2016 12th World Congress on Intelligent Control and Automation (WCICA)
June 12-15, 2016, Guilin, China

An MBO Algorithm for a Flow Shop Problem with Sequence-Dependent Setup Times

Aymen Sioud and Caroline Gagné

***Abstract*— In this paper, we propose a migrating birds algorithm (MBO) to solve a permutation flowshop with sequence-dependent setup times and the objective of minimizing the makespan. In this approach, we adapt several MBO operators and we embed an intensification process. Indeed, we introduce an original leader selection process, a restart mechanism and an adapted neighborhood. Sensitivity analysis results are further presented to show the effectiveness and performance of the presented MBO on benchmarks from the literature.**

I. Introduction

WITH more tough competition and economic environment, manufacturers are obliged to make several optimization plans. The same goes for their production systems where most of the classical scheduling models move away from reality and no longer meet the real requirements. Indeed, managers face increasingly more complex constraints such as job precedence, time lags and setup times, to name just a few. In the literature, sequence-dependent setup times have attracted less attention, although there are frequently setup times on equipment between two different activities in many industries including the food industry, electronics, pharmaceutical and metallurgical production [1]. Sequence-dependent setup times involve operations that have to be performed on machines that are sequence-dependent, i.e, the order changes the setup time, which is not part of the processing time, as considered in a lot of papers [1]. In [2], when studying 250 industrial projects, the author shows that when these setup times are taken into account, 92% of the order deadlines would be met and that 50% of the projects contain sequence-dependent setup times. Many other studies show that considering sequence-dependent setup times produces good schedules [1][3][4].

Among the classical scheduling models, the flowshop scheduling area has been a very active research field where we find a great number of papers [6]. A flowshop scheduling problem can be defined by a given set $N = \{1, \ldots, n\}$ of n independent jobs that have to be processed on a set $M = \{M_1, ..., M_m\}$ of m machines. Sequentially, all jobs are processed on machine M_1, then on machine M_2 and so on until the last machine M_m. Each job has a known and deterministic fixed processing time denoted as p_{ij}, $i \in M$ and $j \in N$. To be closer to real production systems, several constraints have been added to this model, such as sequence-dependent setup times.

Aymen Sioud and Caroline Gagné are with the Department of Computing and Mathematic, The University of Quebec, Chicoutimi, Quebec, Canada (email: {aymen.sioud, caroline.gagne}@uqac.ca).

This work was supported by the Natural Sciences and Engineering Research Council of Canada

In this paper we consider the permutation flowshop problem with sequence-dependent setup times and the objective of minimizing the makespan, noted as $F/prmu, s_{ijk}/C_{max}$ in accordance with the notation of Graham *et al.* [5]. In the flowshop model, when the sequence on the first machine is the same for all the other machines, i.e the jobs can not overlap, the flowshop is called the permutation flowshop. Also, deterministic and non-negative sequence-dependent setup times are also considered and denoted by s_{ijk} on machine i, $i \in M$, when processing the job k, $k \in N$ after processing the job j, $j \in N$. For $m = 1$, the sequence-dependent setup times flowshop problem is known to be a special case of the Traveling Salesman Problem (TSP), that is also well known to be $\mathcal{NP}$-hard [7], making the $F/prmu, s_{ijk}/C_{max}$ problem is also $\mathcal{NP}$-hard.

The general permutation flowshop has been more thoroughly treated in the literature than the permutation flowshop with sequence-dependent setup times. In [6] and [8], the authors give very comprehensive literature surveys of the permutation flowshop problem, but they do not mention any research dealing with sequence-dependent setup times. To solve the $F/prmu, s_{ijk}/C_{max}$, both Gharbi *et al.* [9] and Rios-Mercado *et al.* [11] introduced lower bounds in a branch and bound algorithm that was used to solved small instances. Ladhari *et al.* [10] and Shaller [12] proposed constructive methods and greedy searches, respectively. In the literature, we found only a few metaheuristics to solve the $F/prmu, s_{ijk}/C_{max}$ problem. Shen *et al.* [13], Ruiz and Maroto [14] and Ciavotta *et al.* [15] introduced tabu search, a memetic algorithm and an iterated greedy heuristic (IGH), respectively. In a comparison study, the IGH of Ciavotta *et al.* [15] improved the results of the other heuristics/metaheuristics. More recently, a migrating birds algorithm [16] was used to solve the $F/prmu, s_{ijk}/\sum C_i$ problem. The algorithm presented there behaves well compared to other algorithms from the literature[19].

In this paper we introduce an original migrating birds optimization algorithm (MBO) to solve the $F/prmu, s_{ijk}/C_{max}$ problem. Several adaptations are made to this MBO in order to solve this problem. Indeed, we introduce a new intensification capability, an adapted neighborhood and a restart process. The new leader selection process is also enhanced to improve the MBO behavior. The adaptations, the MBO algorithm and the experimental results are presented in the following sections. The last section provides a conclusion and perspectives for future work.

978-1-4673-8414-8/16/$31.00 ©2016 IEEE

II. MBO ALGORITHM FOR THE $F/prmu, s_{ijk}/C_{max}$

In [16], the authors introduced the migrating birds optimization (MBO), a new nature-inspired metaheuristic for combinatorial optimization problems inspired by bird migration. This neighborhood based metaheuristic uses the birds' "V" formation to improve the solutions. Indeed the "V" formation allows the birds to save energy and therefore to travel longer distances. They use the movement of the air caused by the flapping of a leader's wing. When the leader is exhausted, it goes to the end of the line and another bird takes its place at the front.

Starting with the leader, i.e, the first solution, the MBO tries to improve the solution by exploring its *k size* neighborhood. The following solution evaluates both a number of its own neighbors (*x* neighbors) and a number of the best unused neighbors (*k - x*) from the previous solution. The unused neighbors represent solutions inherited from the neighbor that is not used to replace the existing solution. If it is better, the best solution replaces the current solution. Once all of the solutions in the flock are considered, this process is repeated again. After *m* tours, the leader solution is moved alternately to the end of one of the wings, and the following solution in the same wing is forwarded to the leader position. This process is repeated until a termination condition is met (*K* iterations).

After initializing the MBO algorithm, the leader is improved by generating a mixed neighborhood based on the swap and the forward insertion moves [17]. Then, an improvement heuristic is applied to the leader as an intensification process. In our case we use a simple hill climbing algorithm based on the swap move [17]. The followers are improved in the same manner. This process aims to enhance the intensification process and exploit more search space.

In general, the metaheuristics have different ways of balancing exploitation and diversification. Whereas the MBO focuses strongly on its exploitation ability, we also aim to avoid premature convergence by introducing a restart phase. The main advantage of this restart procedure is that it is a very fast way to introduce diversification inside the presented MBO. The main inconvenience consists of the difficulty in choosing a suitable restarting criterion. In our case, for each solution in the flock, we use an age variable to represent the updating surviving process. The age of a newly created solution is set to zero. At each iteration and when a solution is not replaced, its age is increased by one. If the solution is improved, its age is reset to zero. A solution will be replaced when its age is larger than a parameter noted *age*. In this case, this solution is replaced by a randomly generated one.

In the basic MBO algorithm [16], the leader solution is moved alternately to the ends of the left and the right wings, and the first solution in the corresponding wing becomes the new leader as in the real bird migration process. In the presented MBO algorithm the leader is chosen by a pseudo-random rule according to the age solution variable as shown in Equations (1) and (2). Indeed, the less the age is, the more the solution has a chance to become the leader. This process encourages exploiting promising "'young" regions in the search space. In Equation (1), q is a random number and q_0 is a parameter; both are between 0 and 1. The parameter q_0 determines the relative importance of the *age* variable. Indeed, Equation (1) states that the next leader will be a solution in the left or right wing equiprobably when $q \leq q_0$ or by the probabilistic rule of Equation (2) when $q > q_0$. Equation (2) describes the rule $p_{SOL_i}(t)$ based on the age solution variable.

$$\textit{next leader} = \begin{cases} \textit{first bird in left or right wing} & \textit{if } q \leq q_0 \\ \textit{SOL} & \textit{if } q > q_0 \end{cases} \tag{1}$$

where *SOL* is chosen according to the probability p_{SOL}

$$p_{SOL_i}(t) = \frac{\frac{1}{age_{SOL_i}(t)}}{\sum_{i=1}^{n} age_{SOL_i}(t)} \tag{2}$$

The introduced MBO algorithm is presented in Figure 1. After initializing n random solutions, these solutions are placed on hypothetical V formation arbitrarily. Then the main *K*-loop starts. The intensification process is applied to each solution before generating the neighborhood using both swap and forward insertion. This process is applied b times and returns the best solution found. After both the intensification and neighborhood generating phases, the age variable is updated. Finally, the new leader is chosen by a pseudo-random rule based on the age variable among all the other solutions in the flock after m tours.

Procedure MBO

Initialize n random solutions
Place the n solutions arbitrarily in a hypothetical V formation
$i := 0$
while $i \leq K$ **do**
  for $i := 0$ **to** m **do**
    Apply improvement heuristic to the leader with b iterations
    Update the leader age variable
    Generate the k neighborhood of the leader
    Update the leader age variable solutions
    for all the other flock solutions **do**
      Apply improvement heuristic with b iterations
      Update the solution age variable
      Generate the $k - x$ neighborhood of the solution
      Update the solution age variable
    end for
  end for
  Select a new leader using a pseudo-random rule according to the age
  Promote the new leader
end while

Fig. 1. The MBO algorithm

III. Computational results and discussion

In [15], the authors introduced benchmarks consisting of 220 problem sets and based on the benchmarks of [18]. Each set contains instances with several combinations for the number of jobs n and number of machines m. The $n \times m$ combinations are: {20, 50, 100} x {5, 10, 20} and {200} x {10, 20} for a total of 11 different groups of instances. The processing times p_{ij} are generated from a uniform [1, 99] distribution. The setup times are generated according to two distributions [0, 49] and [0, 124]. This corresponds to a ratio between setup and processing times of 50% and 125%, respectively. The two sets are referred to as SSD50 and SSD125. In the original benchmarks, weights are considered, which is not the case here.

All the experiments were run on an Intel Core i7 2.8 GHz processor with 8 GB of main memory. Each instance was executed 20 times. To evaluate the different algorithms we use the relative percentage deviation (*RPD*) as shown in Equation 3.

$$RPD = \frac{M_{sol} - Best_{sol}}{Best_{sol}} \times 100 \quad (3)$$

Here M_{sol} is the makespan obtained by a given algorithm and $Best_{sol}$ is the best makespan obtained by the whole experiment. The response variable is the average of the 20 executions for the considered heuristic.

We first proceed to the comparisons of the proposed MBO algorithm with other existing algorithms from the literature. Among the existing methods, we have fully recoded in C++ the following : the IGH in [15] and the two migrating birds optimization algorithms in [19] and [16], noted as IGH, TBO and IBO, respectively. All these algorithms have shown high performance in the original papers. In their respective papers, IGH and IBO used CPU times as the stopping criterion. In this work, we used 30 000 evaluations as the stopping criterion for all the algorithms. For the introduced MBO the parameters are initialized as follows : $n = 9$, $m = 100$, $k = 5$, $x = 1$, $age = 100$, $q_0 = 0.7$ and $b = 10$. The averages grouped by group instances of IGH, TBO, IBO and the presented migrating birds optimization algorithm are presented in Table I where the MBO column shows the results of the MBO algorithm.

As we can see, the proposed MBO algorithm outperforms the IGH, TBO and IBO algorithms in every instance group. Indeed, MBO obtain best RPD in every instance group. Adapting the neighborhood enhancing the leader quality could explain this MBO algorithm behavior. We can conclude that these processes enhance the search space exploration. When we look at the MBO results, the improvement is noticeable on all the-job instances. The best averages are in boldface in this table. As can be seen, MBO provides statistically better results them the other tested algorithms.

We also analyze the efficiency of the different algorithms, measuring the time in seconds that a given method needs in order to provide a solution. We remind the reader here that the stopping criterion is 30 000 evaluations for all the considered algorithms. We note that all the algorithms are less time-consuming. The IGH obtains the best CPU time average. We can remark that the introduced MBO is very competitive (1.00 vs 0.59 average) and obtains the second fastest CPU time in every instance group.

TABLE I
Comparison of different algorithms for the $F/prmu, s_{ijk}/C_{max}$

Instances	IGH [15]	TBO [19]	IBO [16]	MBO
20x5	3.77	2.86	5.76	**1.40**
20x10	7.44	7.21	8.71	**5.67**
20x20	15.09	14.41	16.00	**13.23**
50x5	4.06	4.24	1.93	**0.88**
50x10	4.57	4.06	2.36	**1.65**
50x20	11.95	11.64	10.11	**9.14**
100x5	6.85	6.74	1.90	**0.73**
100x10	5.48	5.31	1.79	**0.43**
100x20	9.27	8.75	5.77	**4.68**
200x10	7.69	7.57	3.17	**0.21**
200x20	6.52	6.28	2.71	**0.21**
Average	7.52	7.19	5.47	**3.48**
STD	3.47	**3.41**	4.52	4.31

TABLE II
Average CPU times for tested algorithms in seconds

Instances	IGH [15]	TBO [19]	IBO [16]	MBO
20x5	0.13	0.30	0.27	0.19
20x10	0.14	0.33	0.31	0.22
20x20	0.18	0.40	0.40	0.31
50x5	0.26	0.67	0.78	0.42
50x10	0.29	0.79	0.85	0.52
50x20	0.39	1.06	1.11	0.77
100x5	0.48	1.25	1.80	0.81
100x10	0.58	1.49	1.90	0.98
100x20	0.81	2.09	2.59	1.56
200x10	1.27	2.89	4.25	1.97
200x20	1.94	4.34	5.67	3.31
Average	0.59	1.42	1.81	1.00

IV. Conclusions

In this paper, we have introduced a migrating birds optimization to solve a permutation flow shop with sequence-dependent setup times by minimizing the makespan. The proposed MBO algorithm uses an adapted neighborhood search based on swap and forward insertion moves. Besides adapting the neighborhood solution, the MBO embeds an original new leader selection process and a restart process. The proposed MBO improves state-of-the-art results for all group instances from the literature.

We think that more adaptations can be made to improve the proposed algorithm's capability and behavior. We are also considering using it to solve other real scheduling problems such mutli-objective ones.

REFERENCES

[1] A. Allahverdi, C. Ng, T. Cheng, and M. Y. Kovalyov, "A survey of scheduling problems with setup times or costs," *European Journal of Operational Research*, vol. 187, no. 3, pp. 985 – 1032, 2008.

[2] G. Conner, "10 questions," *Manufacturing Engineering Magazine*, pp. 93–99, 2009.

[3] A. Allahverdi and H. Soroush, "The significance of reducing setup times/setup costs," *European Journal of Operational Research*, vol. 187, no. 3, pp. 978 – 984, 2008.

[4] X. Zhu and W. E. Wilhelm, "Scheduling and lot sizing with sequence-dependent setup: A literature review," *IIE Transactions*, vol. 38, no. 11, pp. 987–1007, 2006.

[5] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. G. H. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[6] R. Ruiz and C. Maroto, "A comprehensive review and evaluation of permutation flowshop heuristics," *European Journal of Operational Research*, vol. 165, no. 2, pp. 479 – 494, 2005, project Management and Scheduling.

[7] J. N. Gupta and W. P. Darrow, "The two-machine sequence dependent flowshop scheduling problem," *European Journal of Operational Research*, vol. 24, no. 3, pp. 439 – 446, 1986, flexible Manufacturing Systems.

[8] Q.-K. Pan and R. Ruiz, "A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime," *Computers & Operations Research*, vol. 40, no. 1, pp. 117 – 128, 2013.

[9] A. Gharbi, T. Ladhari, M. K. Msakni, and M. Serairi, "The two-machine flowshop scheduling problem with sequence-independent setup times: New lower bounding strategies," *European Journal of Operational Research*, vol. 231, no. 1, pp. 69 – 78, 2013.

[10] T. Ladhari, M. Msakni, and A. Allahverdi, "Minimizing the total completion time in a two-machine flowshop with sequence-independent setup times," *Journal of the Operational Research Society*, vol. 63, pp. 445 – 459, 2011.

[11] R. Z. Rios-Mercado and . F. Bard, "A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times," *IIE Transactions*, vol. 31, no. 8, pp. 721–731, 1999.

[12] J. Schaller, "Scheduling a permutation flow shop with family setups to minimise total tardiness," *International Journal of Production Research*, vol. 50, no. 8, pp. 2204–2217, 2012.

[13] L. Shen, J. N. Gupta, and U. Buscher, "Flow shop batching and scheduling with sequence-dependent setup times," *Journal of Scheduling*, vol. 17, no. 4, pp. 353–370, 2014.

[14] R. Ruiz and C. Maroto, "A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility," *European Journal of Operational Research*, vol. 169, no. 3, pp. 781–800, March 2006.

[15] M. Ciavotta, G. Minella, and R. en Ruiz, "Multi-objective sequence dependent setup times permutation flowshop: A new algorithm and a comprehensive study," *European Journal of Operational Research*, vol. 227, no. 2, pp. 301 – 313, 2013.

[16] E. Duman, M. Uysal, and A. F. Alkaya, "Migrating birds optimization: A new metaheuristic approach and its performance on quadratic assignment problem," *Information Sciences*, vol. 217, no. 0, pp. 65 – 77, 2012.

[17] M. Prandtstetter and G. R. Raidl, "An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem." *European Journal of Operational Research*, vol. 191, no. 3, pp. 1004–1022, 2008.

[18] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278 – 285, 1993, project Management anf Scheduling.

[19] Q.-K. Pan and Y. Dong, "An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimisation," *Information Sciences*, vol. 277, no. 0, pp. 643 – 655, 2014.