symmetry

MDPI

Article

A Hybrid Discrete Bacterial Memetic Algorithm with Simulated Annealing for Optimization of the Flow Shop Scheduling Problem

Anita Agárdi [1], Károly Nehéz [1], Olivér Hornyák [1] and László T. Kóczy [2,3,*]

[1] Institute of Information Science, University of Miskolc, 3515 Miskolc, Hungary; agardianita@iit.uni-miskolc.hu (A.A.); aitnehez@uni-miskolc.hu (K.N.); oliver.hornyak@uni-miskolc.hu (O.H.)
[2] Department of Information Technology, Széchenyi István University, 9026 Győr, Hungary
[3] Department of Telecommunication and Media Informatics, Budapest University of Technology and Economics, 1111 Budapest, Hungary
* Correspondence: koczy@tmit.bme.hu or koczy@tmit.bme.hul.com

Abstract: This paper deals with the flow shop scheduling problem. To find the optimal solution is an NP-hard problem. The paper reviews some algorithms from the literature and applies a benchmark dataset to evaluate their efficiency. In this research work, the discrete bacterial memetic evolutionary algorithm (DBMEA) as a global searcher was investigated. The proposed algorithm improves the local search by applying the simulated annealing algorithm (SA). This paper presents the experimental results of solving the no-idle flow shop scheduling problem. To compare the proposed algorithm with other researchers' work, a benchmark problem set was used. The calculated makespan times were compared against the best-known solutions in the literature. The proposed hybrid algorithm has provided better results than methods using genetic algorithm variants, thus it is a major improvement for the memetic algorithm family solving production scheduling problems.

Keywords: discrete bacterial memetic evolutionary algorithm; simulated annealing; flow shop scheduling problem

check for updates

Citation: Agárdi, A.; Nehéz, K.; Hornyák, O.; Kóczy, L.T. A Hybrid Discrete Bacterial Memetic Algorithm with Simulated Annealing for Optimization of the Flow Shop Scheduling Problem. *Symmetry* **2021**, *13*, 1131. https://doi.org/10.3390/sym13071131

Academic Editor: José Carlos R. Alcantud

Received: 19 April 2021
Accepted: 11 June 2021
Published: 24 June 2021

1. Introduction

This paper investigates whether a new meta-heuristic proposed by the authors, an improved and modified version of the discrete bacterial memetic evolutionary algorithm, is capable of solving the flow shop scheduling problem (FSSP) in an efficient way, possibly in a more efficient way than other approaches proposed by other authors. FSSP was first published in 1954 by Johnson [1]. Although there exist exact solution algorithms, they are not feasible for the large-sized scheduling problem, as FSSP is an NP-hard problem. Many researchers have addressed the problem since its introduction. The overview of the proposed solution is as follows.

Wei et al. [2] introduced a hybrid genetic simulated annealing algorithm, which combines the individual steps and operators of the simulated annealing and the genetic algorithm (HSGA). In their solution, the genetic algorithm (GA) finds a new optimal solution, and the simulated annealing attempts to improve that solution. To compare their solution, the widely used Taillard data set [3] was used, which is a reference benchmark for FSSP. The Taillard data set contains benchmark data between 20 and 500 jobs and between 5 and 20 machines for the flow shop scheduling problem. The following state-of-the-art algorithms were compared in [2]: memetic algorithm, iterated greedy algorithm with a referenced insertion scheme, hybrid genetic algorithm (i.e., GA with improved local search method that searches in a larger neighborhood), improved iterated greedy algorithm (using Tabu mechanism to escape from local minima), and discrete self-organizing migrating algorithm. Their hybrid genetic simulated annealing algorithm proved to be the best for the test data set.

Another hybrid genetic algorithm, which combined two local search methods with GA, was introduced by Tseng et al. [4]. Their hybrid genetic algorithm is compared with the primitive genetic algorithm, genetic algorithm+ insertion search, and genetic algorithm+ insertion search with cut-and-repair. Their algorithm found better results than the reference benchmark problems.

The results for the flow shop problem were also compared with the following algorithms: Johnson's algorithm, Nawaz–Enscore–Ham (NEH) heuristic, iterated local search algorithm, and iterated greedy algorithm. Based on the paper of Belabid et al. [5], the NEH heuristic gives better results than the others.

The flow shop scheduling problem has been solved with a relatively new algorithm called the flower pollination algorithm by Qu et al [6]. Compared with other heuristic algorithms, such as Tabu-based reconstruction strategy (TMIIG), discrete particle swarm optimization algorithm (DPSO), improved iterated greedy algorithm (IIGA), effective hybrid particle swarm optimization (HPSO), hybrid differential evolution approach (HDE), and genetic algorithm (GA), the flower pollination approach was the most efficient one.

An invasive weed optimization (IWO) algorithm was introduced in [7] for FSSP. The authors also used the Taillard benchmark to test the efficiency of their algorithm. The algorithm was compared with Nawaz–Enscore–Ham (NEH) algorithm. Their solution obtained better makespan than the NEH algorithm for every instance of 12 different scale benchmarks. It has proved to be better in terms of both final accuracy and convergence. The reason is that the global exploration in [7] based on normal distribution is better than the other algorithms.

Simulated annealing is another efficient optimization algorithm; there are several articles on the topic that solve the flow shop scheduling problem with this algorithm, for example, Ogbu and Smith [8], Lin et al. [9], and Aurich et al. [10].

The following section formulates the FSPP problem itself, and then an overview is given on the state-of-the-art of approximate solving algorithms. Next, the proposed new memetic algorithm is presented, which may be considered as a further development and improvement of the discrete bacterial memetic evolutionary algorithm, an approach that has already been successfully applied to the solution of other discrete NP-hard problems. The authors executed some benchmark-based tests and compared the results with the algorithms discussed in the overview of the state-of-the-art. The table of the results with comparisons and explanations and, finally, some concluding notes are presented in the last section.

2. Formulation of the Flow Shop Scheduling Problem

In the case of the flow shop scheduling problem [2], n jobs and m machines are given with the following constraints: (1) The jobs have the same processing route. (2) Each job must be run on all machines exactly once. (3) All jobs and machines must be ready to work at time zero. (3) All jobs have m processing steps. (4) Neither machines nor jobs have priority. (5) A single machine can do a single job at a time. (6) If a job is started on a machine, no process can interfere. During processing, all jobs must follow the first in–first out (FIFO) rule. The mathematical model of the flow shop scheduling problem can be written in the following way:

$J = \{1, 2, \dots, n\}$: Set of jobs
$M = \{1, 2, \dots, m\}$: Set of machines
$p_{i,j}$: processing time of job i on machine j
$S_{i,j}$ starting time when job i is processed on machine j
$C_{i,j}$ finishing time when job i is processed on machine j
$\pi = \{\pi_1, \pi_2, \dots, \pi_n\}$ the job sequence
$C_{max}(\pi)$: the makespan of a job sequence

The objective function is the minimization of the makespan. Makespan is the overall length of finishing the job sequence. The objective function can be written in the following way:

$$(C_{max}(\pi))\rightarrow\min \tag{1}$$

where the following conditions must be met:

(1) A job can only be processed by one machine at a time:

$$\text{If } {X^t}_{i,j} > 0 \text{ then } p_{i,k} = 0\ \forall k \in [1, 2, \ldots, n],\ k \neq i \tag{2}$$

(2) A machine can only process a single job at a time:

$$\text{If } p_{i,j} > 0 \text{ then } p_{k,j} = 0\ \forall k \in [1, 2, \ldots, m],\ k \neq j \tag{3}$$

(3) The next job cannot start until the current job is completed on the given machine:

$$C_{i,j} \leq S_{i+1,j} \tag{4}$$

(4) If machine $j + 1$ is not ready, the job will delay at machine j until machine $j + 1$ will be free:

$$C_{i,j} \geq C_{i-1,j+1} \tag{5}$$

(5) For any job i, the completion time is the starting and processing time on machine j:

$$C_{i,j} = S_{i,j} + p_{i,j} \tag{6}$$

(6) The makespan of the schedule is the time when the last job finishes on the last machine:

$$C_{max}(\pi) = C_{\pi n,m} \tag{7}$$

Benchmark Data Sets

To compare the efficiency of flow shop scheduling problems, researchers use benchmark data sets. In this paper, the Taillard data set will be used. It consists of 120 benchmark instances. It also provides the best-known upper bounds for the makespan criterion. There are other benchmark data sets provided by Carlier (8 instances) [11], Heller (2 instances) [12], and Revees (21 instances) [13]. These are smaller problems that are straightforward to solve by simple algorithms.

There are two significant indices for measuring the performance of an algorithm. The first one is the solution quality, which can be represented by relative percentage deviation (RPD) over the best-known upper bound. The second important indicator is running time (t_r), which is counted until the cycle taken to reach the last improvement. In this paper we had no investigation into running time.

3. The Family of Bacterial Evolutionary Memetic Algorithms

Bacterial evolutionary algorithm (BEA) is an evolutionary computing algorithm, which was inspired by microbial evolution [14].

BEA is inspired by the interesting process of bacterial recombination. The algorithm uses two operators, bacterial mutation and gene transfer. The first step is to generate an initial population. Then, those two genetic type operations are employed to create new individuals and evaluate them by a fitness function. These operations are repeated until the stop condition.

As the first implementation, the bacterial evolutionary algorithm was only used for finding the optimal parameters of a fuzzy rule-based system. Over the years, it turned out to be efficiently applicable to many other optimization tasks, e.g., interactive nurse scheduling optimization problem [15], automatic data clustering [16], and three-dimensional bin packing problem [17].

According to the early definitions, memetic algorithms are modified genetic algorithms that use an additional local search operator; see Moscato et al. [18]. The idea of combining the BEA with local search came first when an attempt was taken to improve the approximation and optimization capability of the approach when estimating the parameters of fuzzy rule bases. As the latter may be interpreted as black boxes generating input–output functions, the scope of potential benchmark problems was extended to several additional areas, beyond the examples the original Nawa and Furuhashi paper had discussed. Mechanical, chemical, and electrical engineering problems were tested along with transcendental mathematical functions, while the local search applied was a second-order gradient-based method (the Levenberg–Marquardt algorithm). The results turned out to be better than the original ones, even better than any other approach applied in the literature for optimizing the parameters of trapezoidal fuzzy membership functions in fuzzy rule-based "function generators [19]. Later, first-order gradient methods were also tested as local search algorithms, and the results were promising [20].

In the next step, the idea of bacterial memetic evolutionary algorithm was tested on discrete, permutation-based problems, where, as a matter of course, the local search applied was also a discrete process. The first proposed operator family was the n-opt local search, and after some simulations, it was narrowed to the subsequent application of the be 2-opt or 3-opt operators, as when applying $n \geq 4$, the overhead time proved to be too large, thus the efficiency of the algorithm was decreased. The 2-opt operator was first applied to the traveling salesman problem [21], where two edges are exchanged in the graph. 3-opt [22] is similar to the 2-opt operator; the only difference is that, here, three edges are exchanged in one step.

The pseudo-code of the DBMEA is given in Algorithm 1 [23,24]. The algorithm has five input parameters, these are as follows: N_{ind}, N_{clones}, N_{inf}, I_{seg}, *and* I_{trans}; N_{ind} is the number of individuals in the population, N_{clones} is the number of clones in the bacterial mutation, N_{inf} is the number of infections in the gene transfer, I_{seg} is the number of segments in the bacterial mutation, and I_{trans} is the length of the gene transferred part. Step 1 generates an initial population. Step 2 is the application of the bacterial mutation. The third step is the local search (also called as the memetic step). Local search tries to improve on a particular solution (by producing neighbor solutions) until it finds a better neighbor solution. The algorithm stops when it can no longer find a better individual in the neighborhood, i.e., it finds a local optimum. Then, the gene transfer operation is performed. The algorithm repeats steps 2–4 until the termination condition triggers. Then, it returns the best solution.

The following notation is used:

x_1: the actual gene transferred solution,
x_{best}: the best solution found so far,
f: fitness function,
N_{ind}: the number of individuals.

3.1. Discrete Bacterial Memetic Evolutionary Algorithm

Algorithm 1 Discrete Bacterial Memetic Evolutionary algorithm

```
1:  BEGIN PROCEDURE DBMEA ($N_{ind}$, $N_{clones}$, $N_{inf}$, $I_{seg}$, $I_{trans}$)
2:  Step 1: Generate initial population P.
3:  WHILE (termination criteria is not met) DO
4:  Step 2: Bacterial mutation (Population, $N_{clones}$ $I_{seg}$)
5:  Step 3: local search
6:  Step 4: $x_1$ = Gene transfer (Population, $N_{inf}$, $I_{trans}$)
7:  IF $f(x_1) < f(xbest)$ THEN DO
8:  Step 5: $x_{best} = x_1$
9:  END IF
10: END WHILE
11: RETURN $x_{best}$
12: END PROCEDURE
```

3.2. Bacterial Mutation

The bacterial mutation [23] operates throughout the population by performing special mutation operations on each individual (see Algorithm 2). As input parameters, the initial population, the number of clones (N_{clones}), and the length of the segment (I_{seg}) are passed. Steps 1–6 are performed on each element of the population. A certain number of clones (N_{clones}) are made from each bacterium; see Figure 1. The original bacterium is broken down into segments. As the algorithm shows, it happens with high probability for coherent segments, and with low probability for loose segments. During both the coherent and loose segment operations, we go through each segment. We select a non-mutated segment. First, the elements of the segment are inverted to form the first clone. Then, we randomly change the elements of the segment to create other clones. This way, we generate a total of N_{clones} clones. At the end of the mutation, the best clone takes the place of the original bacterium.

Algorithm 2 Bacterial Mutation

```
1:  BEGIN PROCEDURE Bacterial mutation (Population, $N_{clones}$, $I_{seg}$)
2:  FOR i to size(Population) DO
3:  Step 1. Create a random number between 0 and 1
4:  Step 2. Get the ith element of the population: p = Population(i)
5:  Step 3. create $N_{clones}$ clones of p and a random number r [0..1]
6:  IF ($r \leq$ COHERENT_LOOSE_RATE)
7:  Step 4. cut p into coherent segments with $I_{seg}$ length
8:  ELSE
9:  Step 5. cut p into loose segments with $I_{seg}$ length
10: END IF
11: Step 6. replace Population(i) with the best set of the clones and p
12: END FOR
13: RETURN Population
14: END PROCEDURE
```

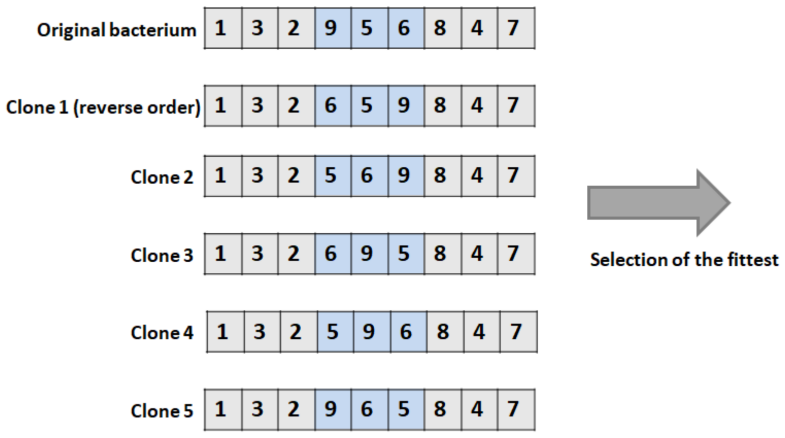
Original bacterium 1 3 2 9 5 6 8 4 7
Clone 1 (reverse order) 1 3 2 6 5 9 8 4 7
Clone 2 1 3 2 5 6 9 8 4 7
Clone 3 1 3 2 6 9 5 8 4 7
Clone 4 1 3 2 5 9 6 8 4 7
Clone 5 1 3 2 9 6 5 8 4 7
Selection of the fittest

Figure 1. Bacterial mutation.

In the case of a coherent segment, the segments are arranged one after the other (Figure 2). In the case of a loose segment, the elements of the segments are not adjacent (Figure 3).

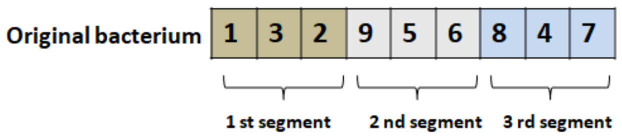

Figure 2. The coherent segment mutation.

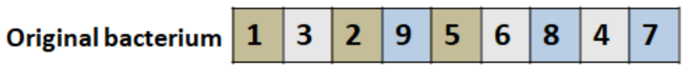

Figure 3. The loose segment mutation.

3.3. *Gene Transfer*

The gene transfer [23] operates on the whole population. At first, the elements of the population are ranked based on the fitness values. Then, the population is divided into two parts, a superior and an inferior part, according to the fitness values. As the next step, the gene transfer operator is executed N_{inf} times with a randomly selected element from the superior and one from the inferior part. During the gene transfer operation, a randomly selected segment with length I_{trans} is transferred from the superior bacterium to the inferior bacterium, so that there are no duplicates in the thus established new bacterium. This process is illustrated in Figure 4. Algorithm 3 presents the process.

Algorithm 3 Gene transfer

```
1:  BEGIN PROCEDURE Gene transfer (Population, $N_{inf}$, $I_{trans}$)
2:  Step 1. sort the Population according to the fitness values
3:  Step 2. divide the population into superior and inferior parts based on the
    fitness values
4:  FOR i to $N_{inf}$ DO
5:  Step 3. selecting a random bacterium from the superior part ($p_{source}$)
6:  Step 4. selecting a random bacterium from the inferior part ($p_{destination}$)
7:  Step 5. selecting a random segment from $p_{source}$ with $I_{trans}$ length
8:  Step 6. copying the segment into $p_{destination}$ bacterium in a random position
9:  Step 7. eliminating the duplicates in $p_{destination}$
10: END FOR
11: RETURN Population
12: END PROCEDURE
```

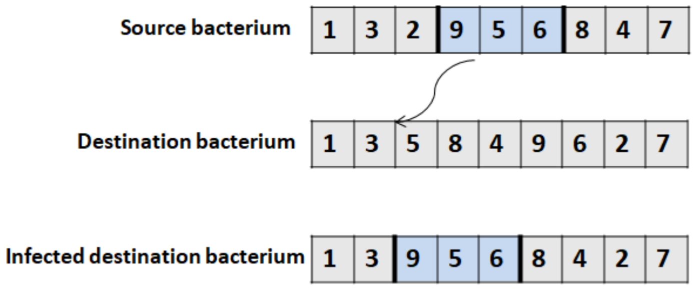

Figure 4. The gene transfer operator.

4. The Simulated Annealing Algorithm

Simulated annealing (SA) [25] operates on a single solution rather than on a whole population. SA first produces a random new neighboring solution. If this neighbor is better than the current solution, it accepts it deterministically as new current solution. If it is not better, the algorithm still may decide probabilistically whether to keep the current one, or replace it with the new neighboring one. The input parameter of the algorithm is the "temperature" (T), which determines the probability of accepting worse solutions in the algorithm. The temperature continuously decreases; it is determined by the temperature control parameter (α). Algorithm 4 illustrates the SA algorithm.

Algorithm 4 Simulated Annealing

```
1:  SIMULATED ANNEALING
2:  BEGIN PROCEDURE Bacterial mutation (T,α,L)
3:  WHILE termination condition is not met DO
4:  WHILE L processing length is not reached DO
5:  Step 1. Create the neighbor (SN) of the current solution (SC)
6:  Step 2. Calculate ΔE as follows: ΔE = E(SN) − E(SC)
7:  IF ΔE > 0 THEN
8:  Step 3. SC = SN
9:  ELSE IF P(E(SN), E(SC), T) > rand[0, 1]) THEN DO
10: Step 4. SC = SN
11: END WHILE
12: Step 5. Reduce temperature (T)
13: END WHILE
14: END PROCEDURE
```

5. A Novel Algorithm: A Hybrid Discrete Bacterial Memetic Evolutionary Algorithm with Simulated Annealing

We could see in the DBMEA pseudo-code that the algorithm uses discrete local search. The originally proposed algorithm applies the 2-opt and 3-opt methods, which operate on individual elements of the population. To improve the efficiency of the local search in the solution of the particular problem on hand, we propose now to use simulated annealing for local search instead. This latter accepts a worse new solution with some probability, and this way allows getting out of small local optimum areas. In addition, we have introduced a mortality rate (N_{mort}). It practically means that certain elements of the population are to be dropped and replaced by randomly generated new individuals. The pseudo-code of our algorithm is shown in Algorithm 5, where N_{ind} is the number of individuals; N_{clones} is the number of clones; N_{inf} means the number of times the gene transfer operator is executed; I_{seg} is the length of the segment; I_{trans} is the length of the segment, which is transferred from the source to the destination bacterium; T is the temperature, which is decreasing along the iterations by α; and L is the iteration control parameter.

Algorithm 5 The Discrete Bacterial Memetic Evolutionary Algorithm with Simulated Annealing

```
1:  BEGIN PROCEDURE DBMEA_SA ($N_{ind}$, $N_{clones}$, $N_{inf}$, $I_{seg}$, $I_{trans}$, T, α, L, $N_{mort}$)
2:  Step 1: Generate initial population P.
3:  WHILE (termination criteria is not met) DO
4:  Step 2: Bacterial mutation (P, $N_{clones}$, $I_{seg}$)
5:  FOR i IN Population DO
6:  Step 3: Simulated annealing (P(i), T, α, L)
7:  END FOR
8:  Step 4: Gene transfer (P, $N_{inf}$, $I_{trans}$)
9:  Step 5: Sort the population, based on the fitness values
10: Step 6: Generate new elements in the population in place of the worst $N_{mort}$ elements
11: Step 7: Store the best solution
12: END WHILE
13: RETURN best solution
14: END PROCEDURE
```

6. Experimental Results

The proposed hybrid algorithm was implemented for a personal computer with 8th generation Intel i7 CPU and 16 GB memory. The Typescript programming language was used, as it allowed the ability to quickly implement the algorithm variants in a portable way. The authors ran the tests on a Windows 10 operating system. Full source codes are available at [26]. To compare the results of the new method with other ones known from the literature, the Taillard benchmark dataset [3] was used. There are 10 individuals for each benchmark data type in the set. Table 1 contains the makespan values calculated by the following algorithms.

Table 1. Experimental results compared with other up-to-date approaches.

Instance	n × m	Lower Bound	Best Known	DBMEA + SA	IWO [7]	HGSA [2]	HGA [4]	HMM-PFA [6]
Ta001	20 × 5	1232	1278	1283	1389	1324	1449	1486
Ta002	20 × 5	1290	1359	1360	-	1442	1460	1528
Ta003	20 × 5	1073	1081	1081	-	1098	1386	1460
Ta004	20 × 5	1268	1293	1293	-	1469	1521	1588
Ta005	20 × 5	1198	1235	1235	-	1291	1403	1449
Ta011	20 × 10	1448	1582	1587	2207	1713	1955	2044
Ta012	20 × 10	1479	1659	1681	-	1718	2123	2166
Ta013	20 × 10	1407	1496	1510	-	1555	1912	1940
Ta014	20 × 10	1308	1377	1384	-	1516	1782	1811
Ta015	20 × 10	1325	1419	1420	-	1573	1933	1933
Ta021	20 × 20	1911	2297	2308	3226	2331	2912	2973
Ta022	20 × 20	1711	2099	2120	-	2280	2780	2852
Ta023	20 × 20	1844	2326	2349	-	2480	2922	3013
Ta024	20 × 20	1810	2223	2223	-	2362	2967	3001
Ta025	20 × 20	1899	2291	2316	-	2507	2953	3003
Ta031	50 × 5	2712	2724	2724	3020	2731	3127	3160
Ta032	50 × 5	2808	2834	2848	-	2934	3438	3432
Ta033	50 × 5	2596	2621	2634	-	2638	3182	3210
Ta034	50 × 5	2740	2751	2776	-	2785	3289	3338
Ta035	50 × 5	2837	2863	2864	-	2864	3315	3356
Ta041	50 × 10	2907	2991	3059	3465	3198	4251	4274
Ta042	50 × 10	2821	2867	2933	-	3020	4139	4177
Ta043	50 × 10	2801	2839	2931	-	3055	4083	4099
Ta044	50 × 10	2968	3063	3077	-	3124	4480	4399
Ta045	50 × 10	2908	2976	3041	-	3129	4316	4322
Ta051	50 × 20	3480	3850	3957	5475	4105	6138	6129
Ta052	50 × 20	3424	3704	3823	-	3992	5721	5725
Ta053	50 × 20	3351	3640	3760	-	3900	5847	5862
Ta054	50 × 20	3336	3720	3823	-	3921	5781	5788
Ta055	50 × 20	3313	3610	3737	-	4020	5891	5886
Ta061	100 × 5	5437	5493	5495	5839	5536	6492	6361
Ta062	100 × 5	5208	5268	5290	-	5302	6353	6212
Ta063	100 × 5	5130	5175	5213	-	5221	6148	6104
Ta064	100 × 5	4963	5014	5023	-	5044	6080	5999
Ta065	100 × 5	5195	5250	5265	-	5358	6254	6179
Ta071	100 × 10	5759	5770	5825	6815	5964	8115	8055
Ta072	100 × 10	5345	5349	5414	-	5596	7986	7853
Ta073	100 × 10	5623	5676	5727	-	5796	8057	8016
Ta074	100 × 10	5732	5781	5892	-	5928	8327	8328
Ta075	100 × 10	5431	5467	5567	-	5748	7991	7936
Ta081	100 × 20	5851	6202	6407	9405	6395	10,745	10,675
Ta082	100 × 20	6099	6183	6334	-	6433	10,655	10,562
Ta083	100 × 20	6099	6271	6480	-	6689	10,672	10,587
Ta084	100 × 20	6072	6269	6409	-	6419	10,630	10,588
Ta085	100 × 20	6009	6314	6518	-	6536	10,548	10,506
Ta091	200 × 10	10,816	10,862	11,002	11,783	11,120	15,739	15,225
Ta092	200 × 10	10,422	10,480	10,627	-	10,658	15,534	14,990
Ta093	200 × 10	10,886	10,922	11,088	-	11,224	15,755	15,257
Ta094	200 × 10	10,794	10,889	11,004	-	11,075	15,842	15,103
Ta095	200 × 10	10,437	10,524	10,666	-	10,793	15,692	15,088
Ta101	200 × 20	10,979	11,195	11,483	15,217	11,642	20,148	19,531

Table 1. *Cont.*

Instance	n × m	Lower Bound	Best Known	DBMEA + SA	IWO [7]	HGSA [2]	HGA [4]	HMM-PFA [6]
Ta102	200 × 20	10,947	11,203	11,535	-	11,683	20,539	19,942
Ta103	200 × 20	11,150	11,281	11,603	-	11,930	20,511	19,759
Ta104	200 × 20	11,127	11,275	11,634	-	11,791	20,461	19,759
Ta105	200 × 20	11,132	11,259	11,549	-	11,728	20,339	19,697
Ta111	500 × 20	25,922	26,059	26,652	30,730	26,859	49,095	46,121
Ta112	500 × 20	26,353	26,520	27,115	-	27,220	49,461	46,627
Ta113	500 × 20	26,320	26,371	n/a	-	27,511	48,777	46,013
Ta114	500 × 20	26,424	26,456	26,974	-	26,912	49,283	46,396
Ta115	500 × 20	26,181	26,334	n/a	-	26,930	48,950	46,251

- DBMEA + SA: Discrete Bacterial Memetic Algorithm + Simulated Annealing;
- IWO: Invasive Weed Optimization [7];
- HGSA Hybrid Genetic Simulated Annealing [2];
- HGA: Hybrid Genetic Algorithm [4];
- HMM-PFA: Hormone Modulation Mechanism Flower Pollination Algorithm [6].

In the test we carried out, twelve different problem sizes were selected; these can be found in the column "$n \times m$: job and machine numbers", namely, 20 × 5, 20 × 10, 20 × 20, 50 × 5, 50 × 10, 50 × 20, 100 × 5, 100 × 10, 100 × 20, 200 × 5, 200 × 10, and 200 × 20. The instance names run from Ta001 to Ta120. Table 1 shows five instances for each problem set, while the full table with comparisons of the results applying the five approaches mentioned above is published also in [26].

For the evaluation of the obtained optima, it is worthwhile to compare both our own results and the ones obtained by other authors. There is an estimation method for an absolute theoretic lower bound, which is proposed in [3] and can be calculated as follows: let b_i the minimum amount of time before machine starts working and a_i is the minimum time until it remains inactive after the end of the operation and let T_i be its total processing time:

$$b_i = \min_j \left(\sum_{k=1}^{i-1} p_{kj} \right) \tag{8}$$

$$a_i = \min_j \left(\sum_{k=i+1}^{m} p_{kj} \right) \tag{9}$$

$$T_i = \sum_{j=1}^{n} p_{ij} \tag{10}$$

Let C_{max} denote the optimal makespan time; it must be greater or equal to the maximum between the minimum of time required by the machines and the minimum of time required each job. This value is called "lower bound", and Table 1 displays this theoretical minimum in the third column:

$$Lower\ bound = max\left\{ \max_i \, (b_i + a_i + T_i), \max_j \left(\sum_{i=1}^{m} p_{ij} \right) \right\} \le C_{max} \tag{11}$$

Our hybrid DBMEA + SA algorithm ran within a reasonably short time, even though no direct measurements were done. In the case of some large instances, however, the running time exceeded the limitation of the available computer resources. Those cases are indicated in Table 1 by n/a entries. Our algorithm always found a better or equal result compared with all other approaches in the literature. It found the best-known solution in 9 cases out of 120. In a further 56 cases, the deviation from the optimal solution was less than 1%; in the remaining 52 cases, the difference was between 1% and 3%. In three cases, the running time exceeded the set limit. Where the algorithm did not find the best-known solution, it got very close to it. Compared with all other algorithms published by other authors, as mentioned above, the proposed new algorithm provided much better results in all cases.

7. Conclusions

There is an interesting symmetry–asymmetry issue when solving complex problems, setting up models for complex systems, and developing algorithms for search and optimization in them. In this paper, the highly complex and mathematically intractable flow shop scheduling problem is in one pan of the scale, while in the other, the new modified discrete bacterial memetic evolutionary algorithm (DBMA) is found. By proper weighing of the costs, namely, the error in the accuracy of the optimization in one pan and the need for resources, especially, the running time of the optimization meta-heuristics in the other one must be brought to equilibrium, this way generating a symmetry in the solution. The exact position of the symmetrical (balanced) solution can, however, be calibrated by the designer of the solution, thus it may fit the application context of the concrete problem, considering the available resources and the expected quality of the quasi-optimum found. Thus, the asymmetric role played by problem to solve and model/algorithm for solution must be balanced and, that way, the whole problem–solution complex must be brought in a symmetrical configuration.

In our approach, there is, however, another aspect of the symmetry–asymmetry concept present. Memetic algorithms consist of two essential components, the "outer" shell that is an evolutionary or population based global searcher, and the "inner" core that is a local searcher, whether traditional gradient based, or exhaustive search type, respectively; or, as in the novel algorithm proposed in this paper, another meta-heuristic method. The two components must also form a symmetric combination in the above sense: the two must be in proper balance of resource intensity and need. Many results have shown that too much local search will slow down the whole optimization procedure, while too little (compared to the outer global search) may lead to ever randomly wandering attempts to approach the optimum, where even the most efficient local search can only produce a local optimum. We trust that, in this novel algorithm, a very efficient and well balanced, let us say, symmetric enough, solution for the combination of the two components of the memetic algorithm was found.

In our paper, the DBMA was very successfully applied for the approximate solution of other, similarly NP-hard discrete problems. Namely, the original DBMA with n-opt type local search method was developed for TSP problems. We found, however, that this local search provided relatively poor results for solving the flow shop scheduling problem. In the proposed new and improved algorithm, we have replaced the local search by the simulated annealing algorithm, a method that has been applied with some success itself for solving similar tasks. We found that this hybrid DBMEA and SA algorithm became unambiguously more efficient, compared with all other population-based metaheuristic approaches proposed by other researchers. The authors calculated the make span times for a known benchmark data set and compared the results with the algorithms in other papers as well as with the best known solutions (it should be clearly stated that the optimal solution cannot be calculated owing to the size of the problem, so the best known published makespan times were used as the basis of the comparison). The proposed new algorithm indeed over-performed all the state-of-the-art algorithms. The calculated makespan times were very close to the best known solutions, while the computing time still remained reasonable, even on a standard personal computer. So, the proposed algorithm has the capability to so far most efficiently solve large-scale FSPP problems.

Author Contributions: Conceptualization, L.T.K. and K.N.; methodology, K.N. and A.A.; software, K.N. and A.A.; validation, O.H.; formal analysis, O.H. and A.A.; investigation, A.A. and K.N.; resources, A.A.; data curation, A.A. and K.N.; writing—original draft preparation, A.A. and O.H.; writing—review and editing, A.A., O.H., L.T.K. and K.N.; visualization, A.A.; supervision, L.T.K. and K.N.; project administration, O.H.; funding acquisition, L.T.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by the National Office of Research, Development, and Innovation grant NKFIH K124055.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are openly available in GitHub at https://doi.org/10.5281/zenodo.4635757 (accessed on 17 April 2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Johnson, S.M. Optimal two-and three-stage production schedules with setup times included. *Nav. Res. Logist. Q.* **1954**, *1*, 61–68. [CrossRef]
2. Wei, H.; Li, S.; Jiang, H.; Hu, J.; Hu, J. Hybrid genetic simulated annealing algorithm for improved flow shop scheduling with makespan criterion. *Appl. Sci.* **2018**, *8*, 2621. [CrossRef]
3. Taillard, E. Benchmarks for basic scheduling problems. *EJOR* **1993**, *64*, 278–285. [CrossRef]
4. Tseng, L.Y.; Lin, Y.T. A hybrid genetic algorithm for no-wait flowshop scheduling problem. *Int. J. Prod. Econ.* **2010**, *128*, 144–152. [CrossRef]
5. Belabid, J.; Aqil, S.; Allali, K. Solving Permutation Flow Shop Scheduling Problem with Sequence-Independent Setup Time. *J. Appl. Math.* **2020**, *2020*, 7132469. [CrossRef]
6. Qu, C.; Fu, Y.; Yi, Z.; Tan, J. Solutions to no-wait flow shop scheduling problem using the flower pollination algorithm based on the hormone modulation mechanism. *Complexity* **2018**, *2018*, 1973604. [CrossRef]
7. Zhou, Y.; Chen, H.; Zhou, G. Invasive weed optimization algorithm for optimization no-idle flow shop scheduling problem. *Neurocomputing* **2014**, *137*, 285–292. [CrossRef]
8. Ogbu, F.A.; Smith, D.K. The application of the simulated annealing algorithm to the solution of the n/m/Cmax flowshop problem. *Comput. Oper. Res.* **1990**, *17*, 243–253. [CrossRef]
9. Lin, S.W.; Cheng, C.Y.; Pourhejazy, P.; Ying, K.C. Multi-temperature simulated annealing for optimizing mixed-blocking permutation flowshop scheduling problems. *Expert Syst. Appl.* **2020**, *165*, 113837. [CrossRef]
10. Aurich, P.; Nahhas, A.; Reggelin, T.; Tolujew, J. Simulation-based optimization for solving a hybrid flow shop scheduling problem. In Proceedings of the 2016 Winter Simulation Conference (WSC), Arlington, VA, USA, 11–14 December 2016; pp. 2809–2819.
11. Carlier, J. Ordonnancements a contraintes disjonctives. *RAIRORecherche Oper.* **1978**, *12*, 333–351. [CrossRef]
12. Heller, J. Some numerical experiments for an M×J flow shop and itsdecision-theoretical aspects. *Oper. Res.* **1960**, *8*, 178–184. [CrossRef]
13. Reeves, C. A genetic algorithm for flowshop sequencing. *Comput. Oper. Res.* **1995**, *22*, 5–13. [CrossRef]
14. Nawa, N.E.; Furuhashi, T. Fuzzy system parameters discovery by bacterial evolutionary algorithm. *IEEE Trans. Fuzzy Syst.* **1999**, *7*, 608–616. [CrossRef]
15. Inoue, T.; Furuhashi, T.; Maeda, H.; Takaba, M. A study on interactive nurse scheduling support system using bacterial evolutionary algorithm engine. *Trans. Inst. Elect. Eng. Jpn.* **2002**, *122*, 1803–1811.
16. Das, S.; Chowdhury, A.; Abraham, A. A bacterial evolutionary algorithm for automatic data clustering. In Proceedings of the IEEE Congress on Evolutionary Computation 2009 (CEC '09), Trondheim, Norway, 18–21 May 2009; pp. 2403–2410.
17. Hoos, H.H.; Stutzle, T. *Stochastic Local Search: Foundations and Applications*; Morgan Kaufmann: San Francisco, CA, USA, 2005.
18. Moscato, P.; Mathieson, L. Memetic Algorithms for Business Analytics and Data Science: A Brief Survey. *Bus. Consum. Anal. New Ideas* **2019**, 545–608. [CrossRef]
19. Gong, G.; Deng, Q.; Chiong, R.; Gong, X.; Huang, H. An effective memetic algorithm for multi-objective job-shop scheduling. *Knowl. Based Syst.* **2019**, *182*, 104840. [CrossRef]
20. Botzheim, J.; Cabrita, C.; Kóczy, L.T.; Ruano, A.E. Fuzzy rule extraction by bacterial memetic algorithms. *Int. J. Intell. Syst.* **2009**, *24*, 312–339. [CrossRef]
21. Muyldermans, L.; Beullens, P.; Cattrysse, D.; Van Oudheusden, D. Exploring variants of 2-opt and 3-opt for the general routing problem. *Oper. Res.* **2005**, *53*, 982–995. [CrossRef]
22. Balazs, K.; Koczy, L.T. Hierarchical-interpolative fuzzy system construction by genetic and bacterial memetic programming approaches. *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* **2012**, *20*, 105–131. [CrossRef]

23. Kóczy, L.T.; Földesi, P.; Tüű-Szabó, B. An effective discrete bacterial memetic evolutionary algorithm for the traveling salesman problem. *Int. J. Intell. Syst.* **2017**, *32*, 862–876. [CrossRef]
24. Tüű-Szabó, B.; Földesi, P.; Kóczy, L.T. An Efficient Evolutionary Metaheuristic for the Traveling Repairman (Minimum Latency) Problem. *Int. J. Comput. Intell. Syst.* **2020**, *13*, 781–793. [CrossRef]
25. Dai, M.; Tang, D.; Giret, A.; Salido, M.A.; Li, W.D. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robot. Comput. Integr. Manuf.* **2013**, *29*, 418–429. [CrossRef]
26. Agárdi, A.; Nehéz, K. Flow Shop Scheduling Problem Optimization with Discrete Bacterial Memetic Evolutionary Algorithm and Simulated Annealing. 2021. Available online: https://github.com/anitaagardi/production-optimization-DBMEA (accessed on 25 March 2021).