Heuristic Optimization for Job Shop Scheduling in a Flexible Manufacturing System

Hisham Alkhalefah
Institute
King Saud University
Riyadh Advanced Manufacturing
11421, Saudi Arabia
halkahlefah@ksu.edu.sa

Usama Umer
Institute
King Saud University
Riyadh Advanced Manufacturing
11421, Saudi Arabia
uumer@ksu.edu.sa

Mustufa Haider Abidi
Institute
King Saud University
Riyadh Advanced Manufacturing
11421, Saudi Arabia
mabidi@ksu.edu.sa

Ahmed Elkaseer*
Department of Production Engineering and Mechanical Design
Faculty of Engineering,
Port Said University
Port Said, 42523, Egypt

Institute for Automation and Applied Informatics
Karlsruhe Institute of Technology
Karlsruhe, 76344 Germany
ahmed.elkaseer@kit.edu
*Correspondence: ahmed.elkaseer@kit.edu

Abstract

Job Shop Scheduling Problem (JSSP) is one of the most difficult discrete combinatorial problems in optimization science and an important challenge in Flexible Manufacturing Systems (FMS): having a substantial impact on minimizing manufacturing time, increasing productivity, and optimizing tool handling. However, the literature studies mostly emphasized production efficiency and work load optimization with-out considering design of the software optimization algorithms. Moreover, the literature studies do not usually include detailed validation of the use of different optimization algorithms with JSSP. Due to these limitations, the authors have given more attention to the optimization of JSSP. This research explores the relative potential of different optimization algorithms, and study the capabilities of software design to develop a useful general solution for JSSPs. In this paper, the authors introduce a software design using genetic and random search algorithms to find the optimal sequences of jobs in JSSPs. Moreover,

the authors have produced a convenient design of soft-ware to provide stable, workable, and upgradable algorithms. Both the proposed algorithms were tested against well-known benchmark tests in JSSP.
Better results for some cases were obtained using the developed algorithms when compared with the results reported in the literature. The testing revealed both pros and cons of the algorithms; the genetic algorithm is able to optimize complex problems of JSSP. However, it requires considerable execution time +150 seconds. Otherwise, the random search algorithm reached a solution in a very short time, within 0.1 seconds, but with lower accuracy and is limited to the optimization of simple problems.

Keywords
Job Shop Problem, Flexible Manufacturing System, Genetic Algorithm, Random Search Method, Evolutionary Algorithm, Heuristic Algorithms, Software Design Pattern.

1. Introduction

A flexible manufacturing system (FMS) is an approach to production whereby a manufacturing system can adapt to significant alterations in the products being produced (Saleh et al. 2009) [1]. FMSs are computerized systems, i.e. smart systems or cyber-physical systems (CPSs), that are configurable and are capable of coping with changes in, for example, materials and/or production levels to meet changing market demand, reduce inventory resources and lower production costs (Saleh et al. 2009 and Tran et al. 2019). The FMS has to be smart enough to deal with complex cases in order to manage/optimize production including as line settings, buffer size and job shop scheduling. The jobs scheduler in a FMS has to optimize scheduling jobs between sub-system components, i.e. machines and tool handling, to determine the optimal sequence of jobs on different machines. Moreover, the jobs scheduler has to consider machine inter-dependency in the fabrication processes as it optimizes the JSSP.

Heuristic optimization algorithms can be developed to enhance FMS techniques and find optimal solutions for managing the FMS operations as part of the JSSP (Low et al. 2006). In order to optimize FMS techniques, the problem has to be accurately expressed as a mathematical model then the optimization algorithm can provide the optimal set for the given model. The mathematical description of the objective function of the expression is to be minimized/maximized via the optimization algorithm. Moreover, the model should describe the constraints of the problem such as machine dependency and number/s of operation per machine/s (Qudeiri et al. 2016). Numerical technique can be used to solve/optimize the mathematical model but can consume substantial computing resources doing so due to the complexity of the problem. JSSP is inherently a com-plex problem and many can be described as nondeterministic polynomial- (NP) complete problems (Sahu et al. 1993 and Qudeiri et al. 2015). Where, a NP-complete problem is defined as a computational problem which does not have an efficient solution, and its solution is calculated and verified without particular rule/s. Heuristic algorithms have been designed to deal with NP-complete problems such as JSSP, these would include problems such as the travelling salesman problem, satisfiability problems, and graph-covering problems (Andrew et al. 2014).

With the rapid development of FMSs, the scheduling of jobs and tool magazines becomes a big opportunity. Due to the complexity of FMS, scheduling algorithms have been utilized to solve many flexible manufacturing problems such as job sequencing, machine selection, tool handling, raw materials handling and job descriptions [8]. During FMS processing, the real-time handling of parts and tools is a major dynamic consideration, with re-searchers in FMS developing optimization algorithms for the tool magazines based on relevant simplifying assumptions (Peynirci et al. 2006, Yang et al. 2016, Kavitha et al. 2018). For example, the constraints of the scheduling algorithm change based on changing requirements, if the cost of parts is higher than the cost of the required cutting tools, the developed framework must optimize the tool copy configuration and tool handling during the process (Jun et al. 1999). Nevertheless, these studies are limited to optimizing the FMS scheduler without impacting valuation of the optimization methods from the computational point. Al-so, these studies do not cover the impact of the software design on the development of the optimization algorithm.

In this paper, the authors introduce a novel development and implementation of the genetic and random search algorithms to optimize/solve the JSSP within an FMS framework. This development is based on a convenient design of software, which makes the solution up-gradable and suited to different requirements such as the addition of new optimization algorithms, changes in input data features/formats, industry 4.0 integration, etc. In the developed software, the model of FMS is defined to calculate the make span of the jobs while meeting the constraints imposed

by the machines and products. The make span will be minimized using the proposed optimization algorithms to minimize any production delays and in-crease productivity.

This paper is organized as follows; the first section is the introduction followed by the theoretical foundation of the algorithms, then the problem definition and proposed solution are discussed. After that, the results are present-ed and discussed. Finally, the conclusions are given and possible future work proposed.

2. Theoretical Foundation of Optimization

Heuristic algorithms are a group of computational procedures, which are designed to solve complex problems quickly, and calculate an approximate solution (Das et al. 2011). Classic methods that can provide an exact solution are invariably very time consuming if the problem is in any way complex. Also, the classic method may not be capable of finding exact solution to the more complex problems, such as NP-complete problems, and certainly not finding a solution quickly (Wittek. 2014, Cano at al. 2021 and Miloševi. 2015). For complex problems the results of heuristic algorithms may not provide an exact solution but can find an approximate solution which can be very convenient. In particular, the heuristic algorithms usually find acceptable solutions in a reasonable amount of time for clearly specified dependent and independent problems (Marzouki et al. 2018). Heuristic algorithms use a trial and error approach when optimizing FMSs to find an acceptable solution in reasonable time as shown in Figure 1 (Arora. 2016 and Nedwed et al. 2017).

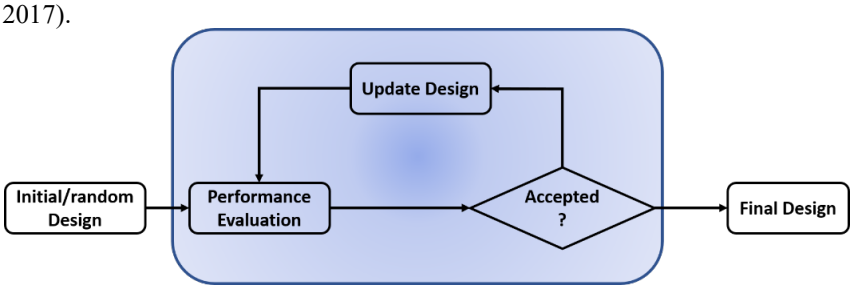

Figure 1: Workflow of optimization algorithms (Arora. 2016).

A genetic algorithm is one of the most effective heuristic algorithms and is based on the Darwinian theory of natural selection as presented in Figure 2 (a). In particular, the algorithm finds the optimal points in the search space in a way, which is similar to how genes enhance themselves in nature. The algorithm starts with a random population of individuals with different levels of fitness. The individuals with high levels of fitness are selected to crossover and reproduce into a new population with a higher fitness (Wittek. 2014 and Parjapati at al. 2015). The individuals contain the solution of the given problem and an individual's fitness refers to the individual's value in the objective function of the given problem (Nguyen et al. 2016).

On other hand side, the random search algorithm presented in Figure 2 (b) finds the optimal point on the distribution. It starts with a random solution and perform an iterative optimization to obtain the global optimum. This algorithm carried out heuristic solutions at every iteration, which are evaluated by the objective function and results are saved in a variables list. The algorithm returns the most effective value as a final solution.

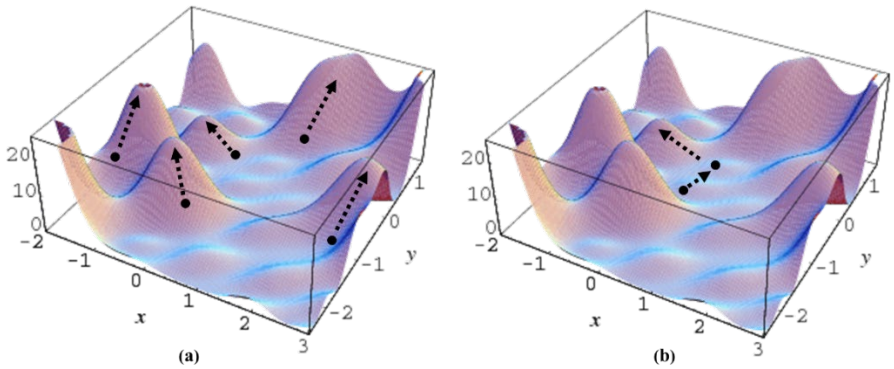

Figure 2: (a) Parallel solving of problem using genetic algorithm; (b) single search solution of problem using random search algorithm

3. Problem Definition and Proposed Solution

3.1. Problem Definition

The problem environment is as follows, there are a number of flexible manufacturing cells (FMCs) each containing a group of different machines. There are numerous products, which are fabricated via different sequences of jobs on different machines. There are two loading station for the material and Automatic Storage and Retrieval (AS/R) to manage the product during fabrication. To describe a solution suitable for the JSSP, assume a job shop scheduling problem with N workpieces and M machines as shown in Figure 3. The solution will be composed of only N x M possibilities if each workpiece is processed only once on each machine. For every operation O, the product uses machine M in operation J as shown in Table 1. The machines sequence of every task is described in Table 2.

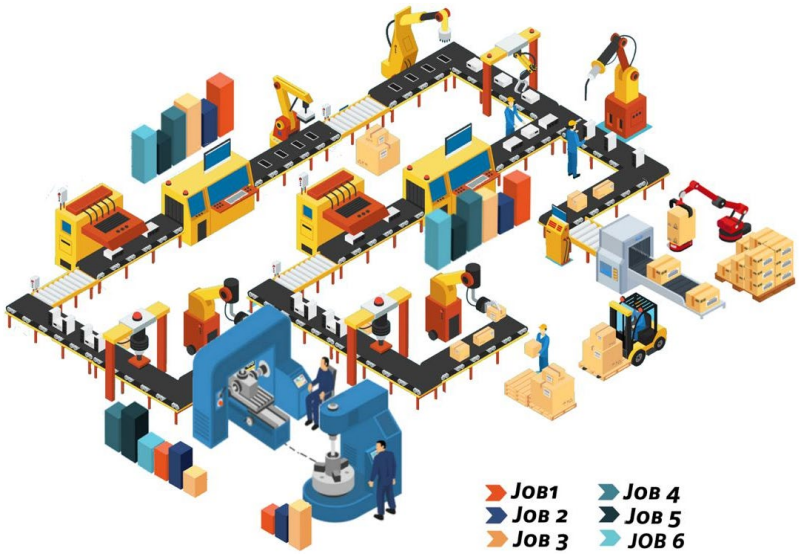

Figure 3: Proposed platform for the problem environment and components

Table 1: Example of JSSP

Operation	O1	O2	O3
Job 1	M2	M3	M1
Job 2	M1	M3	M2
Job 3	M3	M2	M1

The JSSP model was formulated as follows [9,10]:

$$M = \{M1, M2, M3, \ldots, Mm\} \quad (1)$$

$$J = \{J1, J2, J3, \ldots, Jn\} \quad (2)$$

$$O = M \times J \quad (3)$$

$$Cij: M \times J \rightarrow [0, +\infty] \quad (4)$$

$$\mathbf{T = 1 + \frac{\sum_i li}{\sum_{j,k} Pjm} = \frac{c.m}{\sum_{j,k} Pjk}} \quad (5)$$

where: M is Machines, J is Jobs, O is operations, C is scheduling efficiency, li is idle time, P_{jk} is processing time of operation J at machine M, c.m is the cost of manufacturing, and T is make span for production.
In order to test and validate a solution for the above model the authors utilised well-known and accepted benchmarks selected to include a large number of constraints on the JSSP in FMS (Dalle Molle Institute for Artificial Intelligence USI-SUPSI, https://people.idsia.ch//~monaldo/fjsp.html#Problem).

Table 2: Data set of JSSP for FMS

No.	Name	No. of jobs	No. of machines	Constraints
1	Mt10x	10	11	Every machine can perform 1 operation at same time.
2	setb4xyz	20	10	Every machine can perform 1 operation at same time.
3	Setb4xxx	10	5	Every machine can perform 1 operation at same time.
4	01a	10	5	Every machine can perform 1 operation at same time.
5	18a	20	10	Every machine can perform 1 operation at same time.
6	Mk04	15	8	Every machine can perform 2 operation. Each operation can be assigned to only one machine.
7	edata1-la25	10	10	Every machine can perform 1 operation at same time. Each operation can be assigned to only one machine. (edata1)
8	rdata1-la25	15	10	Every machine can perform 3 operations. A few operations can be assigned to different machines. (rdata1)
9	sdata1-orb10	10	10	Every machine can perform 1 operation. Most operations can be assigned to a small number of different machines (sdata1)
10	sdata2-ola25	15	10	Every machine can perform 1 operation. Most operations can be assigned to a small number of different machines (sdata2)
11	vdata-la25	15	10	Every machine can process 5 operation. Each operation can be assigned to many different machines (vdata1)
12	vdata-orb10	10	10	Every machine can process 5 operation. Each operation can be assigned to many different machines (vdata2)

3.2. Proposed Solutions and Software Implementation

In this solution, the proposed algorithms were developed based on adaptable design of software so that it was capable of increasing the number of features and interacting with other algorithms in the future. The software was developed using Python 3.6 programming language (Blank et al. 2020) via the PyCharm Integrated Development Environment IDE (PyCharm, https://www.jetbrains.com/pycharm/). To enable data handling and analysis, and graph/chart visualization, etc. Matplotlib and Plotly libraries were used Stančin et al. 2019. The design of the software was divided into 8 classes, of which one was responsible for specific feature of the software requirements. In Figure 4, the scheduler class is responsible for producing the optimized sequence of Jobs based on the inputs from the genetic algorithm class (ga_optimization) and random search algorithm class (rnd_optimization). Also, the machine and jobs data are handled via the processes class which describes the matrix [jobs x machines] based on its relations with both jobs and machines classes. The drawer class is responsible for printing the optimized time in the console of the PyCharm IDE and it also carried out the optimal scheduling sequences of jobs with machines (JxM matrix) in a visual form. In order to implement the optimization algorithms, the ga_optimization class responsible for performing the genetic algorithm steps and the random search class optimized the jobs sequences (See Figure 5 and Figure 6 for ga_optimization and rnd_optimization, respectively).

- During development, the author added some considerations as follows:
- All machines are added with their processing times in a list.
- The machines are not currently loaded.
- The machine has a working process but it does not work at full capacity.
- The machine performs a process at full capacity. However, there is an operation that can take more time than calculated previously

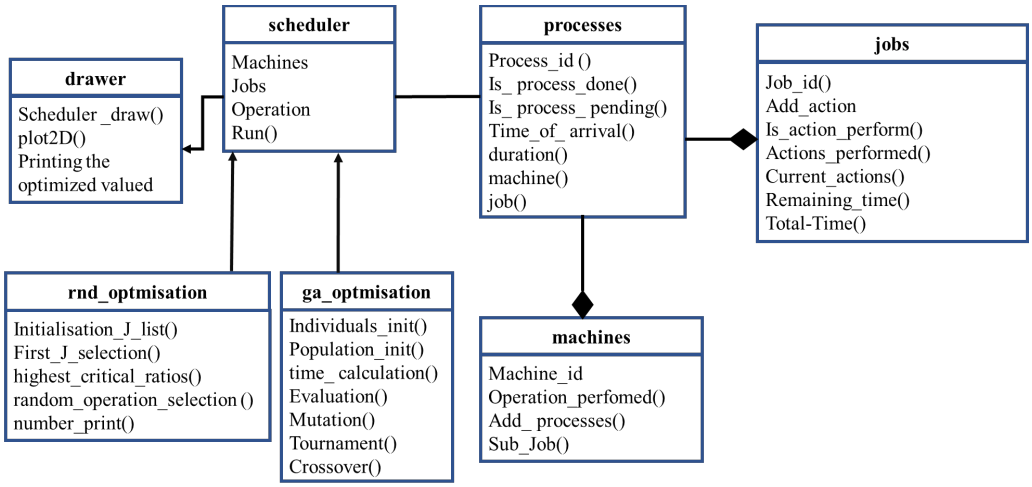

Figure 4: Class diagram of JSSP optimization using different optimization approaches

3.2.1.Genetic algorithm

In order to solve the problem, genetic and random search algorithms were used. In the genetic algorithm, the gene represented the sequences of jobs, and the product was represented by the chromosome. In this case the chromosome will be computed in terms of J x M genes where J refers to jobs and M refers to machines. Iteration by iteration, the algorithm finds the optimal sequence for every chromosome as shown in Figure 5. The idea behind this method is to express the chromosome as a set of work processing procedures for the work piece.

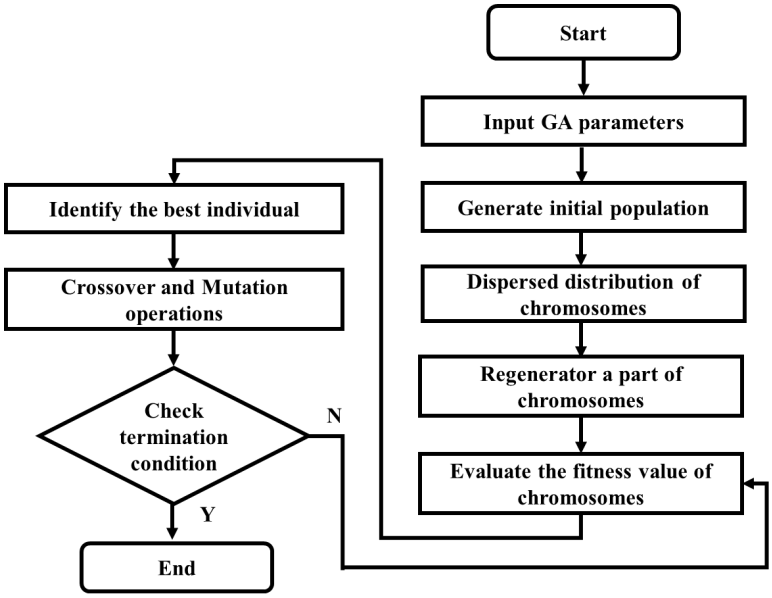

Figure 5: JSSP optimization via genetic algorithm

The user of the application has to define the population number and maximum number of generations after which the algorithm has to stop. To find the optimal settings, the author tried numerous permutations of population number and maximum number of generations, such as: 20 and 400, 20 and 600, 30 and 1000, 50 and 1500, 50 and 2000 respectively. During testing of different configuration, we find the minimum makespn time was carried out form the setting of 50 and 1500. Whereas, by increasing the generation +1500, we found the same results. the optimal results were found to be 50 and 1500. When starting the algorithm, the scheduler makes temporary copy of the list of jobs and machines. After that, the algorithm generated a random solution of the scheduling based on the population number, and then calculated every fitness function. The algorithm selected the two sequences with the highest fitness functions and performed a crossover to obtain enhanced sequences and then mutated it. After that, the algorithm

repeats the process. The fitness functions are calculated based on the desired objective function of the derived model (see Eq. 5).

3.2.2. Random Search

The random search optimization algorithm generates a random solution for the J x M matrix and tests it with the objective function. The next iteration compares the solution with the first and the better solution is saved. Then iteration by iteration, the best solution is saved and referred to as the new solution as shown in Figure 6. This algorithm is designed to find the optimal point in a short time without requiring a significant amount of computer power. Whereas this design is very useful with a simple or an intermediate problem it is not usually suited to complex problems.

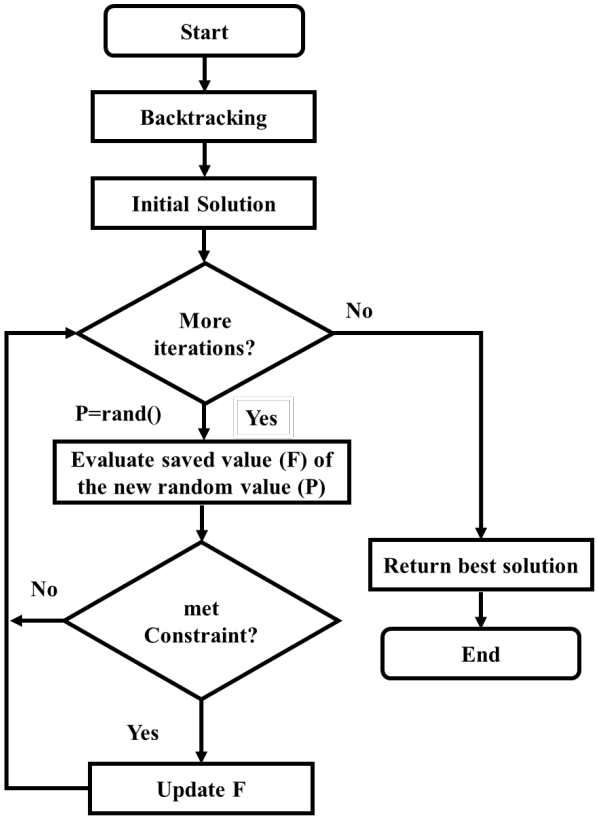

Figure 6: JSSP optimization via random search optimization

4. Results and discussion

The software results show that the genetic algorithm provided more accurate results than the random search algorithm. Whereas, the heuristic behavior of GA individuals resulted in different schedules for jobs which helped find the optimal sequence of jobs. The algorithms were tested against 12 widely used benchmarks obtained from the literature [16]. The benchmarks were the different schemas listed in Table 2 applicable to a flexible manufacturing environment. The results are shown in Table 3. The random search algorithm solves the simpler problems in a very short time, see Table 3 (items 1-5). The random search can solve complex problems (see Table 3 (items 6-12)). However, the algorithm did not give better results than the genetic. On the other hand, the genetic algorithm solved all 12 problems (see Table 3 (items 1-12)) but required a longer execution time. It is also worth stating that better results for some cases were obtained using the developed algorithms when compared with the results reported in the literature. In particular, for benchmarks Mt10x, setb4xyz, Setb4xxx, 01a, 18a and Mk04 generic algorithm successfully obtained the optimal job schedule with minimum production time (Makespan) even less that the minimum results previously obtained and reported in the literature.

Table 3: validation of genetic algorithm and random search algorithm with the benchmarks of Table 2

No.	Name	Results from literature (time units)	Genetic Algorithm Optimization (time units)	Genetic Algorithm, SW Execution Time (second)	Rns Optimization (time units)	Rns, SW Execution Time (second)
1	Mt10x	1137 (Daniëls. 2013) 1023 (Philip. 2005)	1015	65.354	1345	0.0403231
2	setb4xyz	1124 (Daniëls. 2013) 1044 (Philip. 2005)	1001	103.931	1214	0.0671564
3	Setb4xxx	1162 (Daniëls. 2013) 1030 (Philip. 2005)	1039	113.273	1232	0.08569610
4	01a	2975 (Daniëls. 2013)	2745	130.956	3308	0.1044707
5	18a	2664 (Daniëls. 2013)	2257.0	635.754	2384	0.2994417
6	Mk04	83 (Daniëls. 2013)	69.0	74.532	NA	NA

We now consider the first five specific cases, those numbered 1 to 5 in Tables 2 and 3. In every case the initial settings for the genetic algorithm were an initial population of 30 and maximum number of generations 1000.

4.1. Simulation results for Test case (1) (mtx10) 10 Jobs and 11 machines

The problem has 10 Jobs and 11 machines. Every machine can perform 1 operation. The algorithm found that, the minimum time to perform these jobs was 1015 time unit as shown in Figure 7 with sequences of jobs on ma-chines as shown in Figure 8. The algorithm selected the point 14 as optimal because it is the last optimal point attained before reaching the maximum number of generations. The X-axis of Figure 7 shows only the optimal points not all the distributions.

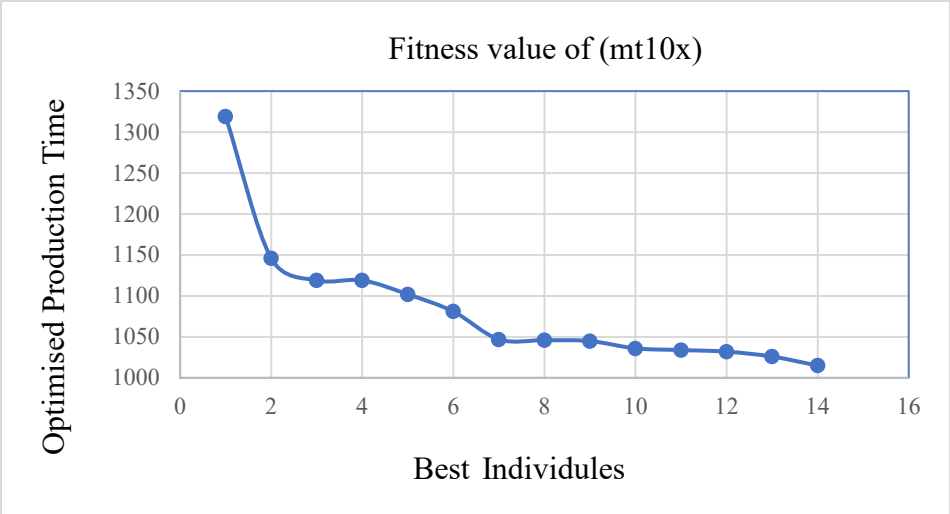

Figure 7: The optimization of the makespan for Test case (1) (mt10x) 10 Jobs and 11 machines

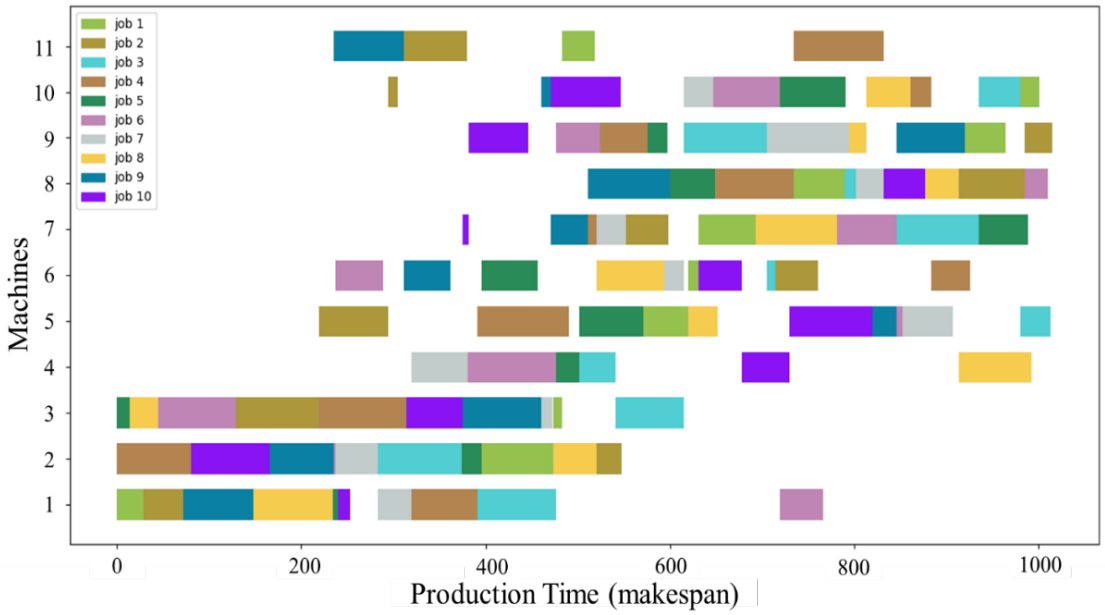

Figure 8: The optimized jobs sequence for Test case (1) (mt10x) 10 Jobs and 11 machines

4.2. Simulation results for Test case (2) (setb4xyz) 15 Jobs and 13 machines

Again, the problem has 15 Jobs and 13 machines. All machines can perform only one operation. Most operations can be assigned to a small number of different machines. The algorithm found the best work sequence was performed in 1015 time units as shown in Figure 9. The work sequence is shown in Figure 10. The algorithm saved point 4 as the optimal because it is the last optimal point before reaching the maximum number of generations.
The X-axis of Figure 9 shows the replacement of the optimal points only, not the entire distribution. By setting a higher value for the maximum number of generations, we found the optimum time to complete the jobs remained 1015 time units.

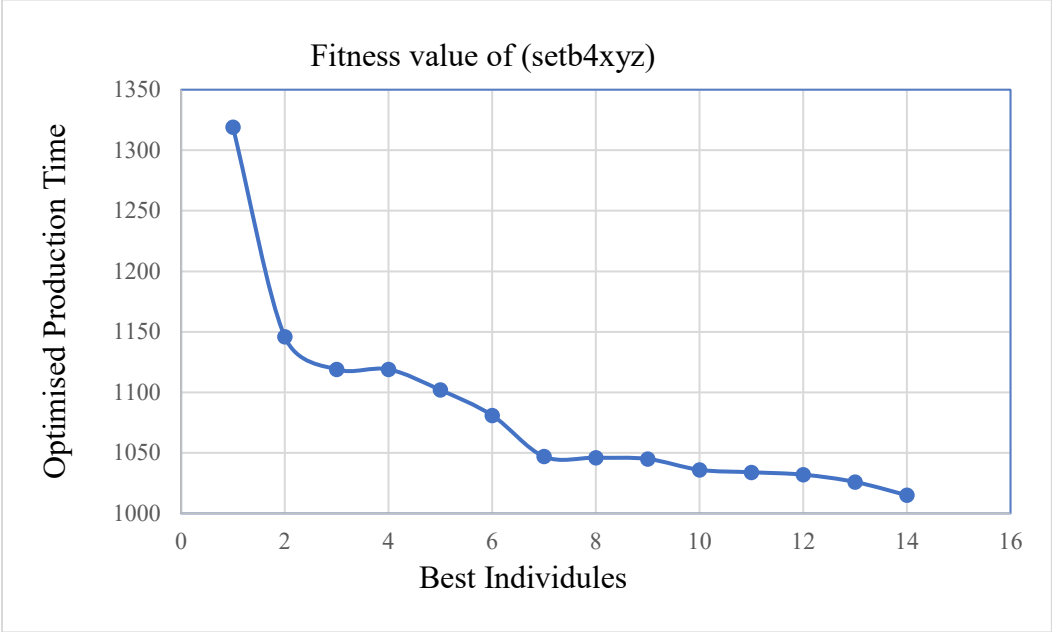

Figure 9: The optimization of the makespan for Test case (2) (setb4xyz) 15 Jobs and 13 machines

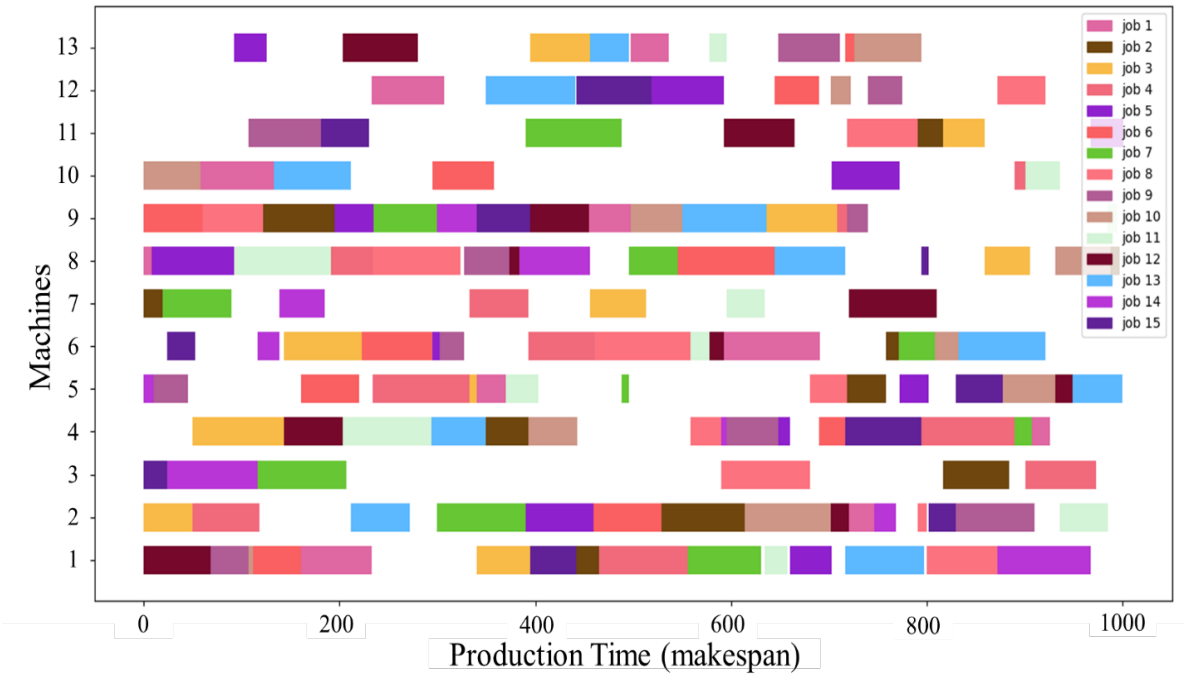

Figure 10: The optimized jobs sequence for Test case (2) (setb4xyz10) 15 Jobs and 13 machines

4.3. Simulation results for Test case (3) (setb4xxx) 15 Jobs and 13 machines

The problem has 15 Jobs and 13 machines. Every machine can perform a single operation. Most operations can be assigned to a small number of different machines. The algorithm found the optimum sequence was performed in 1039 time units as shown in Figure 11 with the best sequences of jobs as shown in Figure 12. The algorithm saved point 16 as optimal. As above, the X-axis of Figure 11 shows the replacement optimal points only. By setting a higher value for the maximum number of generations we found the time to complete the jobs was 1039 time units.

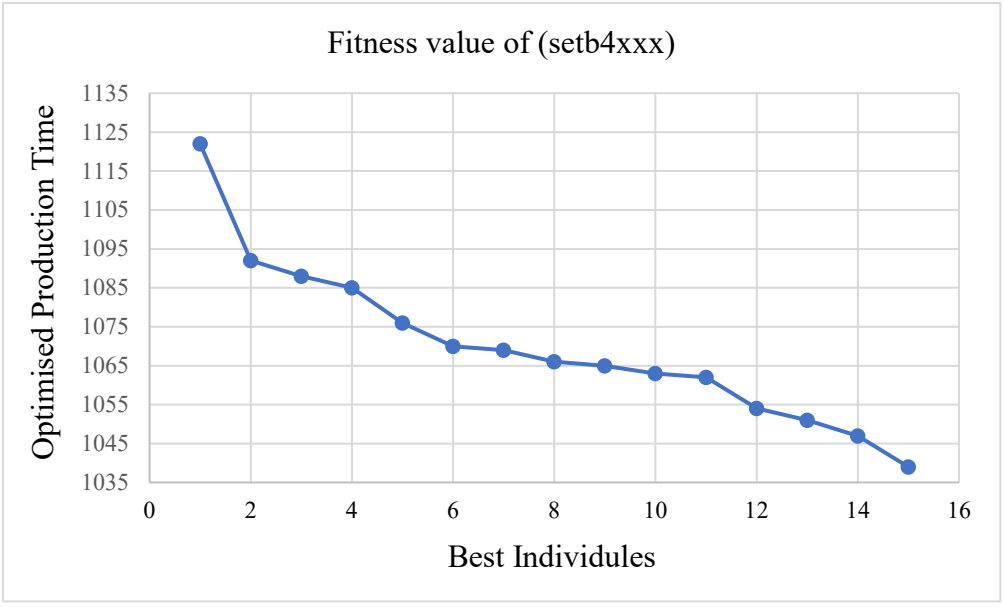

Figure 11: The optimization of the make span for Test case (3) (steb4xxx) 15 Jobs and 13 machines

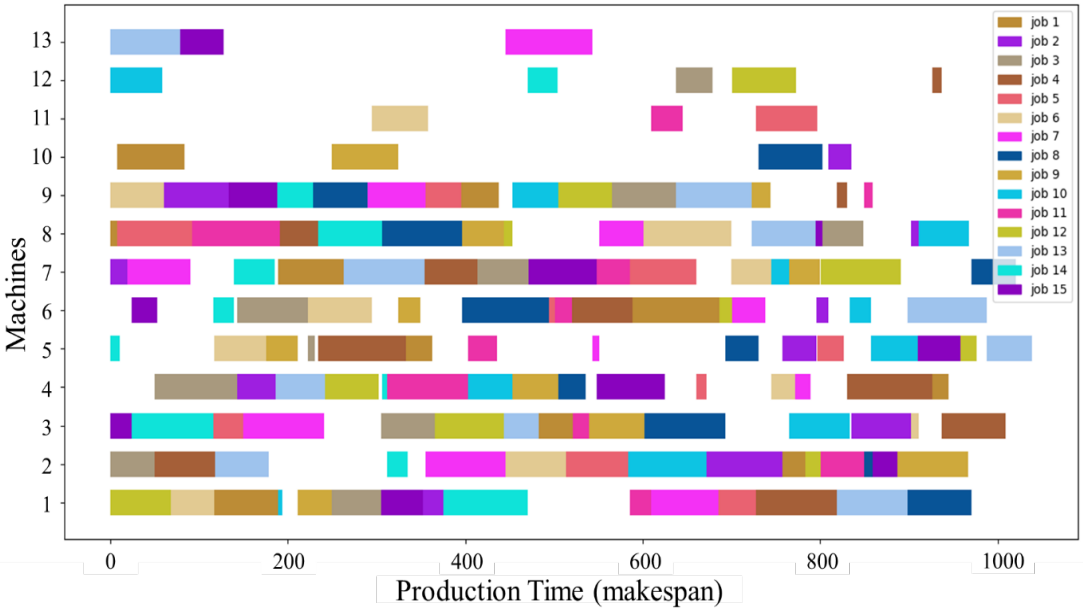

Figure 12: The optimized jobs sequence for Test case (3) (setb4xxx) 15 Jobs and 13 machines

4.4. Simulation results for Test case (4) (01a) 10 Jobs and 5 machines

The problem 10 Jobs and 5 machines. Every machine can process 1 operation. The genetic algorithm found the best sequence was performed in 2745 time units (see Figure 13) with the best sequences of job as shown in Figure 14. The algorithm saved the point 37 as the optimal because it is the last optimal point before reaching the maximum number of generations. The X-axis of Figure 13 shows only the replacement optimal points without showing all the distribution. By setting a higher value for the maximum number of generations we found the time to complete the jobs was 2745 time units.

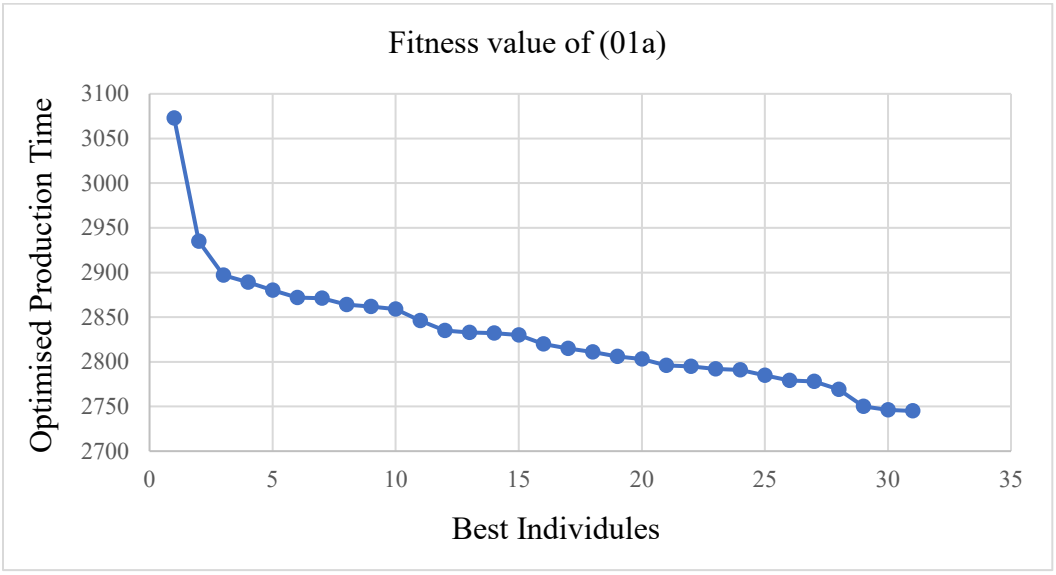

Figure 13: The optimization of the makespan for Test case (4) (01a) 10 Jobs and 5 machines

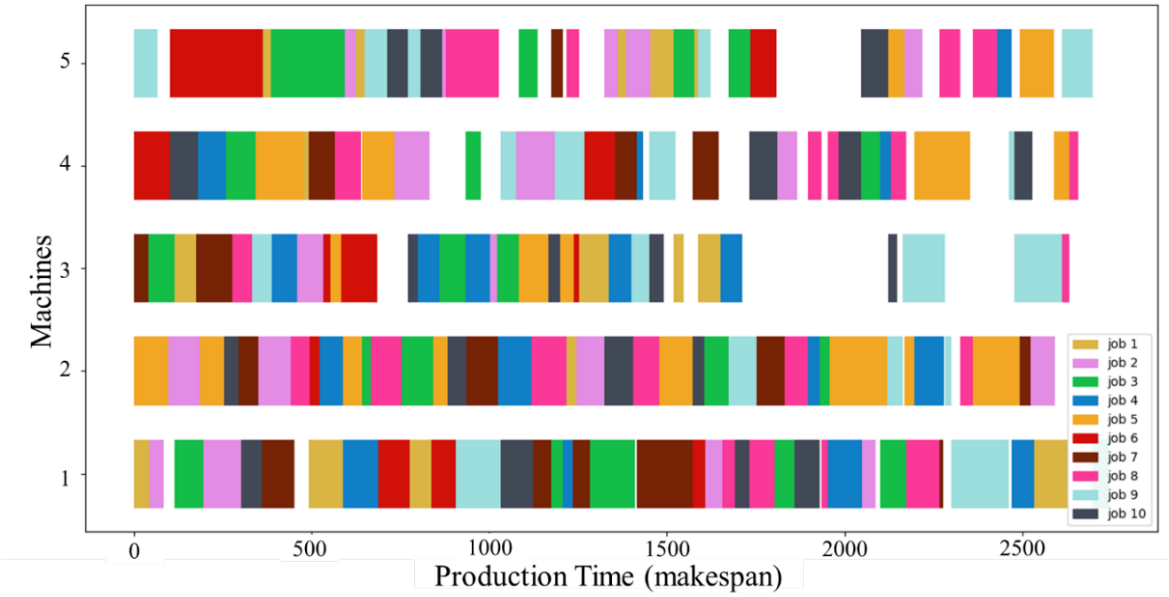

Figure 14: The optimized jobs sequence for Test case (4) (01a) 10 Jobs and 5 machines

4.5. Simulation results on Test case (5) (18a) 10 Machine and 20 Jobs

The simulation of Test case (11) was, for 10 Jobs and 10 machines. Every machine can process 1 operation. The genetic algorithm found the best sequence was performed in 2257 time units (see Figure 15) with the best sequences of jobs as shown in Figure 16. The algorithm saved point 42 as the optimal because it is the last optimal point before reaching the maximum number of generations. The X-axis of Figure 15 shows only the replacement optimal points without showing all the distribution. In this case setting a higher value for the maximum number of generations, did not improve the best sequence.

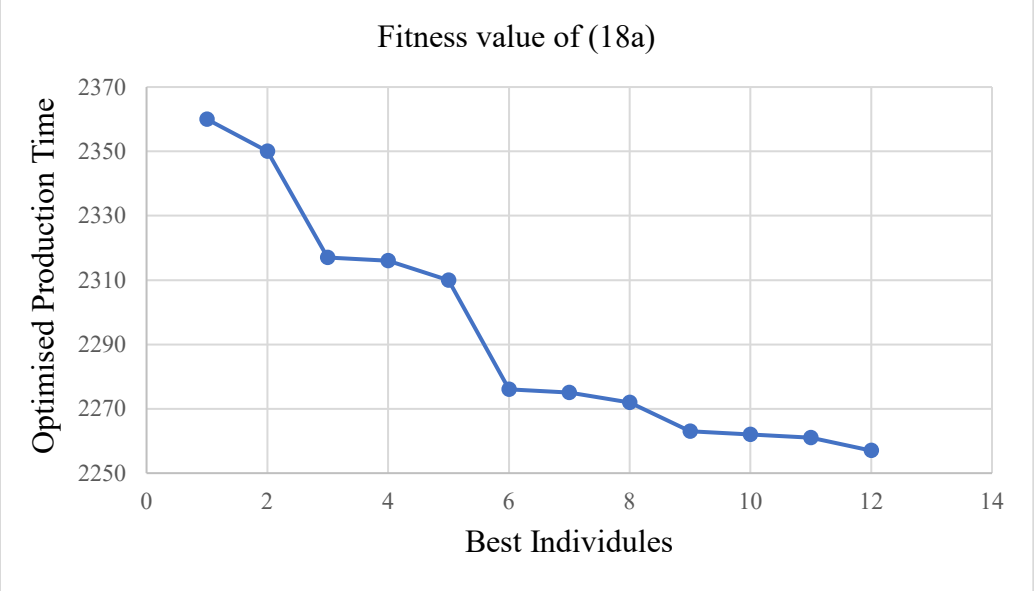

Figure 15: The optimization of the makespan for Test case (5) (18a) 10 Machine and 20 Jobs

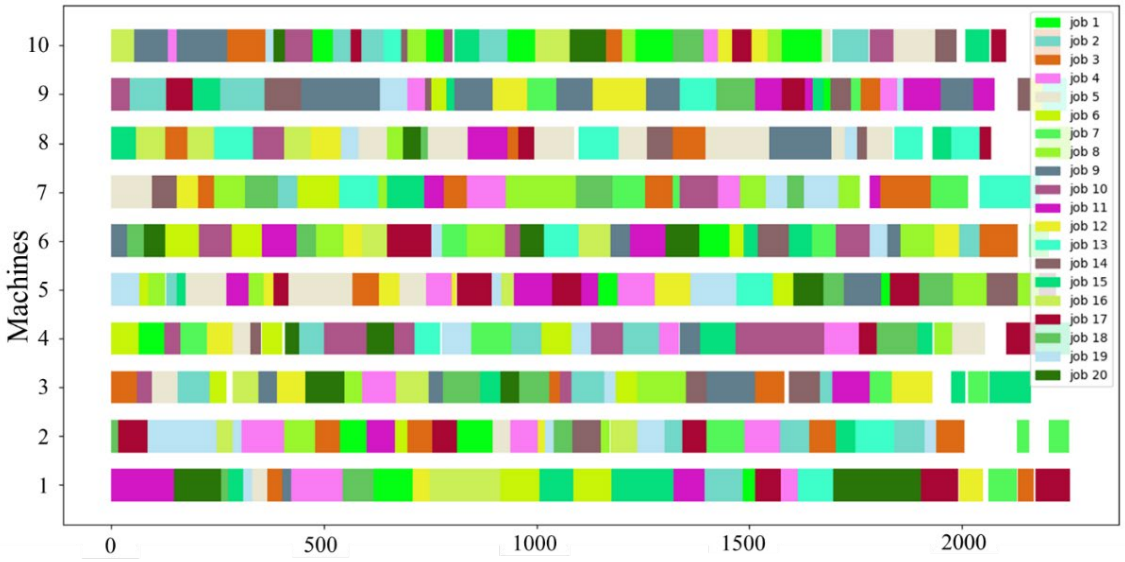

Figure 16: The optimized jobs sequence for Test case (5) (18a) 10 Machine and 20 Jobs

6. Conclusion

This paper proposes a solution for JSSP in FMS. The possible solution was divided into two approaches: the genetic algorithm and the random search algorithm. The authors designed and implemented two optimization procedures and validated them against some the most well-known and accepted benchmarks in JSSP literature. The software was designed to be flexible and able to be extended and/or combined with other optimization approaches or other mathematical formulations for solving the problem. The developed software algorithm was designed to optimize the architecture in order to minimize execution time and enable the algorithm to integrate with other sub-systems.
The cost function was optimized to minimize production time. The results show that the genetic algorithm is capable of solving such complex problems with an optimal setting of initial population of 30 and maximum number of generations of 1000. However, the genetic algorithm takes a long time to find the optimal result. On other hand, the random search algorithm finds the optimal solution in short time but can get stuck in local optima which is clear from the results. So, it is not recommended to use random search with complex problems with local maxima and minima. However, with a simple problem the random search finds a nearly optimal point in a short time.
When the obtained results were compared with those reported in the literature, the developed genetic algorithm successfully gave better results in some cases. However, even for the results that found worse than the previously published, the differences are marginal, which inspires further improvement of the developed algorithm to achieve better results.

Future work should investigate the design of an ANN scheduler to build Job sequences, and to consider optimization of buffer sizes for performed jobs, this would be expected to have a substantial impact on FMS. Moreover, integration with Industry 4.0 via the Internet of Things should be explored as it would significantly improve the performance of the FMSs and strengthen their capability by incorporating smart assists

Acknowledgements

This work was supported by the National Plan for Science, Technology, and Innovation (MAARIFAH), King Abdulaziz City for Science and Technology, Saudi Arabia, under Award 13-INF1155-02.

References

B. Marzouki, O. B. Driss, K. Ghédira, Solving Distributed and Flexible Job shop Scheduling Problem using a Chemical Reaction Optimization metaheuristic, *Procedia Computer Science,* 126, 2018, pp. 1424-1433,

C. Low, Y. Yip, T. H. Wu.: Modelling and heuristics of FMS scheduling with multiple objectives, *Computers & Operations Research*, 2006, 33(3), pp.674-694

Daniëls, F. M. J., On minimzing the probabilistic makespan for the flexible job shop scheduling problem with stochastic processing times, *Master Thesis, Eindhoven University of Technology Department of Mathematics and Computer Science*, 2013

G.Suresh, S.Sahu. Multiobjective facility layout using simulated annealing, *International Journal of production economics*, 1993, 32(2), pp. 239-254

H. Jun, Y. Kim and H. Suh, Heuristics for a tool provisioning problem in a flexible manufacturing system with an automatic tool transporter, *IEEE Transactions on Robotics and Automation*, 1999 15(3), pp. 488-496.

I. Stančin, A. Jović.: An overview and comparison of free Python libraries for data mining and big data analysis, 2019, *42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, pp. 977-982.

J. A. Cano, P. Cortés, E. A. Campo, Al. A. Correa-Espinal. MULTI-OBJECTIVE GROUPING GENETIC ALGORITHM FOR THE JOINT ORDER BATCHING, BATCH ASSIGNMENT, AND SEQUENCING PROBLEM. *International Journal of Management Science and Engineering Management*, 2021, 0:0, pp. 1-17.

J. Abu Qudeiri, U. Umer, F. Abu Khadra, H. Hussein, A. Al-Ahmari, S. Darwish, M. Abidi. Layout design optimization of dynamic environment flexible manufacturing systems, *Advances in Mechanical Engineering* ,2015, 7(6)

J. Arora, Introduction to Optimum Design, 4th ed. *Academic Press*, 2016, ISBN: 9780128009185

J. Blank, K. Deb, pymoo: Multi-objective Optimization in Python, 2020, *IEEE Access*, 8, pp. 89497-89509

J. Błażewicz, E. Pesch, M. Sterna, The disjunctive graph machine representation of the job shop scheduling problem, *European Journal of Operational Research*, 2000, 127(2), Pages 317-331

J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, J. Węglarz, Scheduling in Flexible Manufacturing Systems. *In:* Handbook on Scheduling. International Handbook on Information Systems. Springer, Berlin, Heidelberg, 2001 pp. 539-582

J. E. Abu Qudeiri, U. Umer, M. H. Abidi, A. M. Al-Ahmari, F, Abu Khadra.: On multistage approach for flexible routing in flexible manufacturing systems, *Advances in Mechanical Engineering*, 2016, 8(7), pp.1-11.

J. H. Saleh, G. Mark, N. C. Jordan.: Flexibility: a multi-disciplinary literature review and a research agenda for designing flexible engineering systems, *Journal of Engineering Design*, 2009, 20(3), pp.307-323

L. Andrew.: Ising formulations of many NP problems, *Frontiers in Physics*, 2014, 2(5)

Miloševi, AN OVERVIEW OF GENETIC ALGORITHMS FOR JOB SHOP SCHEDULING PROBLEMS *2015*

N. H. Tran, H. S. Park, Q. V. Nguyen, T. D. Hoang.: Development of a Smart Cyber-Physical Manufacturing System in the Industry 4.0 Context, *Applied Sciences*, 2019, 9(16), 3325

Nedwed, Frank et al., shopST: Flexible Job-Shop Scheduling with Agent-Based Simulated Trading, *MATES (2017).*

P. Wittek.: Chapter 14- Boosting and Adiabatic Quantum Computing In *Quantum Machine Learning, Academic press,* 2014, ISBN 9780128009536

Parjapati, S. , Jain, A. 'Optimization of Flexible Job Shop Scheduling Problem with Sequence Dependent Setup Times Using Genetic Algorithm Approach'. *World Academy of Science, Engineering and Technology, International Journal of Mechanical and Mechatronics Engineering,* 2015, 9(1), 42 - 47.

Philip, D., Scheduling reentrant flexible job shops with sequence dependent setup times, *Master Thesis, Montana State University (MSU), Industrial and Management Engineering*, 2005

Problem Instances and Computational Study of Flexible Job Shop Problem, *Dalle Molle Institute for Artificial Intelligence USI-SUPSI*, last access November 2020 http://people.idsia.ch/~monaldo/fjsp.html#Problem

PyCharm Integrated Development Environment IDE, last access November 2020, *https://www.jetbrains.com/pycharm/*

S. Das, P. N. Suganthan, Differential Evolution: A Survey of the State-of-the-Art*, IEEE Transactions on Evolutionary Computation*, 2011, 15(1), pp. 4-31

S. Kavitha, P. Venkumar, N. Rajini and P. Pitchipoo, An Efficient Social Spider Optimization for Flexible Job Shop Scheduling Problem, *Journal of Advanced Manufacturing Systems,2018,* 17(02)

S. Peynirci, A. Meral. Capacity and tool allocation problem in flexible manufacturing systems. *Journal of the Operational Research Society*. 2006, 57(6) pp. 670-681.

V. Nguyen, H.P. Bao, An Efficient Solution to the Mixed Shop Scheduling Problem Using a Modified Genetic Algorithm, *Procedia Computer Science, 2016, 95*

Yang X, Zeng Z, Wang R, Sun X Bi-Objective Flexible Job-Shop Scheduling Problem Considering Energy *Consumption under Stochastic Processing Times. PLOS ONE,* 2016, 11(12): e0167427