Available online at www.sciencedirect.com

ScienceDirect

IFAC-PapersOnLine 49-12 (2016) 408–413

The migrating birds optimization metaheuristic for the permutation flow shop with sequence dependent setup times

Imène Benkalai [*] **Djamal Rebaine** [*] **Caroline Gagné** [*]
Pierre Baptiste [**]

[*] *Université du Québec à Chicoutimi, Saguenay (QC), Canada G7H-2B1(e-mails: imene.benkalai1@uqac.ca ; djamal.rebaine@uqac.ca ; caroline.gagne@uqac.ca).*
[**] *École Polytechnique de Montréal, Montréal (QC), Canada H3T-1J4 (e-mail: pbaptiste@polymtl.ca)*

Abstract: This paper addresses the problem of scheduling a set of independent jobs with setup times on a set of machines in a permutation flow shop environment. A metaheuristic known as the Migrating Birds Optimization (MBO for short) is designed for the minimization of the overall completion time. A computational study is conducted to analyze the efficiency of this approach on instances that can be found in *Sistemas de Optimizacion Aplicada* (`http://soa.iti.es/problem-instances`).

Keywords: Scheduling, flow shop, setup times, metaheuristic, migrating birds optimization.

1. INTRODUCTION

We address in this paper the permutation flow shop problem with sequence dependent setup times. This problem can be briefly described as follows. We are given a set of n independent jobs to be processed on a set of m machines. Each job comprises m operations that have to be processed without preemption, in this order, on machine M_1, machine M_2, and so on until it completes on machine M_m. The operating sequence of the jobs is the same on every machine, *i.e.* if one job is at the j-th position on machine M_1, then this job will be at the j-th position on all the machines. Furthermore, if operations j and k are processed in that order on machine M_i, then a setup time s_{ijk} must elapse at the completion of operation j and before operation k takes place on machine M_i. This time is needed to adjust machines before they execute the next job in the processing sequence. We seek a valid schedule of the jobs in order to minimize the overall completion time, known as the makespan. Using the classical scheduling notation (see e.g. Pinedo [2002]), this problem is represented as $F_m|prmu, s_{jk}|C_{max}$.

In cases involving sequence dependent setup times, the algorithms for minimizing the makespan tend to be complicated ; indeed, the consideration of setup times adds to the complexity of scheduling problems, as it is the case in the one-machine environment for example [Pinedo [2002]].

Flow shop scheduling problems with setup times exist naturally in many real world situations. Various applications are given in the literature where setup times may be needed for instance to change temperature or a manufacturing mold. We may cite the paper cutting where the machines need to be adjusted when changing from one cutting batch to another [Ruiz and Stutzle [2008]]. Another example is the problem that arises when manufacturing different types of tires requiring to change the manufacturing mold of the multi-purpose machines. A third and last example is the bakery problem where the baked products need different temperatures to cook, so adjustments would be required between two products.

The permutation flow shop scheduling problem with setup times is known to be $\mathcal{NP}$-hard in the strong sense [Gupta [1986]], even for the case of $m = 1$ machine [Ruiz et al. [2005]]. Therefore, the approximation approach is well justified as a solving method. In recent years, intelligence-oriented search algorithms such as Genetic Algorithms, Tabu Search, Simulated Annealing, and so on, that mimic nature phenomena, have been employed to solve scheduling problems. In this paper, we focus on a metaheuristic known as the Migrating Birds Optimization method (MBO for short), recently introduced by Duman *et al.* [Duman et al. [2012]]. This metaheuristic has only been applied to a very few problems. In addition, the promising results of this method on the Quadratic Assignment Problem [Duman et al. [2012]] make it a good candidate to solve the problem under study.

This paper is organized as follows. Section 2 describes the problem under study. Section 3 presents a brief review of the previous works related to this problem. Section 4 discusses the Migrating Birds Optimization method. Section 5 presents the experimental study we conducted on the efficiency of this method. Finally, in Section 6, we present our concluding remarks.

2. PROBLEM DESCRIPTION

The flow shop scheduling problem with setup times can be described as follows. Let $J = \{1, 2, \ldots, n\}$ be a set of n jobs to be processed without preemption by $M = \{M_1, M_2, \dots, M_m\}$ a set of m machines. A processing time p_{ij} is given to denote the time spent by job j on machine M_i, and a setup time s_{ijk} denotes the time that must elapse between the completion of job j and the start of job k whenever the former precedes the latter on machine M_i in a given sequence. We seek a valid schedule of the jobs in order to minimize the makespan. More formally, we seek a permutation $\pi = (\pi(1), \dots, \pi(n))$ on set J such that

$$C_{max}(\pi^*) = \max_{\pi \in \Pi} C_{max}(\pi),$$

where Π denotes the set of the $n!$ permutations over set J, π and π^* are in Π, and $C_{max}(\pi)$ is the overall completion time associated with permutation π.

The value of the makespan for a given permutation π is computed as follows. Let $C(\pi(j), i)$ be the completion time of job $\pi(j)$ on machine M_i. It then follows $C_{max}(\pi) = C(\pi(n), m)$, where $C(\pi(j), k)$ of job $\pi(j)$ on machine M_k is given by the following recursive formula.

$$C(\pi(1), k) = \sum_{i=1}^{k} p_{i,\pi(1)}; k = 1, \ell, m$$

$$C(\pi(j), 1) = \sum_{q=1}^{j-1} (p_{1,\pi(q)} + s_{1,\pi(q),\pi(q+1)}) + p_{1,\pi(j)}; j = 2, \ell, n$$

$$C(\pi(j), k) = \max\{C_{k-1,\pi(j)}, C_{k,\pi(j-1)} + s_{k,\pi(j-1),\pi(j)}\}$$
$$+ p_{k,\pi(j)}; k = 2, \ell, m \ ; \ j = 2, \ell, n.$$

3. RELATED WORKS

Since the problem is $\mathcal{NP}$-hard in the strong sense, the metaheuristic approach has been extensively utilized as a solving approach for the $F_m|prmu, s_{jk}|C_{max}$ problem. In this section, we review the state of the art of the important works that have been done in this area. A general survey is given in Allahverdi *et al.* [Allahverdi et al. [2008]].

Let us first start with the Genetic Algorithm designed by Ruiz *et al.* [Ruiz et al. [2005]]. The authors compare the efficiency of Genetic Algorithms with other approaches that have been shown to produce good results on the standard flow shop (without setup times). They extended Taillard's standard flow shop instances. Their Genetic Algorithm modifies a part of the population if the search happens to "stagnate" [1], and has a selection strategy based on The Roulette Wheel and Tournament Selection [Talbi [2009]]. The offspring are taken to build the next population along with the best parents. The authors also introduce a new crossover operator that they consider more efficient. It allows to preserve sequences of the solutions by copying "blocs" of two jobs that are present in both parents directly in the offspring and then applies the two-point crossover. It is claimed in that paper this algorithm outperforms by far the other algorithms listed in their study.

Kumar and Singhal [Kumar and Singhal [2013]] also proposed a Genetic Algorithm for the resolution of the same problem as above with splittings. This algorithm has the same evolution strategy as the previous one, a tournament selection strategy, and uses a two-point crossover followed by a repair phase [2]. Its mutation operator consists in re-positioning a job. The authors also tested the effects of the variation of crossover and mutation probabilities on the quality of the results. Their algorithm often manages to find the optimal solutions for the instances in their testbed more or less rapidly depending on the previously cited probabilities.

Another algorithm, the Iterated Greedy Algorithm, was also applied to the same problem as above by Ruiz and Stützle [Ruiz and Stutzle [2008]]. This algorithm consists of two phases: a phase of destruction that produces a partial solution, and a phase of construction that completes the latter and replaces the current solution according to some acceptance criterion. Their experimental study shows that the efficiency of this algorithm is quite promising.

Let us also cite the Tabu Search algorithm designed by Santos *et al.* [Santos et al. [2014]], in which the total weighted tardiness is now the criterion to minimize. The idea here is to explore a small neighborhood based on job insertion with a dynamic tabu list, being supported by a speedup due to a bookkeeping of solution information. Their algorithm is quite efficient as it improves the best known solutions within short running times.

To solve a flow shop with the so-called family setup times, Lin *et al.* [Lin et al. [2011]] suggest using a multi-start version of the Simulated Annealing approach. The job set can be split into "families", and the setup times occur only between two jobs that belong to different families. Their approach takes advantage of the main properties of the Simulated Annealing (*e.g.* effective convergence, efficient use of memory, and easy implementation) and those of multi-start hill climbing strategies (*e.g.* sufficient diversification, and efficient sampling of the neighborhood solution space). It performs a given number of restarts from different solutions, thus, offering more chances to escape from local optima. The explored neighborhood is based on family jobs swaps and insertions. Their approach was shown to perform extremely well compared to other existing algorithms such as the Mimetic Algorithm of França *et al.* [França et al. [2005]] and the Tabu Search algorithms of Hendizadeh *et al.* [Hendizadeh et al. [2008]].

Finally, we cite the work of Dong *et al.* [Dong et al. [2009]] in which different priority rules are compared such as the NEHT-RB heuristic [Rìos-Mercado and Bard [1998]], and the PB heuristic by Tseng *et al.* [Tseng et al. [2006]]. This study resulted in a classification of which heuristic is best suitable depending on the setup times.

On the other hand, the Migrating Birds Optimization is a rather recent method that has only been applied to a very few problems to date. First designed by Duman *et al.*, its first application was on the Quadratic Assignment Problem [Duman et al. [2012]]. The authors used a 2-interchange neighborhood that consisted in switching two cells in a solution. The promising results it provided made researchers consider it as a solving method for scheduling problems. It was adapted to solve the permutation flow shop scheduling problem by Tongur and Ulker [Tongur and

[1] The algorithm does not manage to improve the current best solution for a certain number of iterations.

[2] To fix the unfeasible solutions.

Ulker [2014]]. They used the same neighborhood structure as in the original algorithm and tested their method on Taillard's benchmark for the flow shop scheduling problem. They obtained good results comparatively to the benchmark's upper bounds. Another application of the Migrating Birds Optimization was for the industrial scheduling problem [Ramanathan and Ulaganathan [2014]]. Although it is not clear whether the authors dealt with the permutation flow shop scheduling problem or with the hybrid flow shop scheduling problem.

4. THE MIGRATING BIRDS OPTIMIZATION APPROACH

The MBO method is a population-based metaheuristic. The basic principle of this approach relies on a "structured" exploration of the neighborhoods of the population's individuals. In what follows, we describe the MBO method, its analogy with natural phenomena, and the neighborhood we used.

4.1 The Algorithm

The MBO method is a metaheuristic inspired from the flight of migrating birds, and especially their V-flight shape[3] that is known for its energy saving properties as illustrated in Figure 1[4].

Fig. 1. V-flight shape.

For a lone bird, the most important factor in achieving lift is the forward speed, the higher it is, the higher the lift. The power needed to generate this lift is called induced power.

When birds fly together in a specific formation, if one of them flaps its wings, the air goes above and below and generates two tubular vortices which contribute to the lifting of a following bird, thus reducing its requirement for induced power and making it spend less energy [Duman et al. [2012]].

This energy saving depends on multiple parameters such as the distance between the birds and their number. Let us mention that the leading bird is the one that spends most of the energy. So, after a while, it will go back to the end of the group leaving its leading place to one of the birds that was behind it.

Before we describe the MBO algorithm as in Figure 2, let us first list its parameters:

- ℓ: number of solutions.
- k: number of neighbors to evaluate for each solution.
- x: number of neighbors that are shared with the next solution.
- t: number of iterations before changing the "leader" solution.
- K: maximum number of iterations.

```
Generate $\ell$ random solutions;
Place them so as to form a V-shape in an
arbitrary way;
$i \leftarrow 0$;
While($i < K$)
{
    For($j = 0;\ j < t;\ j++$)
    {
        Improve the leader solution by
        generating $k$ neighbors;
        $i \leftarrow i + k$;
        For(Every solution $s_r$ of the flight
        (except the one associated with the
        leader))
        {
            Improve it by evaluating $(k - x)$ of
            its neighbors along with $x$ unused
            neighbors of the solution ahead of
            it;
            $i \leftarrow i + (k - x)$;
        }
    }
    Change the leader;
}
Return the best solution of the flight;
```

Fig. 2. Pseudo-code of the MBO [Duman et al. [2012]].

After randomly generating ℓ solutions and placing them in the form of a V-shape[5], the algorithm will first attempt to improve the leader solution by generating k of its neighbors from which the best will eventually replace that solution in case it is better. In our case, the neighborhood that is explored is that of 3-interchange. It is defined by the transformation presented in Section 4.3. Then, the unique sharing mechanism that is specific to this method comes into play. For every solution, other than the one associated with the leader, we only generate $(k - x)$ solutions, and take the remaining x ones from the set of unused neighbors of the solution ahead of it. The best of these $(k - x) + x$ neighbors will then replace the initial solution if it is better, else, the population remains unchanged.

These steps are repeated for t iterations, and then, the leader solution will be changed ; it will go to the end of the right line and be replaced by the solution that was behind it at the right. The algorithm stops after a given number of K generated neighbors.

4.2 Analogy

The Table 1 presents the analogy between the MBO algorithm and the natural phenomenon of bird migration.

[3] That gets its name from its similarity with the letter "V"

[4] https://karthijaygee.wordpress.com/2011/01/02/do-you-know-why-do-birds-fly-in-v-formation

[5] We use a data structure that keeps in memory the positions of the successors and predecessors of every solution

Table 1. Analogy between MBO and the natural phenomenon of migration.

MBO	**Flight of migrating birds**
Nb of solutions ℓ	Number of birds
Nb of evaluated neighbors k	Required induced power *
Nb of shared neighbors x	Distance between the birds
Nb of Iterations t	Flight time of a bird as a leader

* Power that is necessary for a bird to rise.
N.B : Nb=Number

4.3 Neighborhood

The 3-interchange transformation consists in exchanging in a solution S the positions of three jobs, as illustrated by Figure 3, where the jobs at positions 3, 5, and 9 respectively, are exchanged.

S	1	2	**3**	4	**5**	6	7	8	**9**
S'	1	2	**9**	4	**3**	6	7	8	**5**

Fig. 3. 3-interchange transformation.

This neighborhood has been chosen for its capacity to preserve the absolute positions of the jobs in a solution. The goal is, throughout its exploration, to improve the starting solution by modifying the positions of triplets of jobs.

5. EXPERIMENTAL STUDY

The MBO metaheuristic was coded in C++ language, and debugged using Microsoft Visual Studio 2013 on an Dell Inspiron 3537-1648-BLK machine with an intel core^{TM} i7-4500U processor and a RAM of 8 GB. In what follows, we present a description of our testbed, the tuning of the parameters, and the analysis of the obtained results of the experimental study we conducted.

5.1 Description of the testbed

In order to test the algorithm we designed, we have chosen a set of 16 instances for which the numbers of jobs and machines range, from 20 to 50 and from 5 to 20, respectively. These instances, available at `http://soa.iti.es/problem-instances`, are extensions of Taillard's instances of the classical flow shop where setup times have been added and BKS (Best Known Solution) values are available. More details can be found in [Ruiz et al. [2005]]. We took 4 instances (ta001,ta011, ta021, ta031) out of each one of the 4 instance sets that are as follows:

- **SDST10**: setup time values are uniformly drawn between 1 and 9.
- **SDST50**: setup time values are uniformly drawn between 1 and 49.
- **SDST100**: setup time values are uniformly drawn between 1 and 99.
- **SDST125**: setup time values are uniformly drawn between 1 and 124.

The names of the instances specify their sizes as follows:

- ta001 have 20 jobs and 5 machines.
- ta011 have 20 jobs and 10 machines.
- ta021 have 20 jobs and 20 machines.
- ta051 have 50 jobs and 5 machines.

5.2 Parameters fine tuning

The solutions were represented as vectors of size n. The initial population was generated randomly using a constructive heuristic that assigns at each iteration a task j, $j = \overline{1, n}$ to a random position h, $1 \leq h \leq n$, until the vector representing the solution is completed.

Having set K to 10000 as the maximum number of solution evaluations [6] , we have four parameters left to tune. To do this, we proceeded by progressive steps. First, we assigned reference values to the different parameters as follows: $\ell = 21$, $k = 5$, $t = 2$, and $x = 2$. Then, we tuned the parameters one by one. The Figures 4, 5, 6 and 7 show the mean deviation percentages (y axis) that were obtained for each group of instances (x axis (number of jobs x number of machines).

Number of birds:

For the MBO method, it is recommended to take odd values for ℓ [Duman et al. [2012]]. Indeed, in addition to the leading bird, we would ideally want to have the same number of birds on both legs of the V-shape. In their article, the authors of the MBO method advocate modest sizes for the flight. Thus, we tested Algorithm 1 on sizes ranging from 11 to 41.

We run Algorithm 1 twenty times for each instance. Figure 4 pictures the results of the experiment on our testbed we conducted. From the results we got, we have chosen to take 11 as the flight size as it gives the best average results in general.

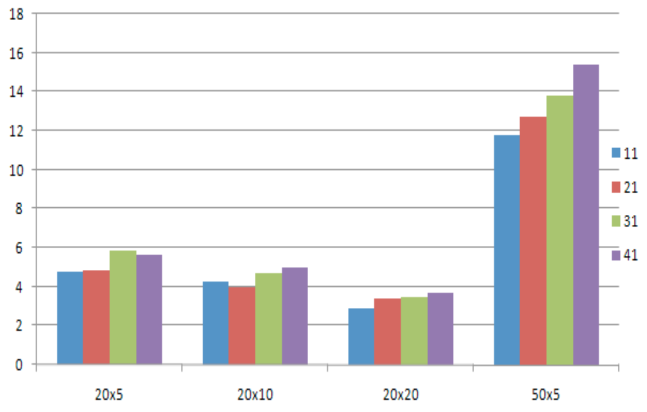

Fig. 4. Effects of the variation of ℓ (the flight size).

Number of evaluated neighbors:

With the limited number of 10000 evaluations, we tested Algorithm 1 on small values of k so as to allow the exploration of the neighborhood of a bigger number of solutions throughout the running of Algorithm 1. As shown in Figure 5, the tests we conducted on the different instances sizes for $k = 5, 10, 15, 20$ made it possible to observe that both 5 produced the best average results.

[6] The stopping criterion for this method is set to the generation of a maximum number of 10000 neighbors.

Following this, we decided to take $k = 5$ for the remaining tests.

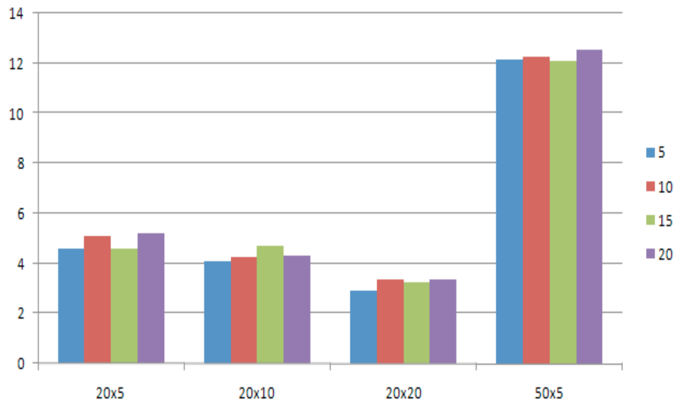

Fig. 5. Effects of the variation of k (the required induced power [Duman et al. [2012]]).

Time spent by a solution as a leader:

Parameter t defines the algorithm running time spent with the same flight configuration before changing the leader bird. This change allows a better exploration of the neighborhoods of the new solutions. Again, it is recommended to take small values for this parameter. Given the maximum number of evaluations and the flight size as well as the number of evaluated neighbors, we run Algorithm 1 another twenty times for each instance for values of t ranging from 1 to 4. The generated results illustrated in Figure 6 support to fix the value of 1.

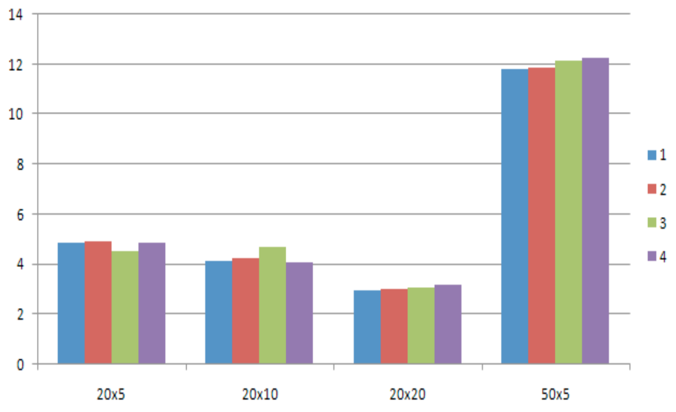

Fig. 6. Effects of the variation of t (Flight time as the leader).

Number of shared neighbors:

It is suggested in [Duman et al. [2012]] to use small values for x such that $x \leq k - 1$. In the series of twenty runs for each instance that were performed on our testbed, we successively took values of $1, 2, 3$ and 4 for x. The results presented in Figure 7 suggest it is preferable to set x to 4.

5.3 Analysis of the numerical results

Algorithm 1 was run five times for each instance. The results are summarized in Table 2; all the computed functions (mean, standard deviation, etc.) are in terms of

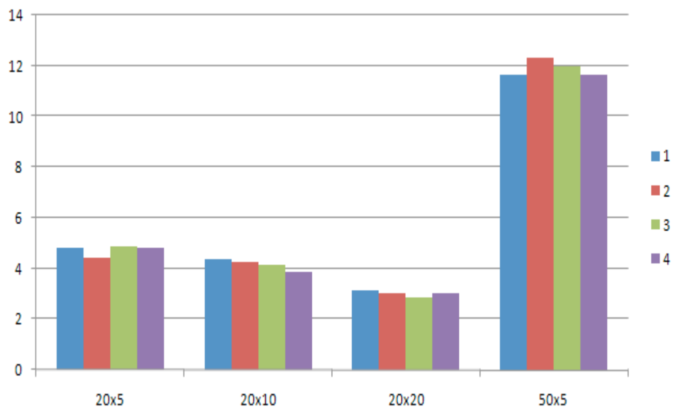

Fig. 7. Effects of the variation of x (distance between birds [Duman et al. [2012]]).

the deviation percentages d relatively to the BKS (Best Known Solution):

$$d = \frac{100(Z_{MBO} - Z^*)}{Z^*},$$

where Z_{MBO} and Z^* are the evaluation of the solution generated by MBO and the Best Known Solution, respectively. The other results are the standard deviation: 4.63, the variance: 21.48, the mean: 5.61, the maximum average: 17.69, and the minimum average: 1.26. The means of computational times in seconds are presented in Table 3.

Table 2. Means of the deviation percentages.

SDST10			
ta001	ta011	ta021	ta031
1,26	2,34	1,62	3,05

SDST50			
ta001	ta011	ta021	ta031
3,62	3,56	2,78	9,95

SDST100			
ta001	ta011	ta021	ta031
6,32	4,24	2,98	15,26

SDST125			
ta001	ta011	ta021	ta031
7,03	4,78	3,25	17,69

Table 3. Means of computational times.

Time (s)			
ta001	ta011	ta021	ta031
509,5	652,35	2617,75	3117,9

According to the results shown in Table 2, it is clear that the MBO method produced results of acceptable quality given the limited number of solution evaluations, which is quite restrictive for a metaheuristic. Indeed, it manages to approximate the best known solution within less than 2% for some instances and does not exceed 18% for the largest ones. The computations take around 8 minutes for the smallest instances and do not exceed an hour (around 51 minutes) for the largest ones.

We have also noticed that the efficiency of the method is not very much affected by the setup times variations nor by the increase of the number of machines. But, it is affected significantly by the increase on the number of jobs. To improve the quality of our algorithm, it would be interesting to assess the quality of other neighborhood structures. We also believe that increasing the number of the evaluations (the stopping criterion) would allow a significant improvement on the quality of the results. For example, if we consider only the largest instances with 50 jobs and 5 machines, we find that the mean is 11.49% and the standard deviation is 5.61. Our computations allowed us to set that the mean could reach 8.5% with a level of significance of α=1%. Also, the mean of all results could be reduced to 4.3% with the same level of significance (The confidence interval of level 99% being [4.27; 6.94]).

6. CONCLUSION

In this paper, we adapted the MBO method for solving the permutation flow shop with sequence dependent setup times. During the design process, we sought to utilize the structural properties of the problem under study so as to obtain a method that would be suitable for its resolution. The MBO method generated good results in general on the instances of the testbed. There is no doubt that the quality of the solutions could be improved by increasing the number of evaluations. Indeed, since our metaheuristic is based on structured neighborhood exploration, a larger number of solution evaluations would allow to improve the latter.

Indeed we noticed that the present version of the MBO provided much better results than another version with only 6000 possible solution evaluations (Although 10000 is still restrictive for a metaheuristic). The results presented above showed an improvement of up to 14.6% of the older results (whose mean was 6.57%).

A way to obtain even better results would be to construct neighborhood structures that would better suit the complexity of the problem. On the other hand, it would be interesting to add other ingredients to the basic MBO, seeking inspiration in the tools previously developed for other metaheuristics. For example, one would consider restructuring the population of the solutions after a given number of iterations without improving the current best solution. That could allow to direct the search towards another area of the solutions space and thus getting a better exploration of the latter. We could also hybridize the MBO with other metaheuristics such as VNS (Variable Neighborhood Search)[Mladenović and [1997]]. This would allow us to enhance both the exploration and exploitation abilities of the method.

REFERENCES

Allahverdi, A., Ng, C., Cheng, T., and Kovalyov, M. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187, 985–1032.

Dong, X., Huang, H., and Chen, P. (2009). Study on heuristics for the permutation flowshop with sequence dependent setup times. In *IEEE International Conference on Information Reuse & Integration*.

Duman, E., Uysal, M., and Alkaya, A.F. (2012). Migrating birds optimization : A new metaheuristic approach and its performance on quadratic assignment problem. *Information Sciences*, 217, 65–77.

França, P., Gupta, J., Mendes, A., Moscato, P., and Veltink, K. (2005). Evolutionary algorithms for scheduling a flow shop manufacturing cell with sequence dependent family setups. *Computers and Industrial Engineering*, 48(3), 491–506.

Gupta, J. (1986). Flowshop schedules with sequence dependent setup times. *Journal of the Operational Research Society of Japan*, 29 (3), 206–219.

Hendizadeh, S., Faramarzi, H., Mansouri, S., Gupta, J., and ElMekkawy, T. (2008). Metaheuristics for scheduling a flowline manufacturing cell with sequence dependent family setup times. *International Journal of Production Economics*, 111(2), 593–605.

Kumar, G. and Singhal, S. (2013). Genetic algorithm optimization of flow shop scheduling problem with sequence dependent setup time and lot splitting. *International Journal of Engineering, Business and Enterprise Applications*, 4(1), 62–71.

Lin, S.W., Ying, K.C., Lu, C.C., and Gupta, J. (2011). Applying multi-start simulated annealing to schedule a flowline manufacturing cell with sequence dependent family setup times. *International Journal of Production Economics*, 130, 246–254.

Mladenović, N. and (1997), P.H. (1997). Variable neighborhood search for the p-median. *Location Sci.*, 5, 207–226.

Pinedo, M. (2002). *Scheduling : Theory, algorithms and systems.* Prentice Hall.

Ramanathan, L. and Ulaganathan, K. (2014). Nature-inspired metaheuristic optimization technique-migrating birds optimization in industrial scheduling problem. *SSRG International Journal of Industrial Engineering (SSRG-IJIE)*, 1(3), 1–6.

Rìos-Mercado, R. and Bard, J. (1998). Heuristics for the flow line problem with setup costs. *European Journal of Operational Research*, 110, 76–98.

Ruiz, R., Maroto, C., and Alcaraz, J. (2005). Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. *European Journal of Operational Research*, 165, 34–54.

Ruiz, R. and Stutzle, T. (2008). An iterated greedy algorithm for the flowshop problem with sequence dependent setup times. *European Journal of Operational Research*, 187, 1143–1159.

Santos, N., Pedroso, J., and Rebelo, R. (2014). A tabu search for the permutation flow shop problem with sequence dependent setup times. *International Journal of Data Analysis Techniques and Strategies (IJDATS)*, 6 (3), 275 – 285.

Talbi, E.G. (2009). *Metaheuristics, from design to implementation.* John Wiley & Sons, Inc.

Tongur, V. and Ulker, E. (2014). Migrating birds optimization for flow shop sequencing problem. *Journal of Computer and Communications*, 2, 142–147.

Tseng, F., Gupta, J., and Stafford, J. (2006). A penalty-based heuristic algorithm for the permutation flow shop scheduling problem with sequence-dependent set-up times. *Journal of the Operational Research Society*, 57, 541–551.