybride pour la resolution du problem de selection de routages alternatives dans un FMS, International Conference on Systems and Processing Information, Guelma, Algeria, (2013)
26. Yamina, H., Mehdi, S., Ahmed, H., Zaki, S., Fayçal, B.: An APIm Algorithm to Solve the Scheduling Problem in an FMS with Presence of Breakdowns, Applied Mechanics and Materials, Vol. 232, pp. 532-536, (2012)