

Solving problems of the flexible scheduling machines

Fis Geci, Eliot Bytyçi

Department of Mathematics, Faculty of Mathematical and Natural Sciences, University of Prishtina "Hasan Prishtina", Prishtinë, Kosovo

Article Info

Article history:

Received Mar 10, 2023

Revised Oct 25, 2024

Accepted Nov 19, 2024

Keywords:

Coarse grained

Flexible machines

Genetic algorithm

Job scheduling problem

Parallel genetic algorithm

ABSTRACT

Flexible job scheduling problem (JSP) as an optimization problem, tends to find solution for allowing different operations to be processed faster. This problem could be solved by genetic algorithm, as we have proven in another experiment. Now, we have tried to outperform state of the art, by using parallel genetic algorithm. Parallel genetic algorithm has two types and we have chosen the most popular one coarsened grained genetic algorithm, for our specific case. The results have improved time wise and are promising in some of the datasets, while a need exists for improving on other ones. In the future, we will compare both versions of parallel genetic algorithms but also compare the results to another algorithm.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Eliot Bytyçi

Department of Mathematics, Faculty of Mathematical and Natural Sciences

University of Prishtina "Hasan Prishtina"

Nëna Terezë, 10000, Prishtinë, Kosovo

Email: eliot.bytyci@uni-pr.edu

1. INTRODUCTION

A primary concern in manufacturing processes revolves around inefficiencies, notably characterized by various forms of waste encompassing product loss, time inefficiency, energy dissipation, and underutilization of labor expertise. Addressing these challenges has prompted the adoption of mechanized automation as a strategic intervention to improve operational inefficiencies, a step that has opened many doors in production lines [1]. The resolution of one predicament within manufacturing processes has caused a secondary challenge. The deployment of mechanized solutions has introduced a new concern pertaining to the systematic arrangement of tasks within these machines. The imperative is to formulate an organizational framework conducive to optimal efficiency in the execution of tasks [2]. These types of problems and related problems are studied in a field called scheduling problems [3]-[21].

The domain of machine scheduling problems holds considerable breadth, finding application across diverse industries reliant on machinery for streamlining production processes. Given its expansive applicability, this problem garners substantial interest within the community. The extensive scope and pertinence of this predicament have motivated our contribution to the broader community, wherein we propose a novel method aimed at addressing challenges inherent in this class of problems.

The job scheduling problem (JSP) [4] is a variant of the multi-objective optimization problem [22]. Within the paradigm of these problems, a configuration comprises a set of m machines and j jobs, each delineated by a specific number of operations denoted as o . The assignment of operations designates the machine on which they are processed and their respective processing speeds. A viable resolution to this challenge manifests as a legitimate sequence of operations allocated across the machines. The conditions that must be met for a solution to be valid are the following [5]:

- An operation of a job starts processing only if the operation before it has been processed. So, for a job j the operations $o1$ and $o2$ of job j are valid only if the processing time $po1$ of $o1$ and $po2$ of $o2$ satisfy the equation $po1 < po2$,
- The time when an operation completes the processing of a machine must be greater than 0,
- An operation can only be processed on one machine.

The problem we are dealing with is a more relaxed variant of JSP called flexible job shop scheduling problem (FJSP) [17]. The objective of FJSP is to find $min Cmax$ [23]. $Cmax$ represents the processing completion time of the last operation. Whereas $min Cmax$ represents the smallest possible $Cmax$ that can be achieved by ordering the operations. The FJSP and JSP variants have only one difference. At FJSP operations have several options on which machines they can be processed. So, an operation can be processed either on one machine or on another, where each machine has a different processing time. This additional condition allows the schedule to be more flexible, which gives the problem its name flexible.

In the context of the FJSP, a defined set of machines is implicated, where distinct operations are subject to processing at specified rates. Integral to the FJSP is the concept of work, which constitutes a series of operations, constituting the fundamental components of the problem. The primary objective entails orchestrating these operations in a sequential manner to fulfill the specified problem objectives. A representative instance of the FJSP, featuring three jobs and three machines, is presented in Table 1.

Table 1. Instance of FJSP with 3 jobs and 3 machines

Job 1		Job 2		Job 3	
O1,1	O1,2	O2,1		O3,1	O3,2
O1,1		O2,1		O3,1	
M1=3	M2=6	M1=6	M2=4	M3=3	M3=4
O1,2				O3,2	
M3=1	M1=2			M1=2	M3=10
				O3,3	
				M2=1	M3=2

Table 1 depicts a scenario comprising three jobs, each characterized by a distinct set of operations. These operations, assigned to respective machines, are subject to processing at varying speeds. As explained earlier, the overarching objective is to ascertain an optimal sequencing of these operations across machines, minimizing the $Cmax$ (maximum time across all jobs) value. The primary aim is to strategically arrange operations to achieve expeditious completion of the work.

Even though there might a number of solutions, Table 2 illustrates one viable solution for the given instance outlined in Table 1. Within Table 2, the sequential arrangement of operations on the machine adheres to the stipulated conditions. In this particular instance, the $Cmax$ value is 7, signifying the completion time upon processing the final operation, denoted as O3,3. It is noteworthy that alternative solutions exist for the instance represented in Figure 1, each potentially yielding a more optimal outcome by achieving a reduced $Cmax$ value. In light of this, our research endeavors are directed towards the development of an algorithm capable of identifying and attaining the optimal solution for benchmark instances.

Table 2. A possible solution of the instance in Table 1

Machine	Operations					
M1	O1,1			O3,2		
M2		O2,1			O3,3	
M3		O3,1		O1,2		
Time	O1,1=3	O3,1=4	O2,1=5	O3,2 and O1,2=6	O3,3=7	$Cmax=7$

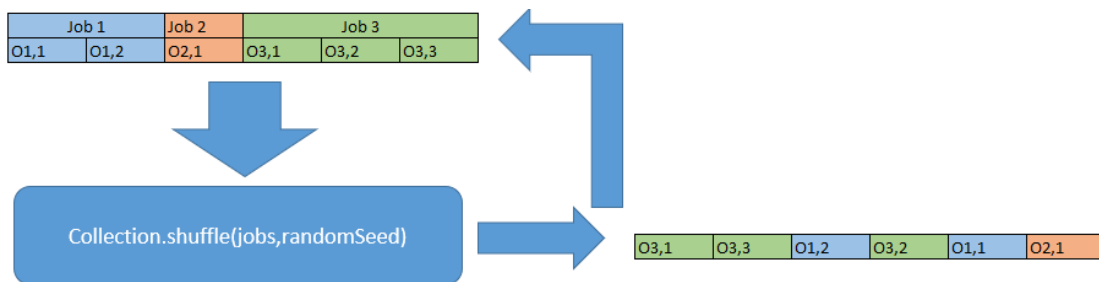


Figure 1. Generating initial population

2. RELATED WORK

As one of the most important problems for multi-objective optimization, FJSP was addressed for the first time in the paper [3]. They created a polynomial algorithm to solve the two-job problem, but this solution was not efficient for problems with more than 2 jobs. Deterministic algorithms are not efficient for solving FJSP with a small number of instances. More efficient algorithms based on heuristic procedures with large number of instances have been proposed using tabu search [6], bee colony [24], ABC algorithm [25], and genetic algorithms [7]–[10].

Burstable [6] developed an algorithm to solve a specific problem. This problem was the ordering of iron pipe manufacturing jobs. Each funnel could be produced at any stage of production but had different production costs at different stages. The organization of how the production of these pipes was done was done manually, but with the development of the algorithm in, it was attempted to automate that process. This automation was necessary because the number of different variations of a pipe production schedule grew exponentially as the number of pipes increased. So, it was no longer practical for this planning to be done manually. The algorithm that was used by Burstable [6] to explore the solution space is tabu search [11].

The paper [6] was only the beginning of the proposed heuristic algorithms for solving the problem. Many different approaches using more sophisticated concepts were proposed over the years. One of these approaches is the paper [7], in which the authors proposed to use a new method for exploring the problem space. This method was an evolutionarily modeled approach. Here a possible solution to the problem would represent individuals in a population. Where after each iteration we produce new solutions by "birthing" new children from the individuals that are in the population, where a better solution is more likely to be a parent. Pezzella *et al.* [7] used this concept and created an algorithm that was comparable to the results of the best approaches through tabu search. The use of the genetic algorithm and the flexibility it offers in the solution opened the way for many other authors to improve and make this approach better.

A very important part of genetic algorithms is the part of generating the initial population. The performance of the algorithm is closely related to the initial population. In relation to this condition, Whitley [8] proposed a new method for generating the initial population and different selection criteria in order to improve the quality of the solutions. Various authors noticed the advantages of algorithms such as tabu search and genetic algorithm, and so they decided to create an algorithm that is the best of both worlds.

This is what the authors of [9] did. They proposed a hybrid between genetic algorithm and local search. They also proposed a method of how solutions can be represented as genes divided into two vectors, where each vector has its own meaning. Local research made it possible to experiment on large numbers of representative problems.

In order for the genetic algorithm to be more efficient and converge to the result faster, Gaham *et al.* [10] proposed that the management of different instances of the genetic algorithm should be done by an algorithm that models the instances independent of each other and you can run the instances in parallel (coarse grained). This separation was necessary because each instance is independent of each other up to a point. Each instance is split across different islands and operates independently until a predetermined migration point is reached. At this point, the best solution is found from all the instances (islands) and they migrate that solution to the other islands. The division of the islands is done in order to create diversity and avoid local maxima, while the migration is done in such a way as to add good solutions to the population. The approach of [10] has greatly accelerated the finding of qualitative solutions of the FJSP.

An attempt to improve the approach of [10] is made in [12]. Here an improvement is proposed to the generation of the initial population, to the processing of a solution to the problem as well as to the selection of an individual for reproduction. Bytyçi *et al.* [12], the structure proposed in [9] was used for dividing the representation of the solution into two vectors. The improvement in the generation of the initial population is made in the use of a method for mixing a collection. This collection in this case represents a sequence of operations in the FJSP.

This work is an extension and improvement of work done by Bytyçi *et al.* [12]. A problem that the first approach in [12] had was that each island, although independent of each other, had to wait for a generation to be completed on one island until the next one continued. This has caused the timing performance to be poor. For this reason, we propose a solution for this in this paper and some new ideas in order to improve the previous approach.

3. DEFINITIONS AND EXPERIMENT SETTINGS

3.1. Definitions

A genetic algorithm is a heuristic search that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where performing individuals are selected for reproduction in order to produce offspring of the next generation [13]. The process of natural selection begins with the selection of the strongest individuals from a population. They produce offspring that inherit

the characteristics of the parents and will be added to the next generation. If the parents have better value, their offspring will be better than the parents and have a better chance of survival. This process continues to repeat and eventually a generation with the strongest individuals will be found [13], [14].

As mentioned above, the algorithm in this paper is a parallel genetic algorithm, since the data of the process could be portioned [18]. There exist two types of parallel genetic algorithms: coarse grained and fine grained [19]. We will use coarse grained since its island model is the most popular. A coarse-grained algorithm. Coarse grained algorithm or algorithm with islands creates several independent instances of genetic algorithms with dart selection. In this algorithm, after n generations, we find the best individual of all the islands. Once we find this individual then we migrate this individual to each other island.

Since our algorithm is coarse-grained, then to code the algorithm we need a structure that enables parallelism emulation. In our solution we used threads to solve this problem. Many programming languages execute code in sequential order. This presents a problem in algorithms where the code must be executed in parallel. If the code is executed sequentially then only one island can be processed once it reaches the migration generation then the next "island" can start processing. The problem with this approach is obvious, the performance is very poor. To solve this problem, many programming languages offer the possibility to declare threads. With threads we can turn each "island" into instances that run in parallel. This is a much more efficient approach to solving the problem. In the next section, the results of the research are explained and at the same time is given the comprehensive discussion. Results can be presented in figures, graphs, tables and others that make the reader understand easily [14], [15].

3.2. Experiment settings

The stages of genetic algorithms in our approach are implemented as follows. The initial population is the first step that must be taken when starting the execution of a genetic algorithm. This step is extremely important because a population with sufficient variation must be generated so that the problem domain is best explored.

As individuals in the population of our implementation are instances of the solution class. What we need to generate is a sequence of operations, i.e., generate an operation vector and assign the machines where the operations will be executed, i.e., generate a machine vector. In order for the generation to be as random as possible, we perform shuffle on the randomly generated seed list. Once this shuffling is done, we start populating the operation vector by randomly selecting from the shuffled list operations one at a time, as seen in Figure 1.

Once a population is generated it must be evaluated. Therefore, an individual must have a certain value that shows the quality of that solution. An order of operations in the operation vector must be placed in a waiting order so that the operations are executed without violating any condition of the problem. The way we approach this problem is through a condition on each operation that tells us whether an operation can be processed at a given time. This condition changes when the previous operation finishes processing, at which point we tell the next operation in that job that it can process. As long as an operation can be processed, it is stored in a structure called a queue.

A key part of the genetic algorithm is selection. An individual that has high fitness (is a good solution) is more likely to be selected for reproduction (according to the law of nature). After evaluating each individual, we assign each individual a selection probability. The way we do this is we take the best fitness in the population and that for us is the numerator.

Once the probability of selection is calculated, the next step begins, the selection of parents for reproduction. This selection algorithm works by choosing a number between [0-1] and then we go randomly selecting from the population and subtract the selection probability from the selected number, if the value becomes negative then we select that individual otherwise we continue until it is filled that condition. After the two parents are selected it is now time to produce a child. The way this reproduction happens is divided into two parts.

First the child's machine vector must be generated and then the child's operation vector must be generated. The child's machine vector creation is easy, where each machine vector gene of the parents is iterated in order, and with a 50/50 probability it can be selected from one of the parents. Generating the child's operation vector is a bit more complex. Initially, a random job is selected. A subset of that job's operations is taken from this job. After that, the operations that are in that generated subset are taken from the operation vector of the first parent and those operations are placed in the operation vector of the child in the same position in the order as they were in the first parent. Then, in the second parent, we remove those operations that are in the generated subset, and in order from left to right, we place the remaining operations in the free positions in the child's operation vector.

Like the crossover, the mutation has two steps. The mutation of machine vector and then of operation vector is done once. With a small probability (2%) mutations can occur in both machine and operation vector. The mutation of machine vector is performed when we select an operation from machine vector and randomly choose one of the other machines that an operation can be processed on. Whereas for operation vector, mutation occurs by selecting two random operations in operation vector and changing the position of those

two operations. As our proposal is to improve the genetic algorithm implementation performed also in [12], then we will run several instances of these implementations with their own populations to achieve better results. We did this by dividing the instances across islands independent of each other where in a certain number of iterations we got the best result up to that moment and migrated it to the other islands.

4. RESULTS

In previous research [12], authors tested the algorithm with several datasets. As an extension to that work, we tested with more data. Data sets that we tested the proposed algorithm are: Bradimarte, Charles and Barnes [15] and Hurink *et al.* [16]. Each of these data sets differs in how flexible the operations are and how many operations must be ordered. The results of the algorithm proposed in this paper are presented.

4.1. Results for Bradimarte dataset

Within Table 3 are presented the outcomes derived from the application of the proposed algorithm to Bradimarte instances. Notably, Bradimarte instances exhibit diminished flexibility and the differentials in outcomes are relatively marginal. Although the proposed algorithm is not specifically tailored for optimization in this particular instance, it manages to yield commendable results, particularly in instances 1, 2, 3, and 4. It is pertinent to highlight that the algorithm's performance is comparatively competitive, as evidenced by its alignment with the upper bound (UB) in these instances, albeit not consistently mirroring the lower bound (LB).

Table 3. Bradimarte results

Name of instance	LB	UB	Average in 50 tests	Best in 50 tests
Mk01.txt	36	39	40.72	40
Mk02.txt	24	26	27.84	27
Mk03.txt	204	204	204	204
Mk04.txt	48	60	65.14	63
Mk05.txt	168	172	173.6	173
Mk06.txt	33	58	79.56	75
Mk07.txt	133	139	144.58	143
Mk08.txt	523	523	523	523
Mk09.txt	299	307	339.12	329
Mk10.txt	165	197	276	265

4.2. Results for Barnes and Hurink datasets

The primary advancement stemming from the newly proposed algorithm is notably discernible in the context of the Hurink and Barnes datasets. Within Table 4 we present the result for Barnes dataset, where we note that unfortunately the algorithm does not improve the results even though in most cases nears the upper bound. Nevertheless, this can be considered as a direct improvement from the algorithm proposed in [12].

Table 4. Barnes results

Name of the instance	LB	UB	Average in 50 tests	Best in 50 tests
mt10c1.txt	655	927	952.06	927
mt10cc.txt	655	910	929.9	911
mt10x.txt	655	918	954.94	927
mt10xx.txt	655	918	952.28	929
mt10xxx.txt	655	918	951.14	931
mt10xy.txt	655	905	936.1	913
mt10xyz.txt	655	847	886.08	858
setb4c9.txt	857	914	967.64	931
setb4cc.txt	857	909	950.58	922
setb4x.txt	846	925	972.54	934
setb4xx.txt	846	930	969.6	942
setb4xxx.txt	846	925	970.76	943
setb4xy.txt	845	924	950.02	930
setb4xyz.txt	838	914	943.9	926
seti5c12.txt	n/a	1185	1233.82	1192
seti5cc.txt	955	1136	1193.74	1160
seti5x.txt	955	1218	1263.16	1221
seti5xx.txt	955	1204	1259.74	1227
seti5xxx.txt	955	1213	1255.8	1226
seti5xy.txt	955	1148	1197.26	1166
seti5xyz.txt	955	1112	1185.28	1153

The Hunrik dataset is composed of three types of data. The first type is *edata* where there is very limited flexibility, somewhere 1-2 machines where an operation can be executed. The second type is *rdata* where flexibility is more prominent where there can be up to 3-5 machines where an operation can be executed, and the last type is *vdata* where flexibility is total where an operation can be executed on any machine. The results for these types of datasets can be seen in Tables 5 to 7. Again, the upper bond is reached in most of the cases and there are some individual cases when the lower bond is reached as well.

Table 5. Hunrik edata

Name of the instance	LB	UB	Average in 50 tests	Best in 50 tests
la01.txt	609	609	609.04	609
la02.txt	655	655	655	655
la03.txt	550	554	565.8	551
la04.txt	568	568	585.18	568
la05.txt	503	503	503	503
la06.txt	833	833	833	833
la07.txt	762	765	778	778
la08.txt	845	845	848.58	845
la09.txt	878	878	878.52	878
la10.txt	866	866	866	866
la11.txt	1087	1106	1106.52	1104
la12.txt	960	960	962.46	960
la13.txt	1053	1053	1053	1053
la14.txt	1123	1123	1126.04	1123
la15.txt	1111	1111	1121.6	1111
la16.txt	892	915	917.36	915
la17.txt	707	757	720.56	707
la18.txt	842	843	851.06	843
la19.txt	796	850	811.94	798
la20.txt	857	919	863.86	857
la21.txt	895	1046	1065.08	1042
la22.txt	832	890	917.34	899
la23.txt	950	953	1006	956
la24.txt	881	918	952.8	925
la25.txt	894	955	978.42	957
la26.txt	1089	1138	1190.66	1160
la27.txt	1181	1215	1249.82	1221
la28.txt	1116	1165	1210.3	1185
la29.txt	1058	0	1212.48	1184
la30.txt	1147	1436	1302	1276

Table 6. Hunrik rdata

Name of the instance	LB	UB	Average in 50 tests	Best in 50 tests
la01.txt	570	574	575.1	572
la02.txt	529	532	534.4	531
la03.txt	477	481	481.34	479
la04.txt	502	504	507.56	505
la05.txt	457	458	460.04	457
la06.txt	799	800	802.72	801
la07.txt	749	801	753.54	752
la08.txt	765	767	767.86	766
la09.txt	853	854	857.92	855
la10.txt	804	805	807.28	806
la11.txt	1071	1073	1079.5	1073
la12.txt	936	936	941.06	938
la13.txt	1038	1038	1043.9	1040
la14.txt	1070	1071	1076.72	1071
la15.txt	1089	1090	1098.26	1094
la16.txt	717	717	741.9	728
la17.txt	646	646	647.22	646
la18.txt	666	669	701.72	691
la19.txt	647	725	732.38	713
la20.txt	756	756	768.84	756
la21.txt	808	846	963.3	925
la22.txt	737	790	880.52	849
la23.txt	816	853	965.44	941
la24.txt	775	825	921.34	902
la25.txt	752	823	902	861
la26.txt	1056	1086	1206.48	1161
la27.txt	1085	1100	1242.3	1213
la28.txt	1075	1097	1230.16	1196
la29.txt	993	1004	1138.28	1110

Table 7. Hunrik vdata

Name of the instance	LB	UB	Average in 50 tests	Best in 50 tests	Name of the instance	LB	UB	Average in 50 tests	Best in 50 tests
la01.txt	570	573	574.78	573	la16.txt	717	717	742.8	720
la02.txt	529	529	532.86	530	la17.txt	646	646	657.94	646
la03.txt	477	482	481.48	477	la18.txt	663	663	714.92	687
la04.txt	502	503	505.38	502	la19.txt	617	617	739.9	704
la05.txt	457	464	461.66	459	la20.txt	756	756	767.28	756
la06.txt	799	802	807.34	803	la21.txt	800	814	1021.4	957
la07.txt	749	751	758.22	752	la22.txt	733	745	929.44	902
la08.txt	765	766	773.6	769	la23.txt	809	818	1020.76	974
la09.txt	853	853	863.6	856	la24.txt	773	796	982.68	927
la10.txt	804	805	811.82	806	la25.txt	751	770	952.58	911
la11.txt	1071	1073	1083.16	1074	la26.txt	1052	1056	1271.62	1211
la12.txt	936	936	946.06	939	la27.txt	1084	1088	1312.1	1272
la13.txt	1038	1038	1048.78	1040	la28.txt	1069	1073	1294.96	1248
la14.txt	1070	1071	1082.98	1070	la29.txt	993	995	1204.92	1143
la15.txt	1089	1089	1103.14	1095	la30.txt	1068	1071	1296.76	1246

5. DISCUSSION

One of the primary objectives of this paper was to propose an efficient algorithm capable of selecting Flexible Job Shop Scheduling Problems (FJSP) within a reasonable timeframe, while achieving results that are comparable to the state of the art. A significant challenge we addressed was the lengthy execution time, which we successfully reduced from one month to one week by implementing a parallel algorithm execution approach. This enhancement not only decreased the time required to obtain results but also improved the quality of those results, achieving lower bounds.

In our experiments, we utilized three datasets that serve as benchmarks for evaluating algorithms targeting FJSP selection. The outcomes from our new approach demonstrated substantial improvements over the algorithm proposed in, which served as our baseline for comparison. Each dataset was designed to assess different facets of the problem.

The Bradimarte dataset specifically evaluates how accurately an algorithm can identify very small improvements. In this scenario, our algorithm underperformed due to the selection method employed. Conversely, the Charles and Chambers and Hunrik datasets tested our algorithm under conditions of high flexibility and significant variation in solutions. Here, our proposed algorithm not only met but exceeded known upper limits in certain instances, consistently performing as well as or better than the previous algorithm.

6. CONCLUSION

In conclusion, our research successfully introduced an efficient algorithm for selecting FJSP that significantly reduces execution time while enhancing result quality. The comparative analysis across three diverse datasets highlighted both strengths and areas for improvement in our approach. Moving forward, we aim to test our algorithm in environments with greater processing power to uncover even better solutions than those achieved on local devices. Additionally, exploring alternative versions, such as fine-grained genetic algorithms, will be a key focus for future research endeavors. This work lays a solid foundation for continued advancements in FJSP selection methodologies.




REFERENCES

- [1] Industrial Automation Services | NRTC Automation, "problem-with-your-manufacturing-line-and-how-you-can-fix-it," [Online]. <https://nrtcautomation.com/blog/the-biggest->. (Accessed: Jan. 01, 2023).
- [2] K. Shigenori, "Scheduling of lot production lines in the Toyota production system," *Journal of Japan Industrial Management Association*, vol. 60, no. 5, pp. 259–269, 2009.
- [3] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.
- [4] R. L. Graham, "Bounds for Certain Multiprocessing Anomalies," *Bell System Technical Journal*, vol. 45, no. 9, pp. 1563–1581, 1966, doi: 10.1002/j.1538-7305.1966.tb01709.x.
- [5] K. Ploydanai and A. Mungwattana, "Algorithm for Solving Job Shop Scheduling Problem Based on machine availability constraint," *International Journal on Computer Science and Engineering (IJCSE)*, vol. 2, no. 5, pp. 1919–1925, 2010.
- [6] R. M. Burstall, "A Heuristic Method for a Job-Scheduling Problem," *Journal of the Operational Research Society*, vol. 17, no. 3, pp. 291–304, 1966, doi: 10.1057/jors.1966.56.
- [7] F. Pezzella, M. Gianluca, and C. Giampiero, "A genetic algorithm for the flexible job-shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 10, pp. 3202–3212, 2008.
- [8] D Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.
- [9] J. Gao, M. Gen, L. Sun, and X. Zhao, "A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop




- scheduling problems,” *Computers and Industrial Engineering*, vol. 53, no. 1, pp. 149–162, 2007, doi: 10.1016/j.cie.2007.04.010.
- [10] M. Gaham, B. Bouzouia, and N. Achour, “An Agent-Based Multi-Population Genetic Algorithm for the Flexible Job Shop Scheduling Problem,” in *8ème conférence internationale conception & production intégrées*, CPL, 2013, pp. 1–5.
- [11] F. Glover, “Future Paths for Integer Programming and Links to Artificial Intelligence,” *Computers and Operations Research*, vol. 13, no. 5, pp. 533–549, 1986.
- [12] E. Bytyçi, A. Berisha and F. Geci, “Coarse-Grained Genetic Algorithm for Flexible Job Scheduling Problem” *SoCPaR 2020. Advances in Intelligent Systems and Computing*, vol 1383, p. 1383, 2021.
- [13] J. Holland, *Addaptation in Natural and Artificial Systems*. *University of Michigan Press*, Ann Arbor, 1975.
- [14] V. Mallawaarachchi, “Introduction to Genetic Algorithms - Including Example Code.” [Online]. Available: <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>. (Accessed: Feb. 05, 2023).
- [15] J. Barnes and J. Chambers, “Graduate Program in Operations and Industrial Engineering,” *The University of Texas at Austin, Technical Report Series, vol. Flexible job shop scheduling by tabu search*, 1996.
- [16] J. Hurink, B. Jurisch, and M. Thole, “Tabu search for the job-shop scheduling problem with multi-purpose machines,” *Operations Research Spektrum*, vol. 15, no. 4, pp. 205–215, 1994, doi: 10.1007/BF01719451.
- [17] P. Brucker and R. Schlie, “Job-shop scheduling with multi-purpose machines,” *Computing*, vol. 45, no. 4, pp. 369–375, 1990, doi: 10.1007/BF02238804.
- [18] Y. Y. Liu and S. Wang, “A scalable parallel genetic algorithm for the Generalized Assignment Problem,” *Parallel Computing*, vol. 46, pp. 98–119, 2015, doi: 10.1016/j.parco.2014.04.008.
- [19] I. Sugiarto and S. Furber, “Fine-grained or coarse-grained? Strategies for implementing parallel genetic algorithms in a programmable neuromorphic platform,” *Telkommika (Telecommunication Computing Electronics and Control)*, vol. 19, no. 1, pp. 182–191, 2021, doi: 10.12928/TELKOMNIKA.V19I1.15026.
- [20] U. Kohlmorgen, H. Schmeck, and K. Haase, “Experiences with fine-grained parallel genetic algorithms,” *Annals of Operations Research*, vol. 90, pp. 203–219, 1999, doi: 10.1023/a:1018912715283.
- [21] K. Gao, Y. Huang, A. Sadollah, and L. Wang, “A review of energy-efficient scheduling in intelligent production systems,” *Complex and Intelligent Systems*, vol. 6, no. 2, pp. 237–249, 2020, doi: 10.1007/s40747-019-00122-6.
- [22] J. L. J. Pereira, G. A. Oliver, M. B. Francisco, S. S. Cunha, and G. F. Gomes, “A Review of Multi-objective Optimization: Methods and Algorithms in Mechanical Engineering Problems,” *Archives of Computational Methods in Engineering*, vol. 29, no. 4, pp. 2285–2308, 2022, doi: 10.1007/s11831-021-09663-x.
- [23] İ. Sariçiçek, “Multi-Objective Scheduling By Maximizing Machine Preferences for Unrelated Parallel Machines,” *Sigma Journal of Engineering and Natural Sciences*, vol. 38, no. 1, pp. 405–420, 2020.
- [24] X. Long, J. Zhang, K. Zhou, and T. Jin, “Dynamic Self-Learning Artificial Bee Colony Optimization Algorithm for Flexible Job-Shop Scheduling Problem with Job Insertion,” *Processes*, vol. 10, no. 3, pp. 1–22, 2022, doi: 10.3390/pr10030571.
- [25] J. Sassi, I. Alaya, P. Borne, and M. Tagina, “A decomposition-based artificial bee colony algorithm for the multi-objective flexible jobshop scheduling problem,” *Engineering Optimization*, vol. 54, no. 3, pp. 524–538, 2022, doi: 10.1080/0305215X.2021.1884243.

BIOGRAPHIES OF AUTHORS



Fis Geci    received the Engineer degree in Department of Mathematics in the University of Prishtina “Hasan Prishtina” in 2022. He currently works as Software Engineer in one of the most renowned companies in Kosovo, Celonis. He can be contacted at email: fis.geci@gmail.com.



Eliot Bytyçi    is an Assistant Professor at the Department of Mathematics, University of Prishtina “Hasan Prishtina”. He is part of several European projects and leads a small group on artificial intelligence in Department of Mathematics. He can be contacted at email: eliot.bytyci@uni-pr.edu.