

MULTI-SWARM BEES ALGORITHM WITH GPU COMPUTING FOR BELT CONVEYOR PRODUCTION SYSTEM

Wojciech Chmiel*, Paweł Kolendo, Joanna Kwiecień

*AGH University of Krakow, Department of Automatic Control and Robotics,
al. Mickiewicza 30, 30-059 Krakow, Poland*

**E-mail: wch@agh.edu.pl*

Submitted: 28th July 2025; Accepted: 19th May 2026

Abstract

The paper presents the application of the Bees Algorithm to solve the $\mathcal{NP}$ -hard optimisation problem related to the Belt Conveyor Systems. For this purpose, the mathematical model for a real-life example of the Belt Conveyor Production System (BCPS) was defined. Creating the multi-swarm Bees Algorithm for solving this optimisation problem on a CUDA platform within the accepted time for real production systems, and analyzing the impact of control parameters on the algorithm's performance are the main objectives of our study. Moreover, the impact of the different block and grid sizes was checked. To ensure comparability of the results, extensive tests have been carried out using the multi-swarm version designed for the GPU and the mono-swarm version designed for the CPU platform. The results show that the proposed multi-swarm Bees Algorithm algorithm for the CUDA platform gives considerable acceleration of the calculations, but often requires the larger size of the neighbourhood.

Keywords: CUDA, quadratic assignment problem, multi-swarm bees algorithm, Belt Conveyor Production System (BCPS)

1 Introduction

The rapid development of science and technology requires constantly increasing computational resources. It is due to the size and complication of the solved problems which in most cases are $\mathcal{NP}$ or even $\mathcal{NP}$ -hard ones. To find an acceptable solution to these problems, new approximation algorithms are proposed and new high computational power solutions are used.

In this paper a parallel version of the approximation algorithm on CUDA platform for optimisation of the Belt Conveyor Production System

(BCPS), based on a nature inspired bees algorithm, was proposed and tested.

Conveyor systems help in quick, safe and efficient moving of a wide variety of materials. This is the reason why conveyors are very popular equipment in several industries. There exist some various types of the belt conveyors used in different industries such as automotive, electronic, aerospace, chemical, computer, pharmaceutical and many others.

One or more parallel conveyor belts can be used to move the equipment parts between workstations

in industrial production systems, where they are processed. As it should be done efficiently, the optimisation problem can be described as a job scheduling problem. Job scheduling can be defined as the assignment of resources to tasks over time in the context of a predefined optimisation criterion. In industrial implementation, the resources and tasks are usually referred to as machines and jobs. The most common criterion is the completion time of jobs is called makespan. The job scheduling problem has been widely researched since the 1950's.

A flow shop problem is one of the models of the job scheduling on dedicated machines. In this problem all jobs are processed on all machines in the same order and each job has one operation on each machine with the known processing time. Operations belonging to the same task can not be executed in parallel. A special type of the flow shop problem is the permutation flow shop scheduling, in which all jobs have the same execution sequence on all machines.

The paper focuses on the real life type of the job shop scheduling problem which appears in the Belt Conveyor Production System. The considered BCPS system includes a conveyor which moves materials or parts of equipment between workstations, where they are processed or combined (Figure 1). In this case the BCPS system workstations are distributed around the orbital conveyor which acts as the internal, dynamic warehouse of a production system. The discussed example of the BCPS system can be modelled as the $\mathcal{NP}$ -hard discrete optimisation problem.

The Bees Algorithm (BA), which is presented in this paper, is an optimisation method, inspired by the observation of bees' behaviour. The idea behind this paper is to focus on the research into implementation of the Bees Algorithm on the GPU platform for solving $\mathcal{NP}$ -hard problems, which appear in the BCPS systems. This platform allows developers to accelerate applications by using computational power of multicore graphic cards and processors. Moreover, the motivation for this work was to check how the capabilities offered by popular GPUs, with the cost under 100\$, can be exploited to accelerate hard optimisation tasks in industrial applications requiring high computational power and tailored to the real life production systems. Additionally, we wanted to examine if there exists a

difference between the results obtained by multi-swarm and the results obtained by mono-swarm version of the BA algorithm. The last but not least aim of the presented research was the examination of how the control parameters of the algorithm and the GPU platform affect optimisation results.

In this paper the following main contributions are made: firstly, the mathematical model of a real production system based on the BCPS system is presented. Secondly, a general framework of the BA algorithm is discussed and the Multi-Swarm Bees Algorithm (MSBA) - the parallel implementation of the BA algorithm on CUDA platform is defined. Thirdly, the optimisation results for different algorithm control parameters are presented and discussed.

The paper is organized as follows. Section 2 defines the optimisation problem, and Section 3 refers to the related works on approximation algorithm for the permutation problems. Section 4 describes the BA algorithm for the BCPS system. Section 5 shortly describes the CUDA platform. It is followed by Section 6 which presents the MSBA algorithm, the parallel multi-swarm implementation of the BA algorithm with CUDA. Experiments performed and their results are presented in Section 7. Section 8 provides concluding remarks. The tests were performed on the basis of the test instances for the $\mathcal{NP}$ -hard permutation problems from the QAPLIB library [5].

2 Belt Conveyor Production System

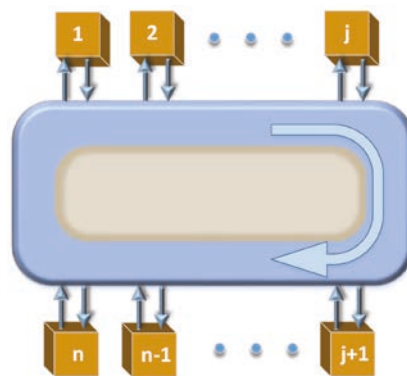


Figure 1. Belt Conveyor Production System with workstations distributed around orbital conveyor.

Figure 1 shows a real example of the BCPS system with the workstations distributed around the orbital conveyor which acts as the internal, dynamic warehouse of the production system. The distances between the workstations (processing positions) spread along the conveyor depends on the transport direction.

For the given set $N = \{1, \dots, n\}$ we define two non-negative matrices $F = [f_{kl}]_{n \times n}$ and $D = [d_{ij}]_{n \times n}$. In the terminology of the BCPS, N is a set of workstations and $\pi(i) \in N, i = 1, \dots, n$ defines the technological task assigned to the workstation i around the orbital conveyor. The matrix D defines distances between the workstations and the matrix F defines the flow of processed elements between pairs of technological tasks assigned to the workstations. The solution to the considered problem can be defined in a permutation form: $\pi = (\pi(1), \dots, \pi(n))$ of the set N which defines the assignment of the technological tasks to the workstations. For example, permutation $\pi = (2, 3, 4, 5, 1)$ defines the assignment of the task number 2 to the workstation 1, task number 3 to the workstation number 2, and so on. Assuming that the conveyor is moving clockwise, the distances between processing positions are equal:

$$d_{ij} = \begin{cases} j - i & \text{if } i \leq j \\ n - i - j & \text{otherwise,} \end{cases} \quad (1)$$

for $i, j = 1, \dots, n$. It is worth noting that $d_{ii} = 0$ and $[d_{ij}]_{n \times n}$ is the asymmetric matrix and a detail transport time between processing position i and j is proportional to d_{ij} , assuming that the transport speed is constant. N technological tasks should be assigned to the workstations in order to minimize the summary of the transportation time of the details on the conveyor:

$$\varphi(\pi^*) = \sum_{i=1}^n \sum_{j=1}^n f_{ij} d_{\pi(i)\pi(j)} \rightarrow \min \quad (2)$$

where $[f_{kl}]_{n \times n}$ is the matrix of transport links between the tasks. For example, it describes the number of the details in the time unit which after technological task k undergoes further technological process – task l . It should be noted that in the presented production system one of the technological tasks is realization of the load/unload task. It should be emphasized that in the case of 26 workstations, there

exist about $4 \cdot 10^{33}$ of the possible workstation allocations around the conveyor.

The mathematical model presented by Equation 2 is similar to the Quadratic Assignment Problem (QAP), the $\mathcal{NP}$ -hard optimisation problem introduced by Koopmans and Beckman [20]. Thanks to that, a solution with a good quality for the considered real example of the BCPS problem can be found using the wide set of the approximation methods applied to the QAP problem. Moreover, a rich set of test instances with different sizes formulated for the QAP problem can be used during the numerical experiments aimed at determining the optimisation quality of the considered problem (e.g. QAPLIB library [5]).

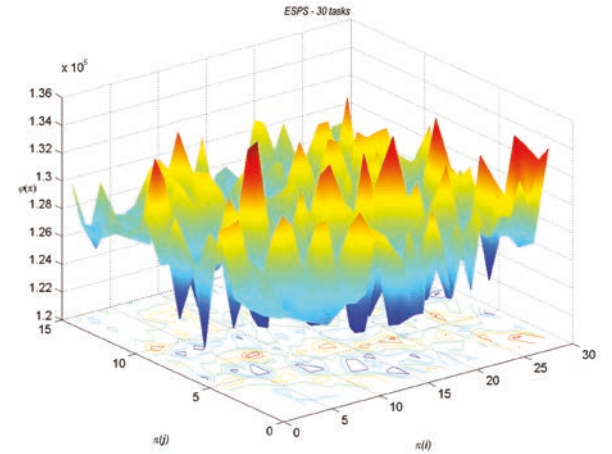


Figure 2. Example of landscape for BCPS system with 30 technological tasks and 2-opt neighborhood structure.

The multimodal landscape presented in Figure 2 shows the low correlation of the neighbourhood solutions in the sense of the Hamming distance, which makes the considered problem difficult to optimise.

3 Related Works

Luo *et al.* [26] proposed a model of energy efficiency for optimisation of a belt conveyor for material scheduling problem. Zang *et al.* [61] presented a constrained multi-objective model for energy efficiency optimisation of a conveying system with a belt conveyor. Wheeler *et al.* [57] discussed models used to reduce the annual equivalent costs of belt conveying systems.

For flow shop problems Nawaz *et al.* [29] proposed a simple heuristic procedure now known as the Nawaz-Enscore-Ham (NEH) heuristic. Salmasi *et al.* [39] proposed a mathematical programming model for the flow shop sequence dependent group scheduling (FSDGS) problem with minimizing the total flow time as criterion function. The defined mixed programming problem (MIP) has been solved using the tabu search (TS) algorithm and hybrid ant colony optimisation (HACO). Mahmoodi *et al.* [28] proposed a set of non-exhaustive heuristics for the job shop cell problem. Lv *et al.* [27] proposed the genetic regulatory network-based rescheduling algorithm for the shop scheduling problem featured with uncertain processing times. Vakharia *et al.* [54] proposed a heuristics based on the simulation annealing algorithm for the flow-line manufacturing cell problem which outperforms heuristic such as the Nawaz-Enscore-Ham. Vijayender *et al.* [55] proposed heuristics for m -machine scheduling jobs problem with sequence-independent setup times and makespan minimization. Skorin *et al.* [42] compared two meta-heuristics for the flow shop problem: the older procedure based on the stimulation annealing and the newly proposed one based on the tabu search algorithm. Sridhar *et al.* [45] proposed a genetic algorithm for the job scheduling in a flowline-based manufacturing cell with the bi-criteria of minimizing makespan and total flowtime. Solimanpur *et al.* [43] proposed a mixed-integer programming problem for the sequence-independent setup times with makespan criterion. The defined problem was solved using metaheuristics based on the tabu search algorithm. Saravanan *et al.* [40] proposed the scatter search method (SS) to minimise the makespan in the Cellular Manufacturing System (CMS). Gupta *et al.* [14] minimize the total flow time for a moderate sized flowline manufacturing cell with independent family setup times using a branch-and-bound algorithm and heuristic algorithms. Karimi *et al.* [18] proposed a new type of the approximation algorithm, imperialist competitive algorithm (ICA) for a hybrid flexible flow shop with sequence-dependent setup times. Until today, many heuristic methods have been created for solving the job shop problems. Ogbu *et al.* in [31] proposed the simulated annealing algorithm, Ben-Daya *et al.* in [3] described a tabu search approach, Rajendran and Ziegler in [37] used ant

colony algorithms, Lian *et al.* in [23] presented the particle swarm optimisation, while Sayadi *et al.* in [41] applied the firefly algorithm. Pan and Ruiz in [32] presented a comparison of many methods, ranging from classic algorithms (Johnson algorithm or dispatching rules) to recent metaheuristics (e.g. tabu search, genetic algorithms, iterated local search) for the permutation flow shop problem with the makespan criterion.

Youcong *et al.* in [30] proposed an EOHIPU algorithm for the Quadratic Assignment Problem. This method mining frequent itemsets with pseudo-utility values to better guide solution generation and balance global and local search. Experimental results on 21 benchmark instances show that EOHIPU outperforms five state-of-the-art methods in both solution quality and convergence speed.

This paper concerns the Bees Algorithm (BA) which imitates the food foraging behaviour of swarms of honey bees [35]. The variations of this algorithm are artificial bee colony (ABC) algorithm [58] and bee colony optimisation (BCO) [52]. Baykasoglu *et al.* [2] proposed the ABC algorithm to solve the generalised assignment problem (GAP). Szeto *et al.* [48] proposed an enhanced version of the ABC algorithm for solving the capacitated vehicle routing problem (CVRP) while Yeh *et al.* [59] presented the penalty guided artificial bee colony algorithm (ABC) to solve the reliability redundancy allocation problem (RAP). Tasgetiren *et al.* [13] proposed the discrete artificial bee colony algorithm (ABC) hybridized with a variant of iterated greedy algorithms to find the permutation that gives the smallest total flowtime.

Wang *et al.* [56] proposed for the large-scale emergency scheduling, particularly in forest fire scenarios involving multiple rescue centers, a hybrid algorithm combining artificial bee colony optimisation with variable neighborhood search is proposed. The experimental results show that the proposed algorithm outperforms the MILP solver CPLEX, genetic algorithms, and particle swarm optimisation algorithms.

Huang *et al.* [15] for the no-wait job shop scheduling problem (NWJSP) proposed a hybrid discrete artificial bee colony (HDABC) algorithm that integrates multiple discrete operations, a novel selection method for onlooker bees, and an adapted left timetabling strategy to minimize total tardiness.

Li *et al.* [22] described a hybrid Pareto-based artificial bee colony (HABC) algorithm for solving the multi-objective flexible job-shop scheduling problem. Thammano *et al.* [53] proposed for this problem a hybrid artificial bee colony algorithm.

Sundar *et al.* [47] applied the artificial bee colony algorithm to the quadratic minimum spanning tree problem (Q-MST).

An extensive study of the artificial bee colony algorithm and its application can be found in [17]. Implementation of the population method in other areas of science and technology can be found in [51, 21].

Suza *et al.* [44] presented the PSO algorithm implementation in CUDA architecture. Luo *et al.* [25] proposed CUBA algorithm (CUDA based on bees algorithm) in order to run it on the GPU.

A generic skeleton implementing the parallelization of the ABC algorithm was proposed by Rubio *et al.* in [12]. The parallelization of the Artificial Bee Colony (ABC) was introduced by Subotic *et al.* in [46], where different threads were assigned to separate swarms and different types of communications between these swarms were described and examined. Three parallel models of the ABC algorithm were compared by Parpinelli *et al.* in [33]: master-slave, multi-hive with migrations and hybrid hierarchical. A PSO version on OpenCL platform for the QAP problem was proposed in [49]. The parallel implementation of the BA algorithm for the QAP problem on CUDA platform was presented by Chmiel *et al.* in [8]. The multi-swarm PSO algorithm for the QAP problem was presented by Szwed *et al.* in [50]. The parallel ABC algorithm for TSP problem was presented by Banharnsakun *et al.* in [1].

In [11], an approach to adapting algorithms originally developed for continuous optimization to discrete problems was presented. The authors proposed a modified variant of the Golden Eagle Optimizer (GEO) for constructing efficient teams in social networks (SNs). The proposed method was applied to a tri-objective formulation of the Team Formation (TF) problem in SNs, with the aim of identifying an optimal team of experts based on communication patterns and skill levels, while simultaneously promoting solution space reduction.

A hybrid algorithm combining the Coyote Optimization Algorithm (COA) and Differential Evolution (DE), referred to as DECOA, was proposed in [10]. The method was evaluated on the CEC-2013 and CEC-2017 test sets, and it was further applied to photovoltaic cell parameter estimation.

Qiao *et al.* [36] present an approach based on the evolutionary algorithms paradigm for the implementation of deep neural networks. An accelerated evolutionary neural architecture search method, termed MFGENAS, was proposed to reduce the computational cost of evolutionary neural architecture search (ENAS) by incorporating multi-fidelity evaluation and a genetic decision controller within the NSGA-II framework.

In [60], a hybrid metaheuristic framework combining portfolio optimization and representation translation was presented to address discrete optimization problems. The cooperative use of unmodified metaheuristics, such as ACO, PSO, and GA, was enabled, and experimental results demonstrated the advantage of the proposed hybrid strategy.

The Bees Algorithm can be implemented in parallel and gives relatively small computational overhead. Moreover, the architecture of the CUDA platform favors the adoption of population algorithms due to the possibility of concurrent execution of thousands of threads (bees) that are relatively independent. These properties are presented in a more detailed manner in Section 6.

4 Bees Algorithm

The Bees Algorithm is the nature inspired algorithm which mimics the food foraging behaviour of swarms of honey bees [35]. The BA algorithm, in its basic version, executed a kind of random search coupled with neighbourhood search.

The bees behaviour, depending on the authors' preferences, can be mapped to model the optimisation problem in many ways [9]. The main elements of the BA algorithm include: the methods of creation of an initial bees' swarm, the algorithms for selecting search directions, the definition of the number of scouts and searched locations and the stop condition. The proposed implementation of the mono-swarm bees algorithm adapted to the considered problem is presented in Algorithm 1.

Algorithm 1 BA algorithm (mono-swarm).Input: $\lambda, I_{max}, l_e, l_b, n_e, n_b, \varphi(\cdot), LT$.**Step 1.** Create a swarm with λ random solutions:

1. Create randomly λ solutions.
2. Evaluate fitness and sort the swarm members from best to worse.
3. Save the best solution in the swarm $\pi_{best} = \arg \min \{\varphi(\pi^i)\}, i = 1, \dots, \lambda$.

Step 2. Select the elite and the best solution from the swarm: $L_e: |L_e| = l_e$ and $L_b: |L_b| = l_b$:

1. Create neighbourhoods using *swap* or *shift* procedure: $\forall \pi \in L_e: N(\pi)$ where $|N(\pi)| = n_e$ and $\forall \pi \in L_b: N(\pi)$ where $|N(\pi)| = n_b$ and calculate the objective function value.
2. For each solution from the set $\{L_e \cup L_b\}$ choose the best solution from the explored neighbourhoods: $\forall \pi \in L_e: \pi^* = \arg \min_{\pi' \in \{N(\pi) \cup \pi\}} \varphi(\pi')$ and $\forall \pi \in L_b: \pi^* = \arg \min_{\pi' \in \{N(\pi) \cup \pi\}} \varphi(\pi')$.

Step 3. Create a new swarm:

1. Replace the solutions from the set $\{L_e \cup L_b\}$ by the new ones obtained in **Step 2.2**.
2. Remove the μ solutions which exist in the swarm longer than LT (*life expectancy*) iterations.
3. Randomly create μ new solutions to fit the swarm size.
4. Sort the swarm from best to worse.

Step 4. Update the best solution: $\pi_{best} = \arg \min \{\varphi(\pi^i)\}, i = 1, \dots, \lambda$.**Step 5.** Check the stop condition:

1. If the I_{max} iteration is reached, then return π_{best} and $\varphi(\pi_{best})$.
2. Otherwise, return to **Step 2**.

In the presented algorithm the solution is defined as a permutation. Therefore, to create a new solution there can be used genetic operators dedicated to the permutation problems: *swap* - swaps the two randomly chosen tasks (the 2-opt neighbourhood) and *shift* - shifts the randomly chosen task to a random workstation (other tasks are moved in the reverse direction). A mechanism based on the *life expectancy* was proposed to prevent stagnancy by getting stuck at local minima. If the solution exists in the swarm more than a predefined number of iterations, then it is replaced by a new one, generated randomly. The algorithm terminates after examining a predefined number of solutions.

Algorithm 1 uses the following parameters: I_{max} – the maximal number of iterations, π_{best} – the globally best found solution, λ – the bees swarm size, $\varphi(\cdot)$ – objective function value, l_e – the number of the elite localizations (number of solutions in the

elite), l_b – the number of good localizations (number of good solutions), n_e – the neighbourhood size for the elite localization, n_b – the neighbourhood size for the good localization, LT – the value of the solution *life expectancy*, μ – the number of solutions existing in a swarm longer than the *life expectancy*. The I_{max} is defined as the maximal number of created solutions during the algorithm execution (or the number of the objective function value calculations).

5 CUDA platform

CUDA (*Compute Unified Device Architecture*) is a software and hardware platform enabling massive parallel execution of programs on the graphics processing units (GPUs). Programs for GPU (called *kernels*) can be executed by thousands of parallel threads which are assigned to thousands of processor cores [19].

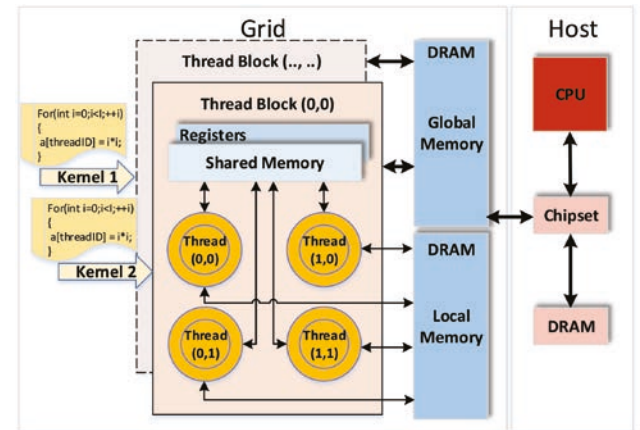


Figure 3. CUDA architecture model.

In GPU platform threads are very lightweight, which assures little creation overhead and very fast switching during execution. Threads divided into batches (blocks) with maximal size equal 1024 can synchronize and communicate using fast shared memory (see Figure 3). This solution makes co-operation more scalable. However, the threads in different blocks cannot communicate and cooperate without CPU mediation. Other types of the memory utilised by threads are: registers (on-chip), local memory (off-chip, uncached) and global memory (off-chip, large, uncached and persistent for all threads). Moreover, CUDA provides some basic

mechanisms supporting parallelism, such as local barriers. CUDA implementation of the BA algorithm presented in this paper is based on CUD-Afy.NET, a .Net library with bindings to CUDA runtime and NVIDIA driver API.

6 Multi-swarm Bees Algorithm on CUDA

The Bees Algorithm, thanks to its construction, has a special ability to execute in parallel. The swarm consists of several identical entities which cooperate with one another to achieve a common goal. In the case of the BA algorithm, the cooperation is based on the exchange information about the objective function value. The algorithm prefers a solution with a better objective function value whose neighbourhood is searched more precisely. For this reason, each entity (a bee) can be evaluated independently in a separate thread. These threads should be synchronised only for the moment of exchanging information about the objective function values. In the CUDA platform the information exchange and synchronisation can take place only between solutions which are executed in the *thread block*. A thread block is a portion of threads with the maximum size from 512 to 1024. Threads from different blocks can not synchronize and communicate. This limitation affects the construction of the proposed algorithm for CUDA platform.

The proposed Multi-Swarm Bees Algorithm (MSBA) is based on the set of swarms which do not cooperate with each other, which allows to achieve a high algorithm performance (see Figure 4). In this sense, the MSBA algorithm resembles the Multi-Start Iterated Local Search algorithm (MS-ILS) [24]. The cooperation takes place only among the bees inside a swarm. The swarm is located inside the block and each bee (solution) from the swarm is associated with one thread. Each bee keeps the associated solution in the shared memory to enable fast (hundred time faster than the global memory) and shared access to it. The quality of the solution (the objective function value) associated with the thread determines to which group it belongs to, i.e. elite solutions - L_e , good solutions - L_b or to the group of the worst solutions, which will be replaced by the new ones, randomly generated.

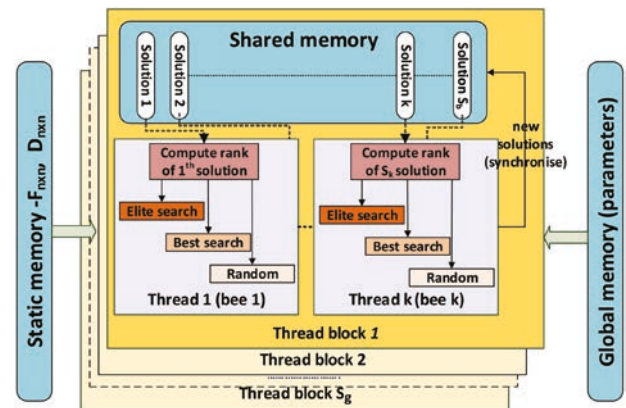


Figure 4. MSBA algorithm execution.

In the MSBA algorithm the solution is represented by a permutation $\pi = (\pi(1), \dots, \pi(n))$ of the set of n elements (workstations), where $\pi(i)$ defines a technological task and i defines the workstation to which this element is assigned. Before the algorithm begins, all necessary data needed by the thread should be prepared and loaded into GPU. These data contain:

- The flow matrix F_{nxn} and the distance matrix D_{nxn} are loaded into the static memory.
- A set of the MSBA algorithm parameters:
 S_g - the GPU grid size (number of the thread blocks), S_b - the number of threads in a block (in other words number of bees in a sub-swarm, therefore the whole swarm has the size $\lambda = S_g \cdot S_b$), l_{eMSBA} - a number of solutions in the elite, l_{bMSBA} - a number of good solutions, n_e - neighbourhood size for the elite localization, n_b - neighbourhood size for the good localization, I_{max} - maximum iteration number, LT - life expectancy.

Each swarm is created using a uniform random distribution with different random seeds, which results in occupying different parts of the solution space by the swarms.

The two matrices F_{nxn} and D_{nxn} are loaded into the static (read-only) memory, which ensures fast access to the data by all the threads. Some restrictions result from the available amount of static memory limited to 64kb, which can become a problem if a big flow and distance matrix are loaded. Algorithm 2 describes the execution of the thread

which is related to one bee (solution). Because solutions are held in the shared memory, all the threads from the block have access to them. This approach enables an easy assignment of the ranking (r) to the solutions. After ranking, the solution neighbourhood is searched more (set L_e) or less (set L_b) exactly or is replaced by a new one, generated randomly.

Algorithm 2 MSBA thread algorithm.

Input: $\pi, \varphi(\cdot), I_{max}, LT$.

Step 1. Determine the ranking r of the solution π . It is performed by counting solutions from the shared memory with the objective function value smaller than $\varphi(\pi)$.

Step 2. Synchronise all solutions in the block (local barrier).

Step 3. If $r \leq |L_e|$, then randomly choose n_e solutions from the neighbourhood of π using *swap* or *shift* procedure. Select the best solution in the neighbourhood $\pi^* = \arg \min_{\pi' \in \{N(\pi) \cup \pi\}} \varphi(\pi')$ and replace π by π^* . Go to **Step 6**.

Step 4. If $|L_e| < r \leq |L_e| + |L_b|$, then randomly choose n_b solutions from the neighbourhood of π using the *swap* or *shift* procedure. Select the best solution in the neighbourhood $\pi^* = \arg \min_{\pi' \in \{N(\pi) \cup \pi\}} \varphi(\pi')$ and replace π by π^* . Go to **Step 6**.

Step 5. If $|L_e| + |L_b| < r$, then replace the solution π by a new one, generated randomly. Go to **Step 7**.

Step 6. If in the steps 3 or 4 a better solution has not been found and π exists in the swarm longer than the LT number of iterations, then replace this solution by a new one, generated randomly.

Step 7. Insert the solution π into the shared memory.

Step 8. Check the stop condition. If it is fulfilled, then STOP, otherwise go to **Step 1**.

7 Experiments and results

7.1 Sub-swarm size

In order to compare the results obtained by the sequential version of the BA algorithm running on CPU (BA) and the multi-swarm parallel version of the algorithm running on the GPU (MSBA), the control parameters were selected in the way which enabled a fair result comparison, i.e., the both algorithms generate the same number of solutions. Thus, for the BA algorithm the parameters values are the following: the size of the elite $l_e = l_{e_{MSBA}} \cdot S_g$, the number of the best solutions $l_b = l_{b_{MSBA}} \cdot S_g$ and the hive size (the number of all bees) $\lambda = S_b \cdot S_g$. The neighbourhood size n_e , n_b and the maximum

number of the algorithm iterations I_{max} are the same for both algorithms.

The computational experiments were aimed to determine the difference between the results achieved by the BA algorithm and the MSBA algorithm which searches the solutions space using small and independent swarms. In particular, the impact of the swarm size in the MSBA algorithm was examined. In case of the experiments for different S_g and S_b values the number of solutions in the elite and the number of the best solutions depend on the hive size (S_b): $l_{e_{MSBA}} = 0.2 \cdot S_b$ and $l_{b_{MSBA}} = 0.15 \cdot S_b$. Therefore, $l_{e_{MSBA}}$ and $l_{b_{MSBA}}$ change from 3 to 123 and from 2 to 92, respectively. The number of the solutions tested in one iteration of the MSBA algorithm in all cases is approximately the same. In the conducted experiments the size of the swarm changes from 630 to 13 while the number of swarms change from 2 to 123.

It should be mentioned that due to memory limitation, the maximal size of one swarm is about 630. This limitation arises from the amount of the static memory which is accessed for a thread block which is limited to 40–64 kB. The hardware used in the tests is described in Table 1.

Table 1. CPU and GPU device parameters.

CPU		GPU	
Intel Core i7-870		Nvidia GeForce GT 730	
Cores	4	Cores	384
Cache	8 MB	Memory	2048 MB
Threads	8	Memory Clock	1.8 Gbps
Instruction Set	64 Bit	Memory Interface Width	64-bit
Base Frequency	2930 MHz	Base Clock	902 MHz
Memory Bandwidth	21 GB/s	Memory Bandwidth	14.4 GB/sec

Moreover, to examine the impact of *life expectancy* (LT) to the convergence of the swarm, the value of LT was changed from 5 to 200.

In the tests 38 instances with the size 25–64 from the QAPLIB library [6] were used. For the comparison of the results obtained by the evaluated algorithms the best known objective function values were used. They were obtained using different optimisation algorithms. For example, the best solutions to the Tho40 and Wil50 were found using the simulation annealing algorithm by Bölte [4] and to the Ste36a using the tabu search algorithm by Skorin-Kapov [16].

Table 2 presents the results obtained by the MSBA and BA algorithms. In this table, the second column (φ_{ref}) contains the best known value of the objective function taken from the QAPLIB library. The following seven columns contain the computation results of the two algorithms - the MSBA and the BA. The φ_{best} is the best objective function value found by the algorithm. T_A - defines the total time of execution in seconds of the I_{max} algorithm iterations. The column E contains $E = \frac{\varphi_{best} - \varphi_{ref}}{\varphi_{ref}} \cdot 100\%$ - the percentage relative gap between the best known value of the objective function value from the QAPLIB library and the best objective function value of the solution found by the MSBA or BA algorithm. Column η describes the computation time ratio - $\eta = \frac{T_A(BA)}{T_A(MSBA)}$. The last row in the tables contains the average value ($E_{avr}, \bar{E}_{MSBA/BA}, Avr$) of all the values in the column above.

The results presented in Table 2 were obtained by the MSBA algorithm run with the following control parameters: $S_b/S_g = 400/4$, $LT = 200$, $l_{e_{MSBA}} = 123$, $l_{b_{MSBA}} = 92$, $I_{max} = 200$ - a number of algorithm iterations (executed by every thread). To obtain similar execution conditions, the BA algorithm has the following parameters: the size of the elite equals $l_e = l_{e_{MSBA}} \cdot S_g = 123 \cdot 4 = 492$, the set size of the good solution $l_b = l_{b_{MSBA}} \cdot S_g = 92 \cdot 4 = 368$, $\lambda = S_b \cdot S_g = 400 \cdot 4 = 1600$ and $I_{max} = 200$. The neighbourhood size, $n_e = 25$ and $n_b = 18$, are the same for both algorithms. The results presented in this table show a 9 times reduction in the execution time of the MSBA algorithm in comparison with the BA algorithm, whereas the obtained results are similar.

Tables 3 and 4 summarize the statistical results produced by the MSBA and BA algorithms under two distinct control parameter configurations. Specifically, the S_b/S_g and n_e/n_b parameter sizes were set to 400/4 and 25/18, and to 123/13 and 177/80, respectively. In this table, the $\varphi_{best}/\varphi_{worst}$ column denotes the best/worst objective function values achieved by the algorithm. The columns φ_{avr} , σ , and I_{avr} correspond to the mean value, the standard deviation of the best results, and the iteration in which the best objective value was found, respectively, averaged over 34 independent runs. A one-sided Wilcoxon signed-rank test applied to the φ values reported in Table 3 (38 paired instances)

showed no statistically significant difference between the two series ($W = 343$ and $p = 0.532$). However, this test applied to the φ values reported in Table 4 confirmed a statistically significant advantage of the MSBA algorithm over BA ($W = 130$ and $p = 7.2 \cdot 10^{-4}$). The results demonstrate a statistically significant difference in the behavior of the MSBA algorithm for different numbers of independent swarms.

Table 5 presents the average running time of the algorithm depends on the instance size for the MSBA and BA algorithms.

Table 5. Average iteration I_{avr} at which the best solution was found versus instance size for the MSBA and BA algorithms ($LT = 200$, $S_b/S_g = 400/4$, $n_e = 25$, $n_b = 18$).

Instance size	I_{avr}	
	MSBA(GPU)	BA(CPU)
22	57.94	26.94
26	56.34	27.10
30	28.39	28.39
32	32.13	25.27
36	39.10	54.80
40	76.61	88.82
42	110.53	93.71
49	67.53	86.35
50	102.80	82.78
56	87.59	64.18
60	101.62	53.59
64	85.71	73.76

Table 6 and Table 7 present the impact of the different block and grid sizes (S_b/S_g) for the six selected *life expectancy* values (LT). The E_{avr} values in the table are the average percentage relative gap E for 38 test instances from the QAPLIB and the constant (S_b/S_g) and LT parameter values (such as the E_{avr} row in Table 2).

Table 8 presents the impact of the LT value on the results obtained by the MSBA and BA algorithms. Figure 5 shows the influence of life expectancy (LT) on the results of the MSBA algorithm.

Each value in the column $\bar{E}_{MSBA}$ and $\bar{E}_{BA}$ is the average value of the average percentage relative gap E_{avr} obtained by the algorithms for 38 test instances and for the 17 different S_b/S_g parameter

Table 2. Results for the MSBA and the BA algorithm for 38 test instances ($LT = 200$, $S_b/S_g = 400/4$, $n_e = 25$, $n_b = 18$).

Name	Φ_{ref}	MSBA(GPU)			BA(CPU)			η
		Φ_{best}	T_A	E	Φ_{best}	T_A	E	
Bur26a	5426670	5427110	10.4	0.01	5426670	122.3	0.00	11.8
Bur26b	3817852	3817852	10.7	0.00	3817852	122.6	0.00	11.4
Bur26c	5426795	5426795	10.0	0.00	5426795	122.3	0.00	12.2
Bur26d	3821225	3821225	10.4	0.00	3821232	122.6	0.00	11.8
Bur26e	5386879	5386879	10.2	0.00	5386879	122.8	0.00	12.0
Bur26f	3782044	3782044	10.4	0.00	3782044	122.4	0.00	11.8
Bur26g	10117172	10117172	9.7	0.00	10117172	122.5	0.00	12.6
Bur26h	7098658	7098658	9.7	0.00	7098658	123.1	0.00	12.7
Chr22a	6156	6236	7.4	1.30	6266	87.3	1.79	11.8
Chr22b	6194	6286	7.5	1.49	6320	87.3	2.03	11.7
Esc32a	130	134	19.4	3.08	134	185.0	3.08	9.5
Esc32b	160	168	19.1	5.00	168	186.9	5.00	9.8
Esc32c	642	642	20.1	0.00	642	185.9	0.00	9.2
Esc32d	200	200	18.9	0.00	200	185.0	0.00	9.8
Esc32e	2	2	20.7	0.00	2	185.0	0.00	8.9
Esc32f	2	2	20.7	0.00	2	185.0	0.00	8.9
Esc32g	6	6	20.4	0.00	6	185.1	0.00	9.1
Esc32h	438	438	19.3	0.00	438	185.2	0.00	9.6
Esc64a	116	116	116.8	0.00	116	775.5	0.00	6.6
Kra30a	88900	89100	15.5	0.22	90100	163.4	1.35	10.5
Kra30b	91420	91990	15.5	0.62	91850	163.4	0.47	10.5
Lipa30a	13178	13196	16.2	0.14	13222	163.7	0.33	10.1
Lipa30b	151426	151426	15.8	0.00	151426	163.3	0.00	10.3
Lipa40a	31538	31940	34.7	1.27	31945	296.4	1.29	8.5
Lipa40b	476581	498634	34.8	4.63	487526	296.2	2.30	8.5
Lipa50a	62093	62862	61.7	1.24	62872	467.8	1.25	7.6
Lipa50b	1210244	1347609	62.5	11.35	1342719	467.7	10.95	7.5
Lipa60a	107218	108422	95.6	1.12	108419	680.2	1.12	7.1
Lipa60b	2520135	3031375	96.9	20.29	3036389	680.2	20.49	7.0
Sko42	15812	16080	38.4	1.69	16044	327.4	1.47	8.5
Sko49	23386	23892	56.6	2.16	23800	449.4	1.77	7.9
Sko56	34458	35292	78.6	2.42	35000	590.2	1.57	7.5
Ste36a	9526	9796	24.4	2.83	9854	239.4	3.44	9.8
Ste36b	15852	16438	23.7	3.70	16378	239.7	3.32	10.1
Ste36c	8239110	8393064	24.2	1.87	8408548	239.4	2.06	9.9
Tho30	149936	150910	15.4	0.65	151172	163.5	0.82	10.6
Tho40	240516	245946	33.3	2.26	245230	296.2	1.96	8.9
Wil50	49320	49244	59.7	0.88	49342	467.9	1.08	7.8
E_{avr}			30.9	1.80		262.3	1.77	8.5

Table 3. Statistical summary of 34 runs conducted under identical control parameters of the MSBA and the BA algorithm for 38 instances ($LT = 200$, $S_b/S_g = 400/4$, $n_e = 25$, $n_b = 18$).

Name	MSBA(GPU)				BA(CPU)			
	Φ_{best}	Φ_{worse}	Φ_{avr}	σ	Φ_{best}	Φ_{worse}	$varphi_{avr}$	σ
Bur26a	5427110	5427110	5426723	126	5426670	5427110	5426696	107
Bur26b	3817852	3817965	3817865	35	3817852	3817936	3817857	20
Bur26c	5426795	5426795	5426795	0	5426795	5426803	5426796	3
Bur26d	3821225	3821263	3821230	12	3821232	3821232	3821227	3
Bur26e	5386879	5387103	5386916	67	5386879	5387092	5386930	79
Bur26f	3782044	3782044	3782044	0	3782044	3782044	3782044	0
Bur26g	10117172	10117473	10117194	73	10117172	10117276	10117178	25
Bur26h	7098658	7098658	7098658	0	7098658	7098658	7098658	0
Chr22a	6236	6288	6207	49	6266	6318	6242	55
Chr22b	6286	6392	6334	29	6320	6402	6324	37
Esc32a	134	136	134	2	134	136	133	1
Esc32b	168	168	168	0	168	168	168	0
Esc32c	642	642	642	0	642	642	642	0
Esc32d	200	200	200	0	200	200	200	0
Esc32e	2	2	2	0	2	2	2	0
Esc32f	2	2	2	0	2	2	2	0
Esc32g	6	6	6	0	6	6	6	0
Esc32h	438	438	438	0	438	438	438	0
Esc64a	116	116	116	0	116	116	116	0
Kra30a	89100	90560	89857	542	90100	90660	89892	516
Kra30b	91990	92290	91916	234	91850	92350	91912	226
Lipa30a	13196	13372	13257	65	13222	13331	13235	39
Lipa30b	151426	151426	151426	0	151426	151426	151426	0
Lipa40a	31940	31969	31938	20	31945	31965	31943	10
Lipa40b	498634	506029	497163	4297	487526	504628	497517	3715
Lipa50a	62862	62895	62872	14	62872	62895	62865	16
Lipa50b	1347609	1414617	1344248	37463	1342719	1390636	1327501	26099
Lipa60a	108422	108456	108419	25	108419	108450	108424	19
Lipa60b	3031375	3040364	3026688	15350	3036389	3041563	2994236	72837
Sko42	16080	16104	16069	28	16044	16116	16071	23
Sko49	23892	23924	23837	51	23800	23908	23851	32
Sko56	35292	35432	35298	63	35000	35396	35269	100
Ste36a	9796	9888	9804	52	9854	9870	9800	51
Ste36b	16438	16720	16439	143	16378	16736	16514	124
Ste36c	8393064	8403788	8381194	20703	8408548	8449634	8421365	22820
Tho30	150910	151804	151022	463	151172	151646	151121	322
Tho40	245946	247526	246444	762	245230	247140	246281	527
Wil50	49244	49350	49279	42	49342	49350	49284	39

Table 4. Statistical summary of 34 runs conducted under identical control parameters of the MSBA and the BA algorithm for 38 instances ($LT = 200$, $S_b/S_g = 123/13$, $n_e = 177$, $n_b = 80$).

Name	MSBA(GPU)				BA(CPU)			
	Φ_{best}	Φ_{worse}	Φ_{avr}	σ	Φ_{best}	Φ_{worse}	Φ_{avr}	σ
Bur26a	5427110	5427110	5426735	268	5426670	5427110	5427337	1203
Bur26b	3817852	3817965	3818009	326	3817852	3817936	3817880	56
Bur26c	5426795	5426795	5426795	0	5426795	5426803	5426837	94
Bur26d	3821225	3821263	3821231	10	3821232	3821232	3821242	29
Bur26e	5386879	5387103	5386893	43	5386879	5387092	5386983	156
Bur26f	3782044	3782044	3782045	6	3782044	3782044	3782045	6
Bur26g	10117172	10117473	10117187	18	10117172	10117276	10117180	16
Bur26h	7098658	7098658	7098658	0	7098658	7098658	7098687	82
Chr22a	6236	6288	6299	56	6266	6318	6349	73
Chr22b	6286	6392	6375	57	6320	6402	6403	40
Esc32a	134	136	132	2	134	136	133	2
Esc32b	168	168	168	0	168	168	168	0
Esc32c	642	642	642	0	642	642	642	0
Esc32d	200	200	200	0	200	200	200	0
Esc32e	2	2	2	0	2	2	2	0
Esc32f	2	2	2	0	2	2	2	0
Esc32g	6	6	6	0	6	6	6	0
Esc32h	438	438	438	0	438	438	438	0
Esc64a	116	116	116	0	116	116	116	0
Kra30a	89100	90560	89947	736	90100	90660	89941	782
Kra30b	91990	92290	91665	184	91850	92350	91742	266
Lipa30a	13196	13372	13278	80	13222	13331	13302	66
Lipa30b	151426	151426	151426	0	151426	151426	152740	5418
Lipa40a	31940	31969	31888	32	31945	31965	31898	31
Lipa40b	498634	506029	478838	6371	487526	504628	481234	19186
Lipa50a	62862	62895	62724	60	62872	62895	62753	22
Lipa50b	1347609	1414617	1228831	52519	1342719	1390636	1293777	90506
Lipa60a	108422	108456	108194	99	108419	108450	108227	20
Lipa60b	3031375	3040364	2659083	211325	3036389	3041563	2985646	47720
Sko42	16080	16104	15916	60	16044	16116	15942	40
Sko49	23892	23924	23601	103	23800	23908	23591	52
Sko56	35292	35432	34796	193	35000	35396	34826	77
Ste36a	9796	9888	9694	77	9854	9870	9717	77
Ste36b	16438	16720	16073	199	16378	16736	16177	173
Ste36c	8393064	8403788	8328280	32203	8408548	8449634	8325415	54469
Tho30	150910	151804	150709	361	151172	151646	150847	407
Tho40	245946	247526	243406	1650	245230	247140	243362	1119
Wil50	49244	49350	48978	120	49342	49350	48994	62

Table 6. Average value of E for 38 test instances for different LT and S_b/S_g values ($n_e = 25$, $n_b = 18$).

LT	3			42			82		
S_b/S_g	E_{avr}		η_{avr}	E_{avr}		η_{avr}	E_{avr}		η_{avr}
	MSBA	BA		MSBA	BA		MSBA	BA	
13/123	13.79	1.57	1.1	3.62	1.57	1.0	2.59	1.57	1.0
16/100	13.28	1.75	1.2	3.50	1.76	1.1	2.46	1.75	1.1
20/80	13.51	1.81	1.3	3.70	1.81	1.2	2.62	1.81	1.2
27/60	13.60	1.85	1.6	3.64	1.85	1.4	2.55	1.85	1.4
40/40	13.82	1.89	2.0	3.47	1.89	1.8	2.83	1.89	1.7
55/30	13.93	1.82	2.8	3.48	1.82	2.3	2.65	1.82	2.1
80/20	13.90	1.80	3.6	3.55	1.80	3.0	2.69	1.80	2.7
160/10	13.60	1.85	6.6	3.37	1.85	5.5	2.76	1.85	5.0
178/9	14.04	1.76	6.6	3.22	1.76	5.5	2.58	1.76	5.0
200/8	13.47	1.67	8.1	3.39	1.67	6.6	2.49	1.67	6.1
240/7	13.87	1.76	8.9	3.70	1.76	7.3	2.67	1.76	6.6
260/6	13.84	1.90	9.9	3.37	1.90	8.1	2.64	1.90	7.5
280/6	13.85	1.81	11.3	3.52	1.81	9.3	2.52	1.81	8.5
320/5	13.98	1.85	9.8	3.63	1.85	8.1	2.89	1.85	7.5
400/4	13.88	1.77	14.0	3.60	1.77	11.4	2.96	1.77	10.7
533/3	14.19	1.57	13.0	3.35	1.57	10.9	2.50	1.57	10.1
630/2	13.80	1.73	15.4	3.79	1.73	12.6	2.59	1.74	12.2
$\bar{E}_{MSBA/BA}$	13.78	1.77	6.9	3.52	1.77	5.7	2.65	1.77	5.3

Table 7. Average value of gap E for 38 test instances for different LT and S_b/S_g values ($n_e = 25$, $n_b = 18$).

LT	122			162			200		
S_b/S_g	E_{avr}		η_{avr}	E_{avr}		η_{avr}	E_{avr}		η_{avr}
	MSBA	BA		MSBA	BA		MSBA	BA	
13/123	2.20	1.57	1.0	1.94	1.57	1.0	1.78	1.57	1.0
16/100	2.03	1.75	1.1	1.81	1.75	1.1	1.73	1.75	1.1
20/80	2.07	1.81	1.2	1.91	1.81	1.2	1.89	1.81	1.2
27/60	2.13	1.85	1.4	1.91	1.85	1.4	1.79	1.85	1.4
40/40	2.35	1.89	1.7	2.02	1.90	1.6	2.00	1.89	1.6
55/30	2.22	1.82	1.9	1.77	1.82	1.8	1.59	1.82	1.8
80/20	2.19	1.80	2.5	2.06	1.80	2.3	1.82	1.80	2.2
160/10	2.24	1.85	4.7	2.02	1.85	4.4	1.78	1.85	4.3
178/9	2.19	1.76	4.7	1.96	1.76	4.4	1.82	1.76	4.3
200/8	2.25	1.67	5.7	1.83	1.67	5.4	1.74	1.67	5.3
240/7	2.15	1.76	6.2	1.97	1.76	5.9	1.86	1.76	5.8
260/6	2.31	1.90	7.1	2.01	1.90	6.8	1.69	1.90	6.6
280/6	2.21	1.81	8.0	1.97	1.81	7.6	1.83	1.81	7.4
320/5	2.40	1.85	7.1	2.14	1.85	6.7	1.98	1.85	6.6
400/4	2.35	1.77	10.1	1.88	1.77	9.7	1.80	1.77	9.5
533/3	2.23	1.57	9.6	2.03	1.57	9.2	1.90	1.57	9.0
630/2	2.13	1.73	11.9	1.92	1.73	11.6	1.85	1.73	11.5
$\bar{E}_{MSBA/BA}$	2.22	1.77	5.0	1.95	1.77	4.8	1.81	1.77	4.7

Table 8. Influence of *life expectancy* (LT) on the results obtained by the MSBA and BA algorithms ($n_e = 25, n_b = 18$).

LT	$\bar{E}_{MSBA}$	$\bar{E}_{BA}$	$\bar{\eta}_{avr}$	LT	$\bar{E}_{MSBA}$	$\bar{E}_{BA}$	$\bar{\eta}_{avr}$
3	13.78	1.77	6.9	100	2.39	1.77	5.2
5	10.09	1.77	6.5	105	2.34	1.77	5.2
10	7.62	1.77	6.3	110	2.28	1.77	5.1
15	5.78	1.77	6.1	115	2.24	1.77	5.1
20	5.13	1.77	6.0	120	2.22	1.77	5.2
25	4.57	1.77	6.0	125	2.18	1.77	5.0
30	4.12	1.78	5.8	130	2.13	1.77	5.0
35	3.78	1.77	5.8	135	2.10	1.77	5.0
40	3.52	1.77	5.7	140	2.08	1.77	4.9
45	3.47	1.77	5.7	145	2.04	1.77	4.9
50	3.20	1.77	5.6	150	2.01	1.77	4.9
55	3.08	1.78	5.5	155	1.98	1.77	4.9
60	2.89	0.77	5.5	160	1.95	1.77	4.8
65	2.88	1.77	5.4	165	1.92	1.77	4.8
70	2.78	1.77	5.4	170	1.90	1.77	4.8
75	2.67	1.77	5.4	175	1.87	1.77	4.8
80	2.65	1.77	5.3	180	1.87	1.77	4.8
85	2.60	1.77	5.3	185	1.85	1.77	4.8
90	2.52	1.77	5.2	190	1.83	1.77	4.8
95	2.46	1.77	5.2	195	1.82	1.77	4.8
				200	1.81	1.77	4.7

values (such as the $\bar{E}_{MSBA/BA}$ row in Tables 6 and 7). The parameters of the BA algorithm were calculated using the formulas defined above based on the parameters of the MSBA algorithm.

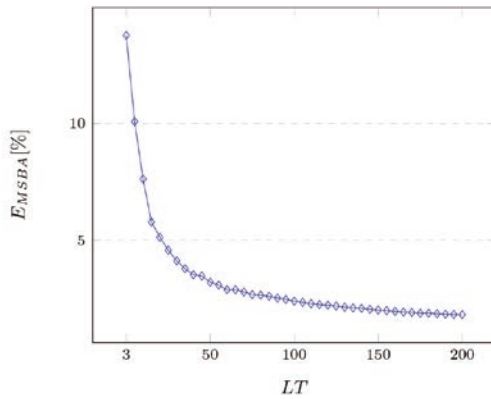


Figure 5. Influence of *life expectancy* (LT) on the results of the MSBA algorithm (Tab. 8).

Figure 6 shows the change of the objective function value of the 1600 solutions processed in each thread (or a bee) in the MSBA algorithm. It can be observed that the courses of the solution values are divided into two groups. The first one is characterised by the notably bigger (worse) objective function values (the upper bunch of the courses).

In contrast, the second one (the lower bunch) of the courses is characterised by the constant improvement in the consecutive iterations of the objective function value. This group contains the solutions which emanate from the better ones (or in other words from the promising areas of the solutions space) randomly generated at the beginning of the algorithm. These solutions never abandon the elite or the best solution set and thus the number of the courses in the lower bunch is equal $(l_e + l_b)S_g$. The high value of *life expectancy*, which equals $LT = 200$, is the reason why the first group of the solutions, which emanates from the worse part of the initial solutions, will never have a chance to be improved and to get to the elite or the best solution set.

On the other hand, the small value of *life expectancy* (Tables 6 and 8: $LT = 3$) leads to the too strong exploration and does not allow the exhaustive search of the promising regions of the solutions space. This is the reason why the $\bar{E}_{MSBA}$ is about 14% and is eight times bigger than the corresponding average gap value for the BA algorithm ($\bar{E}_{BA}$). Besides, the analysis of the results from Tables 6 to 8 shows that the average results obtained for the identical LT values by the BA algorithm ba-

sically do not change. Increasing *life expectancy* enables a better exploitation of the promising regions of the solutions space by the MSBA while it does not affect the results obtained by the BA algorithm (Tables 7 and 8: $LT = 200$). It should be noted that the value of the S_b/S_g does not affect the obtained results by both algorithms.

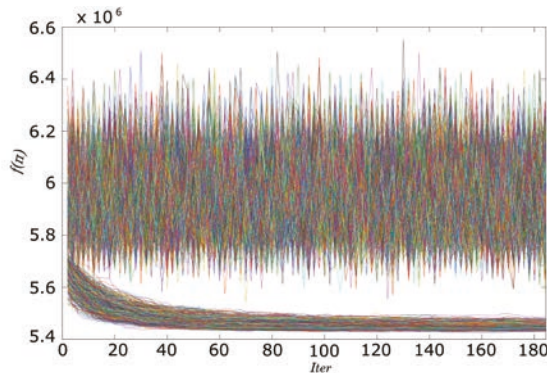


Figure 6. Courses of the MSBA algorithm for the BCPS problem with 50 workstations, for all 1600 bees in swarm ($\lambda = 1600, LT = 200$, $S_b/S_g = 55/30$, $n_e = 25$, $n_b = 18$).

The results presented in Tables 2 to 8 are worse in the case of the MSBA algorithm, because they were obtained using small neighbourhood size: $n_e = 25$, $n_b = 18$. For these parameters the BA algorithm searches more intensively the promising regions than the MSBA algorithm. In other words, the BA algorithm searches only one region exactly while the MSBA algorithm searches several regions less intensively. This phenomenon can be observed in Figure 6, where some solutions can not be improved, among others, due to the small value of the parameters n_e and n_b . It is the reason why for the small values of the n_e and n_b , an increasing number of the independent swarms (S_g) does not affect the obtained results by the MSBA algorithm. The same reason is behind the weak effect of changing the LT value in the case of the BA algorithm contrary to the MSBA algorithm.

The results presented in Table 9 show that parameters n_e and n_b have a big impact on the optimisation results and, when properly chosen, allow for significantly better results by the MSBA algorithm than by the sequential BA algorithm. The presented results were obtained using the MSBA and BA algorithms for the $n_e = 177$, $n_b = 80$ while the other

parameters are the same as those used in the previous experiments.

Table 9. Average value of E_{avr} , $T_{A_{avr}}$ and η for 38 test instances and different S_b/S_g values ($n_e = 177$, $n_b = 80$).

S_b/S_g	MSBA		BA		η_{avr}
	E_{avr}	$T_{A_{avr}}$	E_{avr}	$T_{A_{avr}}$	
13/123	0.62	205.0	1.12	120.6	0.6
16/100	0.75	186.1	1.13	124.9	0.7
20/80	0.68	166.7	1.06	125.7	0.8
27/60	0.70	148.1	1.08	137.6	0.9
40/40	0.62	135.6	1.11	131.0	1.0
55/30	0.87	133.0	1.08	134.6	1.0
80/20	0.61	106.9	1.18	118.0	1.1
160/10	1.15	57.6	1.14	88.7	1.5
178/9	0.59	57.3	1.51	85.6	1.5
200/8	0.64	46.7	1.71	82.4	1.8
240/7	0.76	46.1	1.46	82.1	1.8
260/6	0.61	36.4	1.38	74.9	2.1
280/6	0.65	38.0	1.53	79.1	2.1
320/5	1.02	41.0	1.76	73.4	1.8
400/4	0.67	28.7	1.80	70.6	2.5
533/3	1.90	4.7	2.00	67.7	14.6
630/2	1.85	2.4	2.60	52.4	21.2
Avr	0.86	84.7	1.45	97.0	3.6

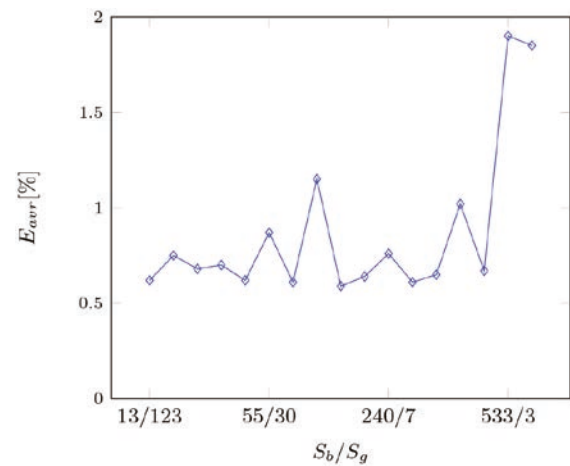


Figure 7. Average percentage gap E_{avr} for different S_b/S_g values.

The analysis of the results leads to the conclusion that the smaller average percentage relative gap value can be obtained for the large number of the swarms (see Figure 7). For $S_g = 123$ the relative average gap obtained by the MSBA algorithm and the BA algorithm equal 0.62% and 1.12%, respectively. For $S_b = 630$ and $S_g = 2$ the algorithm ac-

celeration is the largest: $\eta = 21.2$. Unfortunately, in this case the E_{avr} for the MSBA algorithm equals 1.85%, which is smaller than the E_{avr} of the BA algorithm, but it is three times bigger than the best results obtained by the MSBA algorithm. It can be observed (Figure 8) that the increase in the size of the block results in the bigger acceleration of the calculations.

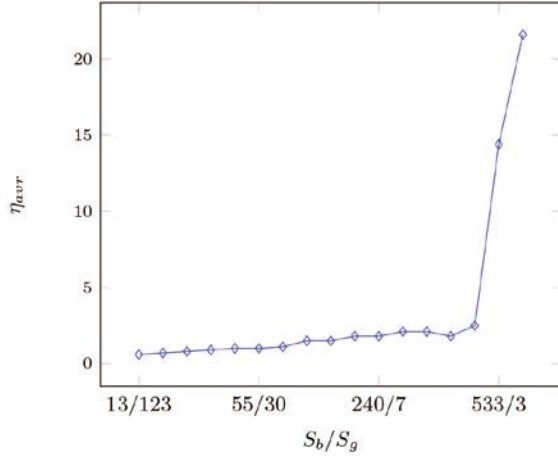


Figure 8. Computation time ratio of the MSBA algorithm for different S_b/S_g values.

7.2 Swarm entropy

Shannon's information entropy (SIE) can be used as a diversity measure of the swarm [7, 38]. For the population S , where k phenotype classes can be distinguished, SIE is defined as follows:

$$SIE(S) = - \sum_k p_k \log p_k \quad (3)$$

where p_k is the proportion of S occupied by population partition k at given time [38]. SIE characterizes the amount of chaos in a system. Large values of entropy correspond to small values of p_k (each partition has a non-significant number of individuals), while small values of entropy correspond to larger values of p_k (significant number of individuals is concentrated in a few partitions). For the population S with size λ at the iteration t , the SIE value can be computed as follows [34]:

$$SIE_t(S) = - \sum_{i=1}^{\lambda} P_i(t) \log P_i(t) \quad (4)$$

where $P_i(t) = \frac{f(\pi_i(t))}{\sum_{j=0}^{\lambda} f(\pi_j(t))}$. $P_i(t)$ can be interpreted as the proportion of the objective function value of

i th particle's best position, π_i , at iteration t . SIE providing information regarding the spread of the individuals' objective function values. High values of SIE correspond to a wide spread of objective function values. Smaller values of SIE indicate similar objective function values.

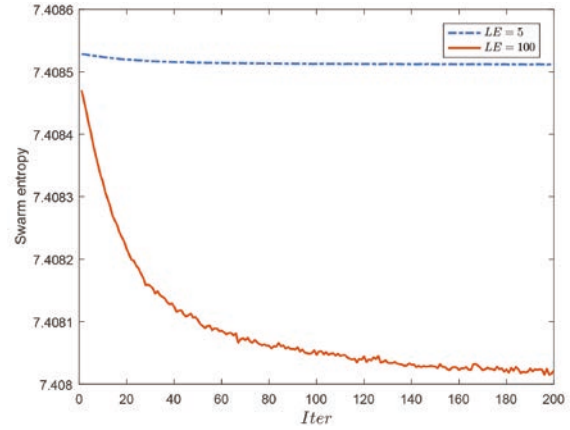


Figure 9. MSBA swarm entropy for BCPS problem with 50 workstations for $LT = 5$ and $LT = 100$ ($\lambda = 1600$, $n_e = 177$, $n_b = 80$).

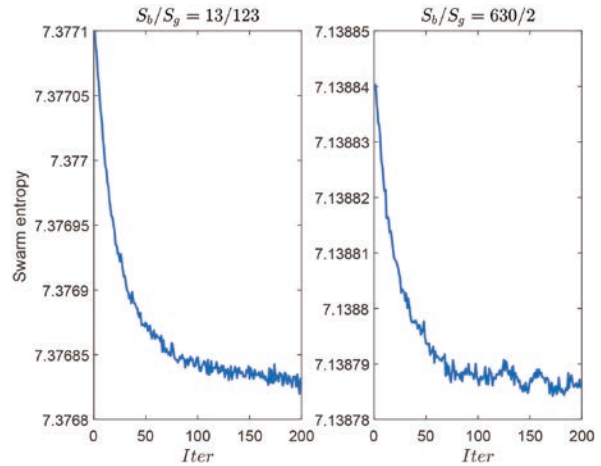


Figure 10. MSBA swarm entropy for BCPS problem with 50 workstations for $S_b/S_g = 13/123$ and $S_b/S_g = 630/2$ ($\lambda = 1600$, $n_e = 177$, $n_b = 80$).

Figure 9 shows the $SIE_t(S)$ value for the population of the MSBA algorithm for the small and high LT values. It can be noted that entropy practically does not change for the small LT value, while for high LT value entropy decreases indicating loss of population diversity. This observation proves that low LT value allows better exploitation of the promising regions of search space.

For $S_b/S_g = 630/2$ the solutions space is searched only by the two independent swarms, which favours fast convergence, but contributes to the small improvement of the objective function value. It confirms the observation (Figure 10) that entropy at the beginning is relatively low (i.e. the objective function values are similar) and changes a little until the end of the algorithm (about $5 \cdot 10^{-5}$). The reversed phenomenon can be observed for $S_b/S_g = 13/123$ when the solutions space is searched by the 123 independent swarms with size 13, which improves exploration of the solutions space and enables obtaining better solutions. In this case, at the beginning entropy is relatively high (i.e. objective function values are widely spread) and changes significantly until the end of the algorithm because swarms exploit different promising regions of the solutions space.

8 Conclusion

The presented paper was devoted to the examination of the opportunities of the application of the Bees Algorithm implemented on the CUDA platform for the optimisation of the Conveyor Belt Production Systems modelled as the QAP problem.

The proposed Multi-Swarm Bees algorithm, opposite to its CPU version, enables concurrent processing of the solutions (bees) in the swarm. In the MSBA algorithm each bee (solution) is processed separately in the thread (*kernel*) and the information about the results obtained by the bee is exchanged only within the solutions belonging to the same block. The swarms located in different blocs do not exchange any information between each other.

The presented experiments prove that the acceleration offered by the MSBA algorithm on the GPU platform enables fast, near-optimal, and effective tasks switching in the Conveyor Belt Production Systems.

The results obtained by these two algorithms show that selection of the appropriate parameters of the MSBA algorithm, such as the block size, the grid size, *life expectancy* and, especially, the size of the neighbourhood enable obtaining from two to three times better results than the CPU version of the BA algorithm. It should be noted that the mono-swarm BA algorithm for small values of the $n_e = 25$

and $n_b = 18$ obtains better results than the MSBA algorithm. However, the MSBA algorithm radically outperforms the BA algorithm if the neighbourhood size is about four times bigger. For example, for the $LT = 200$, $S_b/S_g = 179/9$, $n_e = 177$ and $n_b = 80$ the average gap E_{avr} equals 0.59% which is three times smaller and obtained two times faster in comparison to the BA algorithm. The large size of the neighbourhood and the large *life expectancy* guarantee good results of the optimisation, several times better than the results obtained by the CPU version of the BA algorithm. Furthermore, if the number of independent swarms S_g increases, then the relative gap E_{avr} decreases, despite the fact that the total number of the examined solutions is the same.

Moreover, the obtained results show that the proposed MSBA algorithm for the CUDA platform can offer considerable acceleration ($\eta = 21.2$) of the calculations in comparison to the BA algorithm running on the CPU.

Nevertheless, obtaining better results by the MSBA algorithm requires keeping the size of the neighbourhood relatively large, which can cause the elongation of the calculation time.

Swarm entropy presented in Figure 9 and Figure 10 shows that the *life expectancy* (LT) can be successfully used for controlling exploration-exploitation trade-off for the MSBA algorithm. Furthermore, if the number of the independent swarms is relatively large, then the algorithm obtains better results.

The lesson learned shows that a low-end GPU device can be successfully applied to optimisation of the real life $\mathcal{NP}$ -hard discrete problems defined for the Belt Conveyor Production Systems.

The results show that great emphasis should be placed on the local search during implementation of the multi-swarm population algorithms. The future work will be devoted to developing new local search procedures for the multi-swarm algorithms running on the CUDA platform and to shortening of the time computation by reduction of the divergence which occurs when different instructions are executed by threads at the same time. It leads to decreasing of the computation efficiency as some cores run their common instructions while the others remain idle.

References

- [1] Banharnsakun, A., Achalakul, T., Sirinaovakul, B.: Abc-gsx: A hybrid method for solving the traveling salesman problem. In: *Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on*. pp. 7–12 (10 2010)
- [2] Baykasoglu, A., Ozbakor, L., Tapkan, P.: Artificial bee colony algorithm and its application to generalized assignment problem. In: Chan, F., Tiwari, M. (eds.) *Swarm Intelligence, Focus on Ant and Particle Swarm Optimization*, chap. 08. I-Tech Education and Publishing, Vienna, Austria (10 2007)
- [3] Ben-Daya, M., Al-Fawzan, M.: A tabu search approach for the flow shop scheduling problem. *European Journal of Operational Research* **109**(1), 88–95 (1998)
- [4] Bölte, A., Thonemann, U.W.: Optimizing simulated annealing schedules with genetic programming. *European Journal of Operational Research* **92**(2), 402–416 (1996)
- [5] Burkard, R., Karisch, S., Rendl, F.: Qaplib-a quadratic assignment problem library. *Journal of Global Optimization* **10**(4), 391–403 (1997)
- [6] Burkard, R.E., Çela, E.: Heuristics for biquadratic assignment problems and their computational comparison. *European Journal of Operational Research* **83**(2), 283–300 (1995)
- [7] Burke, E.K., Gustafson, S., Kendall, G.: Diversity in genetic programming: an analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation* **8**(1), 47–62 (2004). DOI: 10.1109/TEVC.2003.819263
- [8] Chmiel, W., Szwed, P.: Bees algorithm for the quadratic assignment problem on cuda platform. In: *Man–Machine Interactions 4, Advances in Intelligent Systems and Computing*, vol. 391, pp. 615–625. Springer International Publishing (2016)
- [9] Chong, C.S., Sivakumar, A.I., Low, M.Y.H., Gay, K.L.: A bee colony optimization algorithm to job shop scheduling. In: *Proceedings of the 2006 Winter Simulation Conference* (2006)
- [10] Dou, Y., Jin, C., Dou, Y., Fan, H., Zhang, X.: Hybrid coyote optimization with differential evolution and its application to the estimation of solar photovoltaic cell parameters. *Journal of Artificial Intelligence and Soft Computing Research* **15**(4), 385–412 (2025). DOI: 10.2478/jaiscr-2025-0019, <https://doi.org/10.2478/jaiscr-2025-0019>
- [11] Elashmawi, W.H., Sheta, A., Alqadi, B.S., Abdelminaam, D.S., Alsekait, D.M.: A new version of the golden eagle optimizer algorithm and its application for solving a trio-objective skillful team formation problem in a social network. *Journal of Artificial Intelligence and Soft Computing Research* **15**(4), 357–384 (2025). DOI: 10.2478/jaiscr-2025-0018, <https://doi.org/10.2478/jaiscr-2025-0018>
- [12] F. Rubio, A. de la Encina, P.R., Rodríguez, I.: Eden bees: Parallelizing artificial bee colony in a functional environment. *Procedia Computer Science* **18**, 661–670 (2013)
- [13] Fatih Tasgetiren, M., Sevkli, M., Liang, Y.C., Gencyilmaz, G.: Particle swarm optimization algorithm for single machine total weighted tardiness problem. In: *2004 Congress on Evolutionary Computation*. vol. 2, pp. 1412–1419 (7 2004)
- [14] Gupta, J.N.D., Schaller, J.E.: Minimizing flow time in a flow-line manufacturing cell with family setup times. *Journal of the Operational Research Society* **57**(2), 163–176 (2006)
- [15] Huang, X., Yin, J., Deng, G.: A hybrid discrete artificial bee colony optimization algorithm for the no-wait job shop problem with tardiness criterion. *Journal of Algorithms & Computational Technology* **19**, 17–27 (2025). DOI: 10.1177/17483026251322104, <https://doi.org/10.1177/17483026251322104>
- [16] J., S.K.: Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing* **2**(1), 33–45 (1990)
- [17] Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N.: A comprehensive survey: artificial bee colony (abc) algorithm and applications. *Artificial Intelligence Review* **42**(1), 21–57 (2012)
- [18] Karimi, N., Zandieh, M., Najafi, A.: Group scheduling in flexible flow shops: a hybridised approach of imperialist competitive algorithm and electromagnetic-like mechanism. *International Journal of Production Research* **49**, 4965–4977 (2011)
- [19] Kirk, D., Hwu, W.: *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. (2010)
- [20] Koopmans, T.C., Beckmann, M.J.: Assignment problems and the location of economic activities. *Econometrica* **25**, 53–76 (1957)
- [21] Lewicki, A., Panczer, K., Tadeusiewicz, R.: The Use of Strategies of Normalized Correlation in the Ant-Based Clustering Algorithm, pp. 637–644. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

- [22] Li, J., Pan, Q., Xie, S.: Flexible job shop scheduling problems by a hybrid artificial bee colony algorithm. In: 2011 IEEE Congress of Evolutionary Computation (CEC). pp. 78–83 (7 2011)
- [23] Lian, Z., Gu, X., Jiao, B.: A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. *Applied Mathematics and Computation* **175**(1), 773–785 (2006)
- [24] Lourenço, H., Martin, O., Stützle, T.: *Iterated Local Search: Framework and Applications*, pp. 363–397. Springer US, Boston, MA (2010)
- [25] Luo, G.H., Huang, S.K., Chang, Y.S., Yuan, S.M.: A parallel bees algorithm implementation on gpu. *Journal of Systems Architecture* **60**(3), 271–279 (2014)
- [26] Luo, J., Shen, Y.: Energy efficiency optimization of belt conveyor for material scheduling problem. In: IEEE International Conference on Information and Automation. pp. 122–127 (8 2015)
- [27] Lv, Y., Zhang, J., Qin, W.: A genetic regulatory network-based method for dynamic hybrid flow shop scheduling with uncertain processing times. *Applied Sciences* **7**(1) (2017)
- [28] Mahmoodi, F., Dooley, K.: A comparison of exhaustive and non-exhaustive group scheduling heuristics in a manufacturing cell. *International Journal of Production Research* **29**(9), 1923–1939 (1991)
- [29] Nawaz, M., Ensore, E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* **11**(1), 91–95 (1983)
- [30] Ni, Y., Liu, W., Du, X., Xiao, R., Chen, G., Wu, Y.: Evolutionary optimization approach based on heuristic information with pseudo-utility for the quadratic assignment problem. *Swarm and Evolutionary Computation* **87**, 101557 (2024). DOI: <https://doi.org/10.1016/j.swevo.2024.101557>, <https://www.sciencedirect.com/science/article/pii/S2210650224000956>
- [31] Ogbu, F., Smith, D.: Simulated annealing for the permutation flowshop problem. *Omega* **19**(1), 64–67 (1991)
- [32] Pan, Q.K., Ruiz, R.: A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research* **40**(1), 117–128 (Jan 2013)
- [33] Parpinelli, R., Benitez, C., Lopes, H.: *Handbook of Swarm Intelligence: Concepts, Principles and Applications*, chap. Parallel Approaches for the Artificial Bee Colony Algorithm, pp. 329–345. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
- [34] Petalas, Y.G., Parsopoulos, K.E., Vrahatis, M.N.: Entropy-based memetic particle swarm optimization for computing periodic orbits of nonlinear mappings. In: 2007 IEEE Congress on Evolutionary Computation. pp. 2040–2047 (11 2007). DOI: 10.1109/CEC.2007.4424724
- [35] Pham, D.T., Castellani, M.: Benchmarking and comparison of nature-inspired population-based continuous optimisation algorithms. *Soft Computing* **18**(5), 871–903 (2014). DOI: 10.1007/s00500-013-1104-9, <https://doi.org/10.1007/s00500-013-1104-9>
- [36] Qiao, Z., Heidari, A.A., Zhao, X., Chen, H.: An evolutionary neural architecture search method accelerated by multi-fidelity evaluation and genetic decision controller. *Journal of Artificial Intelligence and Soft Computing Research* **15**(4), 413–446 (2025). DOI: 10.2478/jaiscr-2025-0020, <https://doi.org/10.2478/jaiscr-2025-0020>
- [37] Rajendran, C., Ziegler, H.: Two ant-colony algorithms for minimizing total flowtime in permutation flowshops. *Computers & Industrial Engineering* **48**(4), 789–797 (2005)
- [38] Rosca, J.: Entropy-driven adaptive representation. In: *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*. pp. 23–32. Morgan Kaufmann (1995)
- [39] Salmasi, N., Logendran, R., Skandari, M.R.: Total flow time minimization in a flowshop sequence-dependent group scheduling problem. *Computers & Operations Research* **37**(1), 199–212 (2010)
- [40] Saravanan, M., Haq, A.N.: A scatter search method to minimise makespan of cell scheduling problem. *International Journal of Agile Systems and Management* **3**(1-2), 18–36 (2008)
- [41] Sayadi, M., Ramezani, R., Ghaffari-Nasab, N.: A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations* **1**(1), 1–10 (2010)
- [42] Skorin-Kapov, J., Vakharia, A.: Scheduling a flow-line manufacturing cell: a tabu search approach. *International Journal of Production Research* **31**(7), 1721–1734 (1993)
- [43] Solimanpur, M., Elmi, A.: A tabu search approach for group scheduling in buffer-constrained flow shop cells. *International Journal of Computer Integrated Manufacturing* **24**(3), 257–268 (2011)

- [44] Souza, D., Monteiro, G., Martins, T., Dmitriev, V., Teixeira, O.: Pso-gpu: Accelerating particle swarm optimization in cuda-based graphics processing units. In: *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*. pp. 837–838. GECCO '11, ACM, New York, NY, USA (2011)
- [45] Sridhar, J., Rajendran, C.: A genetic algorithm for family and job scheduling in a flowline-based manufacturing cell. *Computers & Industrial Engineering* **27**(1), 469–472 (1994)
- [46] Subotic, M., Tuba, M., Stanarevic, N.: Parallelization of the artificial bee colony (abc) algorithm. In: *Proceedings of 11th WSEAS International Conference on Evolutionary Computing*. pp. 191–196. NN'10/EC'10/FS'10, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA (2010)
- [47] Sundar, S., Singh, A.: A swarm intelligence approach to the quadratic minimum spanning tree problem. *Information Sciences* **180**(17), 3182–3191 (2010)
- [48] Szeto, W.Y., Wu, Y., Ho, S.C.: An artificial bee colony algorithm for the capacitated vehicle routing problem. *European Journal of Operational Research* pp. 126–135 (2011)
- [49] Szwed, P., Chmiel, W.: Multi-swarm pso algorithm for the quadratic assignment problem. In: Malley, R.E.O. (ed.) *The 14th International Conference on Artificial Intelligence and Soft Computing ICAISC 2015* (2015)
- [50] Szwed, P., Chmiel, W., Kadłuczka, P.: Opencl implementation of pso algorithm for the quadratic assignment problem. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L., Zurada, J. (eds.) *Artificial Intelligence and Soft Computing. Lecture Notes in Computer Science*, vol. 9120, pp. 223–234. Springer International Publishing (2015)
- [51] Tadeusiewicz, R., Lewicki, A.: The ant colony optimization algorithm for multiobjective optimization non-compensation model problem staff selection. In: Cai, Z., Hu, C., Kang, Z., Liu, Y. (eds.) *Advances in Computation and Intelligence*. pp. 44–53. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
- [52] Teodorovic, D., Lucic, P., Markovic, G., Dell'Orco, M.: Bee colony optimization: Principles and applications. In: *2006 8th Seminar on Neural Network Applications in Electrical Engineering*. pp. 151–156 (2006)
- [53] Thammano, A., Phu-ang, A.: A hybrid artificial bee colony algorithm with local search for flexible job-shop scheduling problem. *Procedia Computer Science* **20**, 96–101 (2013)
- [54] Vakharia, A., Chang, Y.L.: A simulated annealing approach to scheduling a manufacturing cell. *Naval Research Logistics (NRL)* **37**(4), 559–577 (1990)
- [55] Vijayender, R., Narendran, T.T.: Heuristics and sequence-dependent set-up jobs in flow line cells. *International Journal of Production Research* **41**(1), 193–206 (2003)
- [56] Wang, L., Zhao, X., Wu, P.: Resource-constrained emergency scheduling for forest fires via artificial bee colony and variable neighborhood search combined algorithm. *IEEE Transactions on Intelligent Transportation Systems* **25**(6), 5791–5806 (2024). DOI: 10.1109/TITS.2023.3338017
- [57] Wheeler, C.: Evolutionary belt conveyor design-optimizing costs. *Bulk Material Handling by Conveyor Belt* 7 pp. 3–11 (01 2008)
- [58] XingBao, L., ZiXing, C.: Artificial bee colony programming made faster. In: *2009 Fifth International Conference on Natural Computation*. vol. 4, pp. 154–158 (8 2009)
- [59] Yeh, W.C., Hsieh, T.J.: Solving reliability redundancy allocation problems using an artificial bee colony algorithm. *Computers & Operations Research* **38**(11), 1465–1473 (2011)
- [60] Zajecka, M., Mastalerczyk, M., Chong, S.Y., Yao, X., Kwiecień, J., Chmiel, W., Dajda, J., Kisiel-Dorohinicki, M., Byrski, A.: Portfolio optimization with translation of representation for transport problems. *Journal of Artificial Intelligence and Soft Computing Research* **15**(1), 57–75 (2024). DOI: 10.2478/jaiscr-2025-0004, <https://doi.org/10.2478/jaiscr-2025-0004>
- [61] Zhang, S., Xia, X.: Modeling and energy efficiency optimization of belt conveyors. *Applied Energy* **88**(9), 3061–3071 (2011)



Wojciech Chmiel received the M.Sc. degree in Electronics and Control Engineering, the Ph.D. degree in Automatic Control and Robotics, and the habilitation degree from AGH University of Science and Technology (AGH-UST) in Krakow, Poland. He is currently an associate professor at AGH-UST and serves as the leader of several research

projects. His research interests include operations research, artificial intelligence, discrete optimization, workflow management systems, computational intelligence, and decision-making.

<https://orcid.org/0000-0002-4773-9123>



Pawel Kolendo is a Ph.D. student at AGH University of Krakow, specializing in evolutionary computation. His research focuses on automated algorithm configuration and the integration of Large Language Models (LLMs) into algorithm design to enhance optimization performance.

<https://orcid.org/0009-0008-3685-280X>



Joanna Kwiecień - Associate Professor at AGH University of Krakow. She is a specialist on methods and algorithms of artificial intelligence (mainly swarm algorithms). Her interests include also modeling and optimization of real systems, discrete optimization, and decision making.

<https://orcid.org/0000-0002-8225-7605>