DISCRETE BACTERIAL MEMETIC ALGORITHM FOR THE FLOW SHOP SCHEDULING PROBLEM

Anita AGÁRDI[1,*]

[1] *University of Miskolc, Institute of Informatics, Egyetemváros, 3515, Miskolc, Hungary*
Corresponding author, Anita Agárdi, e-mail: anita.agardi@uni-miskolc.hu

ABSTRACT*: The article investigates a production scheduling task, the Flow Shop Scheduling Problem. Since this task is an Np-hard problem, it has already been solved using several metaheuristics. The article examines the effectiveness of the Discrete Bacterial Memetic Algorithm for Flow Shop Scheduling. The Discrete Bacterial Memetic Algorithm is a population-based algorithm that maintains a set of solutions during iterations. It improves the solutions iteratively, thus creating new solutions. The article also presents the test results and the analysis of the search space of the problem. The test results were also compared with other metaheuristic algorithms, such as the Hormone Modulation Mechanism Flower Pollination Algorithm, Hybrid Genetic Algorithm, Improved Iterated Greedy Algorithm, and Invasive Weed Optimization Algorithm. Based on the test results, the Discrete Bacterial Memetic Algorithm can be effectively applied to solve the Flow Shop Scheduling problem.*

KEYWORDS*: Discrete Bacterial Memetic Algorithm, Flow Shop Scheduling, fitness landscape analysis*

1. INTRODUCTION

In the field of Industry 4.0, cost-effective and time-efficient production solutions are very important. A common production scheduling optimization is the Flow Shop Scheduling (FSS) [1]. The FSS has many versions which have appeared over the years. In the case of the classical problem, the number of machines and jobs is given. Each job must be performed on each machine, but exactly once. If the machines have started a job, they must finish it, and only then can they start another job. Over the years, several versions of the Flow Shop Scheduling have appeared, which have been adapted to real demands over the years. In the following, the paper highlights the most common Flow Shop Scheduling versions.

In most of the problems, the objective is the makespan minimization [2], and the machines have availability constraints [2]. We can also define Flow Shop Scheduling with stochastic parameters [3], where some parameters are stochastic, e.g. the number of machines, the number of jobs, and the time to complete the jobs on each machine. In the literature, we can also find just-in-time scheduling [4] or blocking FSS [5], where jobs can be blocked, i.e. if a machine has started a job, it can block and do another job in the meantime, and if it is ready, then it returns to the unfinished job. Only a limited number of machines may be available at a given time [6]. Another version of FSS is the industrial multiprocessor [7] problem, but recirculation [7] is also possible. The machines can have the same or different [8] parameters. They can differ, for example, in how long they can complete a given job, or whether they can interrupt the given job and start another job in the meantime, or whether they have to complete the given job and only then can they do another job. Permutation Flow Shop Scheduling [9] is a task when all machines have the same work order. Flexible Flow Shop Scheduling [10] is a problem when the number and configuration of machines can be changed due to different jobs and work groups. Fault-Tolerant Flow Shop Scheduling [11] considers that machines can fail. In this case, outages resulting from errors must also be minimized. In the case of Parallel Flow Shop Scheduling [12], jobs can be done parallel, several machines of the same type can work at one station at the same time, so a job does not go through all machines, but all machine types. In the case of No-Wait Flow Shop Scheduling [13], the goal is to eliminate the waiting time between stations. Closed-loop FSS [14] is a problem when the flow of work forms a closed circle, i.e. certain stations (machines) can be revisited cyclically. According to the preemptive problem [15], a task can be stopped temporarily and then resumed later. Based on Weighted FSS [16], it assigns weights and priorities to jobs or machines. Based on Batch Processing [17], products and parts are processed together. If energy consumption is also considered, we are talking about an Energy-Efficient [18] problem.

The Discrete Bacterial Memetic Algorithm [19] is a recently developed discrete optimization algorithm. It combines the Bacterial Algorithm and the Memetic Algorithm.

The article is further structured as follows: the Discrete Bacterial Memetic Algorithm and the Flow Shop Scheduling are presented in Section 2. Then the test results and their comparison with other heuristic algorithms are detailed. In addition, the fitness landscape results are also presented. The last section is the conclusions and future research direction section.

2. MATERIALS AND METHODS

In this section, the Flow Shop Scheduling Problem is presented first, followed by the applied Discrete Bacterial Memetic algorithm.

In the article, the classic Flow Shop Scheduling [1] problem was investigated. The numbers of machines and jobs are given based on the task. All jobs must be done on all machines. The jobs move through each machine in a specific order. The ordering should be done in such a way that the makespan will be minimal. There are processing times for individual jobs and machines, which vary from machine to machine and job to job. If a job has been started by a given machine, it must be finished, the machine cannot start the other job, and the job must wait.

2.1. Discrete Bacterial Memetic Algorithm

The Discrete Bacterial Memetic Algorithm (Discrete Bacterial Memetic Algorithm) [19] combines the Bacterial Algorithm and the Memetic algorithm for discrete optimization tasks. The algorithm maintains a population of solutions and continuously improves individual elements of the population using a heuristic search operator. It uses search operators such as bacterial mutation, local search (the paper used the 2-opt), and gene transfer operator. The algorithm initially generates the population randomly. The population consists of N_{ind} number of elements. It then continues bacterial mutation, local search, and gene transfer until the termination condition. The parameter of the bacterial mutation operator is the population itself, N_{clones} is the number of newly created possible solutions, and I_{seg} is the number of segments. The local search (2-opt) creates a single possible solution (an element of the population). The gene transfer parameter operates on the entire population, N_{inf} is the number of infections and I_{trans} is the length of the gene transfer.

The algorithm consists of the following steps:

- Generating random solutions that will be the elements of the population.
- Performing a bacterial mutation on the elements of the population. The parameters of the bacterial mutation are as follows: Population, N_{clones}, I_{seg}.
- Performing local search (2-opt) on the individual solutions of the population (on individual elements of the population).
- Performing gene transfer on the elements of the population. Gene transfer has the following parameters: Population, N_{inf}, I_{trans}.
- The steps 2-4. are repeated until the stop condition is not met.
- The algorithm returns with the best solution.

2.1.1. Bacterial mutation

Bacterial mutation [20] performs loose segment mutation and coherent segment mutation on the entire population. The input parameters of the bacterial mutation are as follows: Population, N_{clones}, I_{seg}, where N_{clones} is the number of clones created from each bacteria, and I_{seg} is the number of segments.

The steps of bacterial mutation are as follows:

- Generating a random number in the interval [0,1], denoting this number by r.
- Selecting the element i from the population, where i is the current index.
- Creating N_{clones} clones from the population element (from a possible solution). This practically means creating N_{clones} of solutions that are identical to the population i. with its element.
- If $r \leq 0.9$, then the element p is divided into coherent segments of length I_{seg}.
- If $r > 0.9$, the element p is divided into I_{seg} length loose segments (loose segments).
- Modifying the N_clones clones based on the segments.
- The element i of the population will be the best element between clones and p.
- The whole process is carried out until the algorithm goes through the entire population.

Figure 1. illustrates the bacterial mutation, where the original bacterium and 5 clones are presented. The length of the segment is 3. It goes through all possible solutions.

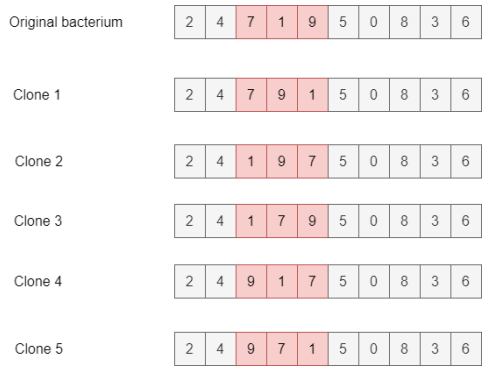

Fig. 1 Bacterial mutation

In Figure 1 it can be seen a permutation example that contains numbers in the interval 0-9. The length of the segment is 3 and in the example, we can see all possible solutions.

In the following, the paper will present the differences between the coherent segment and the loose segment.

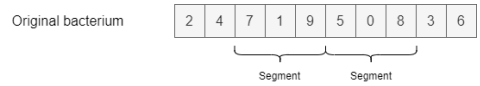

Fig. 2 Coherent segment mutation

In Figure 2 it can be seen that the paper divided the original solution (bacterium) into 3 long segments.

The following figure shows the loose segment mutation.

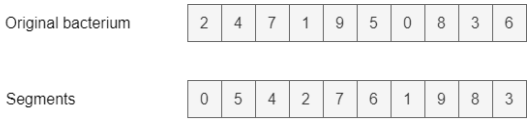

Fig. 3 Loose segment mutation

During the loose segment mutation, the segments will also be permutations. These are the sequences of elements found in the original bacterium. So, the elements of the segment will be the indices of the original bacteria, and the values of the indices will be the new bacteria (solutions).

2.1.2. Gene transfer

Gene transfer [20] is performed on the entire population. First, it ranks the elements of the population. It then divides the ranked population into two parts, an upper part, and a lower part. Then the algorithm performs various transformations N_{inf} times (and creates new solutions) on individual elements of the population.

The algorithm consists of the following steps:

- The population is ranked, and the ranked elements are divided into two parts: top and bottom part.
- Executing the following steps N_{inf} times:
  - Selecting one bacterium (possible solution) from the upper and lower parts, these will be p_{source} and $p_{destination}$.
  - A segment is randomly selected from the p_{source} solution, the length of which is I_{trans}.
  - The segment is copied into the $p_{destination}$ bacterium (solution) at a randomly selected location.
  - Duplicates are deleted from the $p_{destination}$ bacterium, so a valid solution is created.

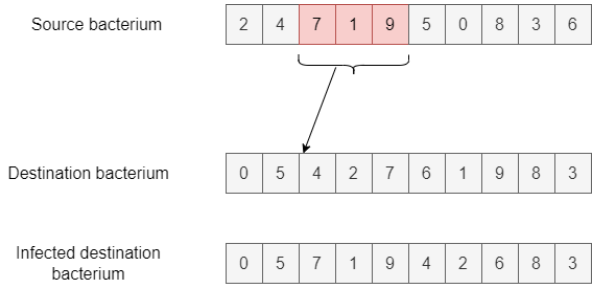

Fig. 4 Gene transfer

The figure 4 illustrates the gene transfer procedure, where the destination bacterium is infected by the source bacterium.

3. TEST RESULTS

In this section, the test results are presented. The article first presents the test runs on the Taillard [21] dataset. Table 1 contains the fitness value of the Discrete Bacterial Memetic Algorithm. In addition, the relative performances compared to four other algorithms. These algorithms are HMM-PFA (Hormone Modulation Mechanism Flower Pollination Algorithm [22]), HGA (Hybrid Genetic Algorithm [23]), IIGA (Improved Iterated Greedy Algorithm [23]), IWO (Invasive Weed Optimization Algorithm [24]). Except for IWO, there were data from Ta001 to Ta030 for the other algorithms. In

the case of IWO, there were only running results for Ta001, Ta011, and Ta021.

Table 1. Test results for the Taillard dataset

		Relative performance			
Instance	**DBMA**	**HMM-PFA %**	**HGA %**	**IIGA %**	**IWO %**
Ta001	1339	110.98	108.22	110.98	103.73
Ta002	1428	107.00	102.24	107.00	-
Ta003	1219	119.77	113.70	119.77	-
Ta004	1406	112.94	108.18	112.94	-
Ta005	1340	108.13	104.70	108.13	-
Ta006	1275	116.16	112.16	116.16	-
Ta007	1320	112.35	110.68	112.35	-
Ta008	1313	112.87	109.14	112.87	-
Ta009	1364	107.70	102.49	107.70	-
Ta010	1217	113.15	108.79	113.15	-
Ta011	1755	116.47	111.40	114.59	125.75
Ta012	1856	116.70	114.39	116.70	-
Ta013	1682	115.34	113.67	115.34	-
Ta014	1528	118.52	116.62	118.52	-
Ta015	1622	119.17	119.17	119.17	-
Ta016	1555	121.67	117.49	121.67	-
Ta017	1645	119.33	118.18	119.33	-
Ta018	1716	119.87	116.90	119.87	-
Ta019	1728	114.18	110.42	114.18	-
Ta020	1747	117.40	114.54	117.40	-
Ta021	2512	118.35	115.92	118.35	128.42
Ta022	2301	123.95	120.82	112.21	-
Ta023	2521	119.52	115.91	119.52	-
Ta024	2430	123.50	122.10	123.50	-
Ta025	2503	119.98	117.98	119.98	-
Ta026	2402	124.81	121.07	124.40	-
Ta027	2511	121.55	118.28	121.55	-
Ta028	2411	117.75	114.60	117.75	-
Ta029	2468	121.92	120.42	121.92	-
Ta030	2423	122.95	120.47	122.95	-

Table 1 presents, that the DBMA gave 1339 fitness result for the Ta001 dataset. It was 10% better than HMM-PFA and IIGA, 8% better than HGA, and 3% better than IWO. For Ta002, DBMA gave 1428 result, which was 7% better than HMM-PFA and IIGA and 2% better than HGA. For Ta003, DBMA gave 1219 result, which was 19% better than HMM-PFA and IIGA, and 13% better

than HGA. For Ta004, DBMA resulted in a fitness value of 1406. This was 12% better than HMM-PFA and 8% better than HGA. DBMA gave 1340 result for Ta005, which was 8% better than HMM-PFA and IIGA and 4% better than HGA. The DBMA algorithm gave even better results for a larger data set. For the Ta011, the DBMA resulted in a fitness value of 1755, which was 16% better than HMM-PFA, 11% better than HGA, 14% better than IIGA, and 25% better than IWO. The algorithm gave 1856 fitness value for Ta012. It was 16% better than HMM-PFA, 14% better than HGA, and 16% better than IIGA. If we examine Ta021, we get even better results. The fitness value of DBMA was 2512, which was 18% better than HMM-PFA, 15% better than HGA, 18% better than IIGA, and 28% better than IWO. For Ta022, there were 2301 fitness value, which was 23% better than HMM-PFA, 20% better than HGA, 12% better than IIGA, and 12% better than IWO.

Table 2. Test results summary for Taillard dataset

Algorithm	Number of data sets (on which the comparisons were made)	Number of better results
HMM-PFA	30	30
HGA	30	30
IIGA	30	30
IWO	3	3

Table 2 summarizes how many data sets were tested, and how many of these data sets the DBMA algorithm was better at. Except for the IWO algorithm, the other algorithms could be run on 30 benchmark data sets. In all cases, the DBMA algorithm gave a better result than the given comparison algorithms.

Table 3. Fitness landscape analysis

		Ta001		Ta002		Ta003		Ta004		Ta005	
	Distance	LB	UB	LB	UB	LB	UB	LB	UB	LB	UB
Fitness values		1360	1381	1437	1474	1240	1500	1495	1597	1385	1385
Average of fitness distances	Fitness	6.3	14.7	6.3	24.7	28.7	233.3	10.2	91.8	0	0
Average of Hamming distances	Hamming	4.8	11.2	9.8	17.6	3.5	17.1	1.7	15.3	0	0
Average of basic swap sequence distances	BSS	4.2	9.8	8.4	15.6	2.9	14.3	1.5	13.5	0	0
Cost density		4.0	7.0	2.0	5.0	1.0	9.0	1.0	10.0	10.0	10.0

Based on the fitness landscape analysis, the results given by the DBMA iterations were analyzed. Regarding the iterations, the paper used fewer iterations here (10 iterations) than in the comparison with the other algorithms, because the goal of the fitness landscape analysis is the analytical analysis. Table 3 contains the metrics, the name of the benchmark dataset, the LB (lower bound) and UB (upper bound) values. The following metrics were used [25]:

- Fitness values
- Average of fitness distances
- Average of Hamming distances
- Average of basic swap sequence distances
- Cost density

The results of the Discrete Bacterial Memetic (DBMA) algorithm for Ta001 were between 1381 and 1360 fitness values, for Ta002 between 1474 and 1437, for Ta003 between 1500 and 1240, for Ta004 between 1597 and 1495, while for Ta005 it was 1385. A large improvement is found in the Ta003. The average of the fitness distances is also the highest here (Ta003). For Ta001 the fitness distances are low, for Ta002 it was a bit higher, while for Ta003 the upper limit is very high, as is true for Ta004. For Ta005, all distance values are 0, since the individual iterations gave identical solutions. The Hamming and BSS distances are also the smallest for Ta001 (except for Ta005) and large at Ta002, Ta003, and Ta004. Overall, it can be said that the Discrete Bacterial Memetic Algorithm can

be an effective algorithm based on the fitness landscape analysis.

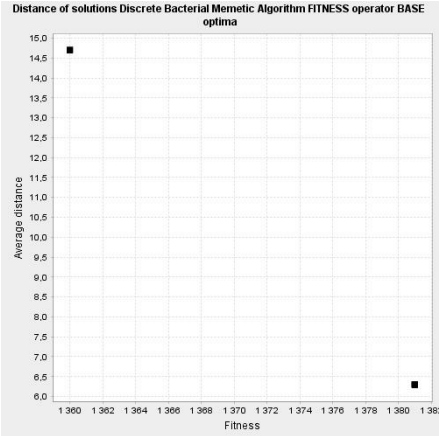

Fig. 6 Fitness distances for Ta001

The fitness distances for Ta001 are illustrated in Figure 6. The average number of distances for a fitness value of 1380 is 14.5, and the average number of distances for a fitness value of 1380 is 6.5.

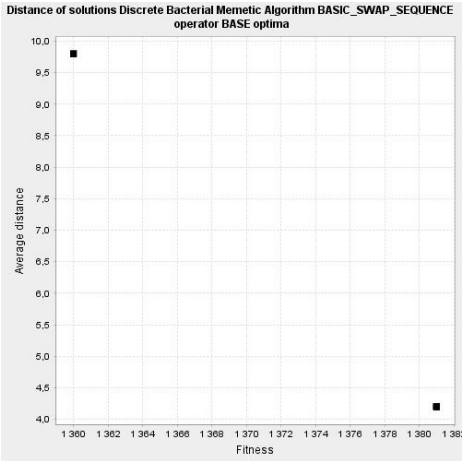

Fig. 7 Basic swap sequence distances for Ta001

Figure 7 shows the basic swap sequence distances. For a fitness value of 1380, the average distance is 4, and for a fitness value of 1380, it is 9.5.

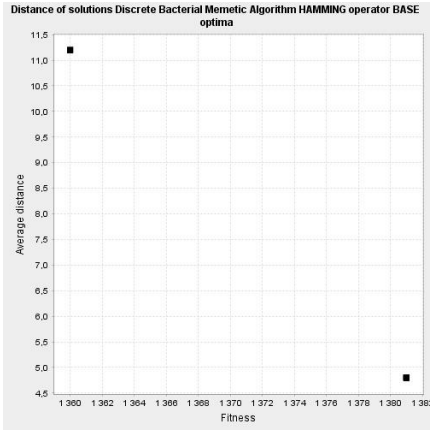

Fig. 8 Hamming distances for Ta001

Figure 8 shows the Hamming distances for the Ta001 dataset. For 1360 fitness values, the average of the distances is 5, for 1360 fitness values, the average of the Hamming distances is 11.

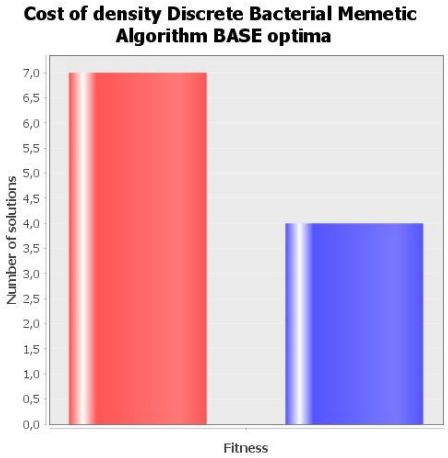

Fig. 9 Cost density for Ta001

Figure 9 illustrates the cost density values. The solutions gave two different fitness values.

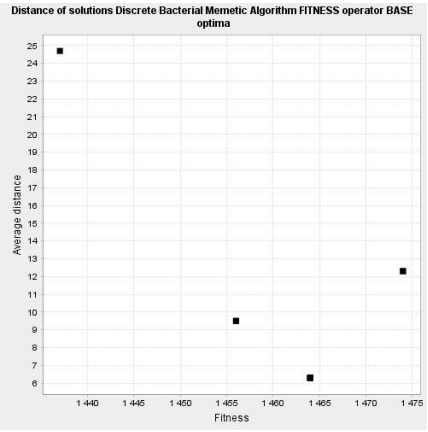

Fig. 10 Fitness distances for Ta005

Figure 10 illustrates the fitness distances for the Ta005 dataset. For a fitness value of 1485 the average fitness distance is 6, for a fitness value of 1455 it is 10, for a fitness value of 1475 it is 13, and for a fitness value of 1440 it is 25.

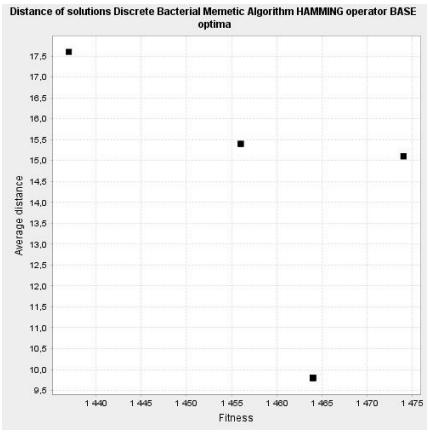

Fig. 10 Hamming distances for Ta005

Figure 10 examines the effectiveness of DBMA for Ta005. The average Hamming distance was 10 for the 1465 fitness value, 15 for the 1475 fitness value, 15.5 for the 1455 fitness value, and 17.5 for the 1440 fitness value.

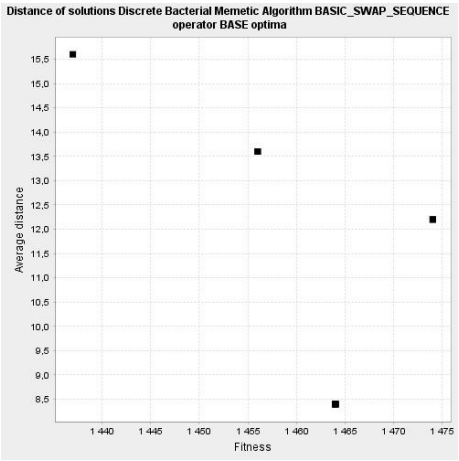

Fig. 11 Basic Swap Sequence distances for Ta005

Based on Figure 11, the basic swap sequence distances are as follows. For a fitness value of 1405, the average BSS distances are 8.5, for a fitness value of 1475, the distance is 12, for a value of 1455 it is 13.5, and for a value of 1440 it is 15.5.

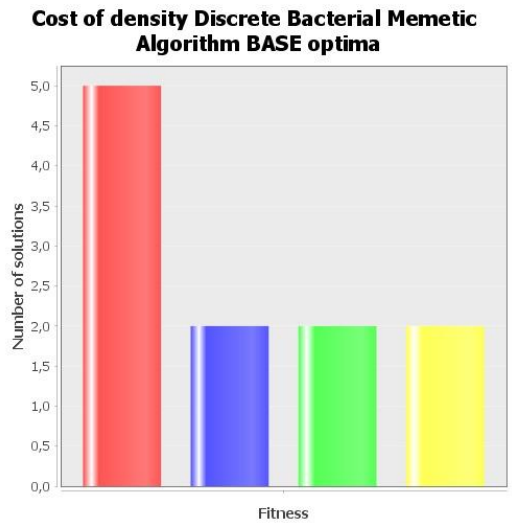

Fig. 12 Cost density for Ta005

Based on Figure 12, each DBMA iteration resulted in 4 different solutions for the Ta005 dataset. The number of solutions belonging to one fitness value was 5 and for the other fitness values, the cost density is 2.

4. CONCLUSION AND FUTURE RESEARCH

The article presented a solution for the Flow Shop Scheduling (FSS) problem. FSS was solved using the Discrete Bacterial Memetic Algorithm. DBMA is an algorithm that performs operations on a population of solutions. It modifies current solutions with operators such as bacterial mutation, loose segment mutation, coherent segment mutation, and 2-opt local search. The article compared the results of the algorithm test with four other algorithms, during which it was revealed the effectiveness of DBMA. The paper presented the efficiency of the DBMA with the fitness landscape analysis. During the analytical fitness landscape analysis, fitness values, fitness distances, Hamming and basic swap sequence distances, as well as cost density values were examined. Based on the fitness landscape analysis, the algorithm also proved to be effective.

Future research direction is the analysis of Flow Shop Scheduling data series with other metaheuristics, for example, Genetic Algorithm, Simulated Annealing, Ant Colony Optimization algorithms, etc. Another research area direction is the development of new hybrid algorithms, the discretization of continuous optimization algorithms, and the verification of their effectiveness for the FSS problem.

5. ACKNOWLEDGEMENTS

„Supported by the ÚNKP-23-4-II. New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.”

6. REFERENCES

[1] Ruiz, R., & Vázquez-Rodríguez, J. A. (2010). The hybrid flow shop scheduling problem. European journal of operational research, 205(1), 1-18.

[2] Aggoune, R. (2004). Minimizing the makespan for the flow shop scheduling problem with availability constraints. European Journal of Operational Research, 153(3), 534-543.

[3] Gourgand, M., Grangeon, N., & Norre, S. (2003). A contribution to the stochastic flow shop scheduling problem. European Journal of Operational Research, 151(2), 415-433.

[4] Shabtay, D. (2012). The just-in-time scheduling problem in a flow-shop scheduling system. European Journal of Operational Research, 216(3), 521-532.

[5] Miyata, H. H., & Nagano, M. S. (2019). The blocking flow shop scheduling problem: A comprehensive and conceptual review. Expert Systems with Applications, 137, 130-156.

[6] Aggoune, R., & Portmann, M. C. (2006). Flow shop scheduling problem with limited machine availability: A heuristic approach. International journal of production economics, 99(1-2), 4-15.

[7] Bertel, S., & Billaut, J. C. (2004). A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. European Journal of Operational Research, 159(3), 651-662.

[8] Benavides, A. J., Ritt, M., & Miralles, C. (2014). Flow shop scheduling with heterogeneous workers. European Journal of Operational Research, 237(2), 713-720.

[9] Framinan, J. M., Gupta, J. N., & Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. Journal of the Operational Research Society, 55, 1243-1255.

[10] Wang, H. (2005). Flexible flow shop scheduling: optimum, heuristics and artificial intelligence solutions. Expert systems, 22(2), 78-85.

[11] Samal, A. K., Mall, R., & Tripathy, C. (2014). Fault tolerant scheduling of hard real-time tasks on multiprocessor system using a hybrid genetic algorithm. Swarm and Evolutionary Computation, 14, 92-105.

[12] Ribas, I., Companys, R., & Tort-Martorell, X. (2017). Efficient heuristics for the parallel blocking flow shop scheduling problem. Expert Systems with Applications, 74, 41-54.

[13] Pan, Q. K., Wang, L., & Qian, B. (2009). A novel differential evolution algorithm for bi-criteria no-wait flow shop scheduling problems. Computers & Operations Research, 36(8), 2498-2511.

[14] Zhao, F., He, X., & Wang, L. (2020). A two-stage cooperative evolutionary algorithm with problem-specific knowledge for energy-efficient scheduling of no-wait flow-shop problem. IEEE transactions on cybernetics, 51(11), 5291-5303.

[15] Khorasanian, D., & Moslehi, G. (2017). Two-machine flow shop scheduling problem with blocking, multi-task flexibility of the first machine, and preemption. Computers & Operations Research, 79, 94-108.

[16] Nejati, M., Mahdavi, I., Hassanzadeh, R., Mahdavi-Amiri, N., & Mojarad, M. (2014). Multi-job lot streaming to minimize the weighted completion time in a hybrid flow shop scheduling problem with work shift constraint. The International Journal of Advanced Manufacturing Technology, 70, 501-514.

[17] Zhang, C., Shi, Z., Huang, Z., Wu, Y., & Shi, L. (2017). Flow shop scheduling with a batch processor and limited buffer. International Journal of Production Research, 55(11), 3217-3233.

[18] Yan, J., Li, L., Zhao, F., Zhang, F., & Zhao, Q. (2016). A multi-level optimization approach for energy-efficient flexible flow shop scheduling. Journal of Cleaner Production, 137, 1543-1552.

[19] Kóczy, L. T., Földesi, P., & Tüű-Szabó, B. (2017). An effective discrete bacterial memetic evolutionary algorithm for the traveling salesman problem. International Journal of Intelligent Systems, 32(8), 862-876.

[20] Tüű-Szabó, B., Földesi, P., & Kóczy, L. T. (2020). An Efficient Evolutionary Metaheuristic for the Traveling Repairman (Minimum Latency) Problem. International Journal of Computational Intelligence Systems, 13(1), 781-793.

[21] E. Taillard, "Benchmarks for basic scheduling problems", EJOR 64(2):278-285, 1993.

[22] Qu, C., Fu, Y., Yi, Z., & Tan, J. (2018). Solutions to no-wait flow shop scheduling problem using the flower pollination algorithm based on the hormone modulation mechanism. Complexity, 2018.

[23] Wei, H., Li, S., Jiang, H., Hu, J., & Hu, J. (2018). Hybrid genetic simulated annealing algorithm for improved flow shop scheduling with makespan criterion. Applied Sciences, 8(12), 2621.

[24] Zhou, Y., Chen, H., & Zhou, G. (2014). Invasive weed optimization algorithm for optimization no-idle flow shop scheduling problem. Neurocomputing, 137, 285-292.

[25] Agárdi, A., Kovács, L., & Bányai, T. (2021). The fitness landscape analysis of the ant colony system algorithm in solving a vehicle routing problem. ACADEMIC JOURNAL OF MANUFACTURING ENGINEERING, 19(2).