Solving FJSSP With A Genetic Algorithm

California Polytechnic University, San Luis Obispo, CA

Fall Quarter 2022, Winter Quarter 2023

March 2023

Author: Michael John Srouji

Advisors: Ryan Matteson, Zachary Peterson

Table of Contents

Abstract

The Flexible Job Shop Scheduling Problem is an NP-Hard combinatorial problem. This paper aims to find a solution to this problem using genetic algorithms, and discuss the effectiveness of this. Initially, I did exploratory work on whether neural networks would be effective or not, and found a lot of trade offs between using neural networks and chromosome sequencing. In the end, I decided to use chromosome sequencing over neural networks, due to the scope of my problem being on a small scale rather than on a large scale.

Therefore, the genetic algorithm was implemented using chromosome sequencing. My chromosomes were represented as binary strings with reserved bits for the machine and job numbers. This allowed me to experiment with different mutations such as random bit flip mutation and machine job swap mutations.

The biggest benefit of genetic algorithms over heuristic algorithms is the potential for improvement. While greedy gives good results initially, genetic beats out greedy quickly after a small number of epochs. Furthermore, I suspect that genetic algorithms should be much faster than other learning algorithms, but as this is an under-documented metric, I decided to contribute my own results to help document this metric.

For future work, it would be interesting to see how a neural network model would have reacted, and how its time to find a solution would compare to chromosome sequencing. Another interesting topic is a scheduler that can adapt to any variation of the Job Shop Scheduling Problem, as this would be very useful in the real world. One final interesting topic would be to implement some kind of dynamic job loading for this genetic algorithm, as in real world situations, new jobs and tasks get scheduled all the time. But, this is a very complicated problem, thus it is best left to the future.

Introduction

Historically, there have been a large array of NP-Complete problems, with difficult solutions to them. While in the past these problems have been solved using heuristic methods, recently, with the advent of machine learning, these problems have found new solutions. For example, there is the traveling salesman problem, which has been solved using a genetic algorithm recently [1].

This paper then focuses on genetic algorithms, and seeks to find how such an algorithm compares to other algorithms when solving NP-Complete problems. An important consideration that was made is that genetic algorithms have been theorized to be effective at solving combinatorial problems [3]. For example, the traveling salesman problem is a combinatorial problem, which was likely a large factor in why genetic algorithms were so effective at solving it.

Therefore, this paper will focus on solving the Job-Shop Scheduling Problem using a genetic algorithm. It will also discuss whether using a neural network model or chromosome sequencing is a better method, and the benefits and shortcomings of both methods. Finally, this paper will compare the make span of chromosome sequencing as compared to well known heuristic methods, and as compared to the theoretically best make span for the benchmarks used.

What is JSSP

The “Job Shop Scheduling Problem” is an NP-hard problem that has been around for decades, and is a very good example of a combinatorial problem. The problem is thought to have originated during the early 1950s, illustrating the issue of manufacturing optimization during this time period [7]. Traditionally, this problem has been solved using heuristic methods, and only recently has there been some research regarding using machine learning algorithms to solve it.

There are many variations of the Job Shop Scheduling Problem, but the base problem has these constraints:

1. There are n number of jobs, each with a set of operations.
2. The operations in each job must be completed in a specific order, require a specific machine, and must run to completion once started.
3. There is a limited amount of resources, most often these resources are denoted as m number of machines.
4. Machines can only process one operation at a time.
5. The goal is to minimize the amount of time it takes to complete all n jobs.

While this is the base problem, many variations exist, such as "Flow Shop Scheduling," and "Flexible Job Shop Scheduling." This paper will focus on one such variation, the "Flexible Job Shop Scheduling Problem." In this variation, the only difference is that operations can run on any machine, rather than being constricted to a single machine. This is a more general sense of the problem, as it abstracts away from the realm of manufacturing, and is more applicable to the realm of computing.

In 1976, in a research paper named "The Complexity of Flow Shop and Job-Shop Scheduling," Garey proved that the job shop scheduling problem is NP-Complete when there are greater than two machines available. Thus, he proved that no optimal solution for this problem could be found in deterministic polynomial compute time when there are 3 machines or greater. Therefore this paper will focus on instances of this problem that have greater than 3 machines.

The reason this problem is NP-Hard, is that there are n! possible solutions for any given instance of this problem, where n is equal to the total number of operations. For example, if there are 2 jobs, each with 2 operations, than in all there are 4! = 24 possible solutions to the problem. Of course, for most applications, there will be many more operations than just 4, at which point it would not be feasible to find the most optimal solution by hand. This is why algorithms for this problem have been researched since the 1950s.

Benchmarking Instances

Eventually, benchmarks were created for JSSP so that solutions could be compared to a baseline. One said benchmark were the Taillard Instances, created by Eric Taillard and documented in his paper "Benchmarks for Basic Scheduling Problems." [8]. In this paper, Taillard illustrates the permutation flow shop, job shop, and open shop scheduling problems, as well as benchmarks for each of them. Although not all of the benchmarks had documented optimum make spans, the Taillard instances were a useful starting point for testing the validity of my solution.

But the Taillard instances did not have very computationally difficult problems, with the most complex instance being 500 jobs and 20 machines. Furthermore, Taillard's instances are not statistically easy to use, as they are a very old benchmark. Therefore, I opted to use benchmarks created by Eva Vallada, Ruben Ruiz, and Jose Framinin outlined in their paper "New hard benchmark for flow-shop scheduling problems minimizing make span" [9]. These benchmarks have instances that are much more computationally difficult, and have more ease of use.

My goal is to achieve a good (if not the optimum) makespan, with a focus as well on time spent achieving this result. While genetic algorithms do not often achieve the most optimum results, they are able to achieve good results faster than most other algorithms, and are very competitive overall. Another goal then, is to explore the advantages and disadvantages of genetic algorithms, and contribute another benchmarking data point for others to test against in the future.

Implementation

My implementation was split between the genetic algorithm, the heuristic algorithms, and the loader (which I will call simulator here). My simulator had the functionality of loading the chosen instance file, fetching all jobs and tasks, and configuring the machine state based on this instance. It would then simulate this machine, and return metrics such as make span

and utilization, which my genetic algorithm uses during mutation and evolution.

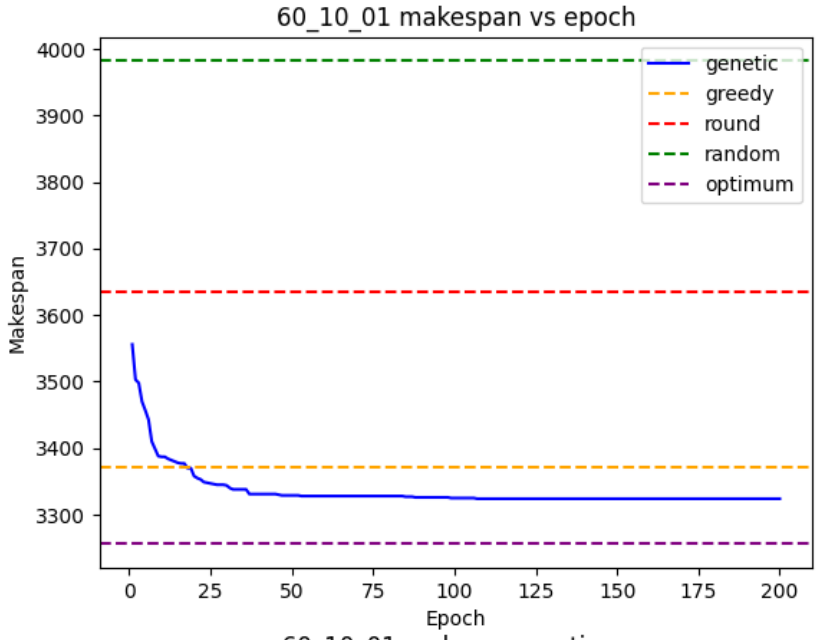
My genetic algorithm initially used neural networks in order to train a model, but this had many shortcomings. The biggest issue was with test and train data, as each benchmark instance is its own isolated problem, and there were not correlating variables between the instances. The result was that neural networks did not train or infer well, and achieved suboptimal make span results, as it did not have good training in place. Given this, I decided to pivot to using chromosome sequencing in my genetic algorithm, rather than a neural network.

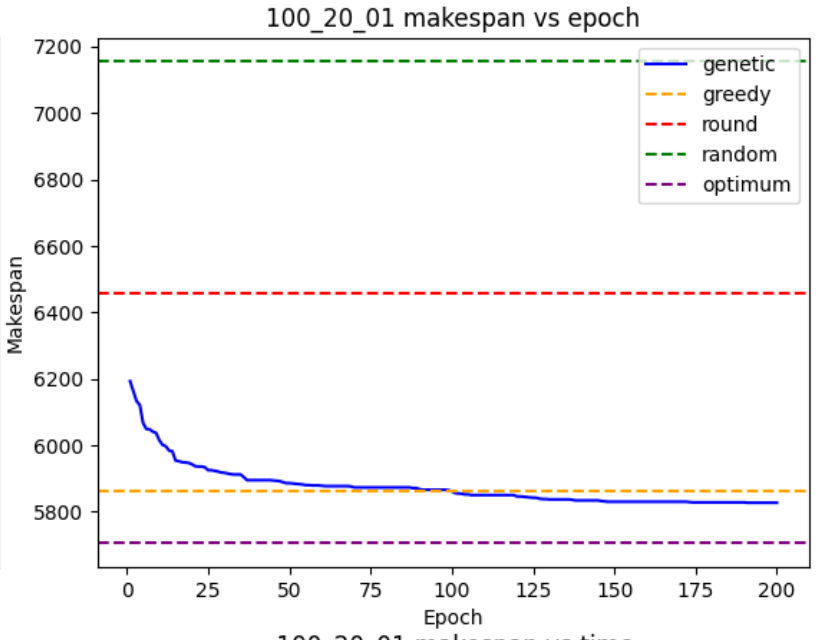
Chromosome sequencing ended up being a much more optimal solution for this type of problem, as it did not have this same issue with training data. To explain how it works, the genetic algorithm first fetches an initial chromosome from the simulator, then mutates it multiple times, then verifies the new chromosomes through the simulator for every epoch. Said chromosomes are binary strings of 0's and 1's, with a reserved number of bits for the machine number and a reserved number for the job number. Note that the binary does not need to contain task number, since only the first available task in any job is valid to be scheduled. My simulator has functions to verify that a chromosome/sequence is valid, to verify that the mutation does not violate the rules of FJSSP.

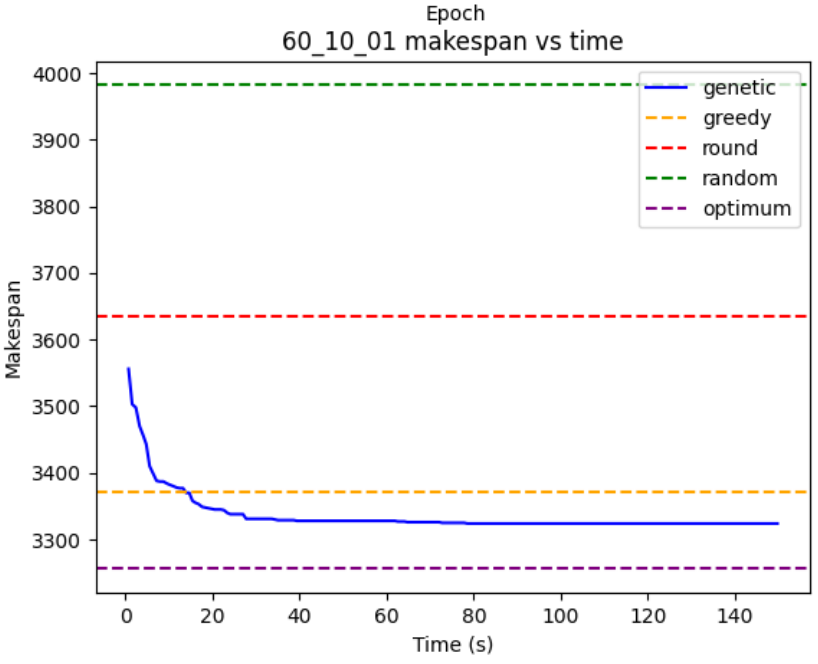
There is also the method of mutation. While random bit flip mutation would have worked, I chose to implement a machine job flip mutation, where I swapped the machine and job bits between separate sequences in the chromosome. The reason that I chose to use this type of mutation over random bit flip mutation, is because with random bit flip mutation, many of the chromosomes generated would be invalid. This causes a lot of overhead time with the algorithm attempting to generate valid chromosomes. On the other hand, simply swapping the machine and job bits (or only changing the machine bits) between different sequences ensures that the resulting chromosome is almost always valid. This is because there should always be the same number of jobs for any generated chromosome per instance.

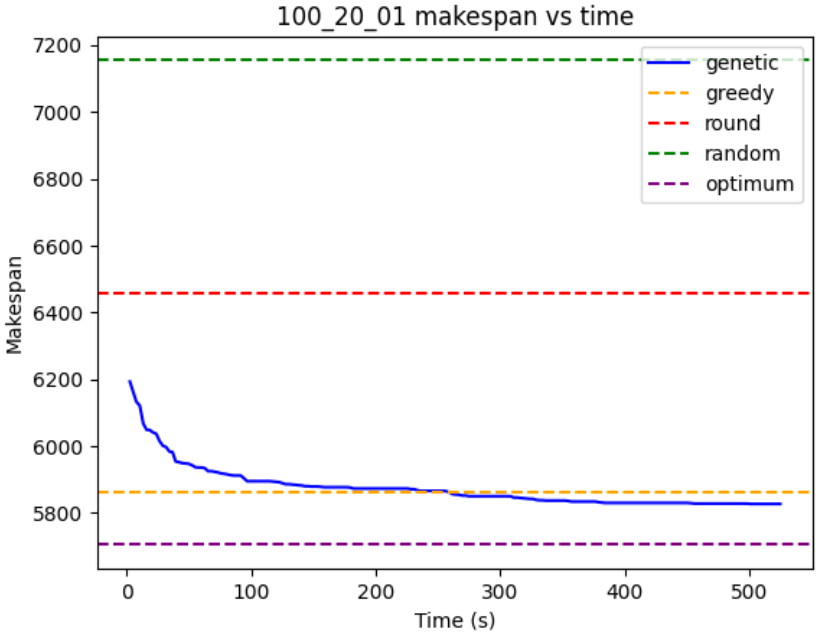
The Benefits of Genetic Algorithms for FJSSP

When compared to heuristic algorithms, a big benefit of genetic algorithms is the potential of improvement, whereas heuristic methods will always yield the same make span. We can see that over more epochs, the make span of the genetic algorithms' solution will continually decrease. Furthermore, the genetic algorithm will almost always find a better solution than a heuristic algorithm, if given enough time. An important consideration to be made then, is how long it takes to find a better solution.









At first, the genetic algorithm returns worse solutions than greedy (and at around the same level as round robin), but it improves very quickly. The genetic algorithm will beat greedy somewhat quickly, although how quickly depends on the size of the instance. Interestingly enough, with bigger instances, it takes more epochs for the genetic algorithm to beat greedy. For example, for an instance with 60 jobs with 10 tasks each, genetic beats greedy after about 20 epochs. But, for an instance with 100 jobs with 20 tasks each, it takes genetic 100 epochs before it beats greedy. Note that about 80% of the improvement for these instances was achieved within the first 40 epochs (20% of runtime) or so.

These results are also interesting in the context of the optimal stopping problem. As the graphs show, the brunt of makespan improvement happens in the first 20% of runtime. After that, the rate of improvement becomes drastically slower. In that case, it might actually be best to stop after the 20% mark to minimize resources/time spent. In real world contexts this could lead to a large amount of time/resources saved (about 80%), for very similar (or good enough) results. Furthermore, the optimal stopping rule being set at about 20% can be observed for both the smaller and larger instances. The difference between the smaller and larger instances, is the rate of improvement after this 20% mark. For the larger instances, we can observe that the rate of decline in improvement is slower than for the small instances. This is likely because larger instances are more complex, and there are many more permutations of job/task/machine combinations in order to find the most optimal result.

The reason heuristic algorithms are faster than genetic algorithms, is because heuristic algorithms return their solution after a single epoch, whereas genetic algorithm will continue to do work to find better solutions. Then, it would be more accurate to compare genetic algorithms to other types of learning algorithms. Sadly, almost no other papers regarding this subject and specific problem make note of the runtime or train time for their algorithm, which makes such a comparison difficult to make. Therefore, this paper will contribute its recorded runtimes for Vallada’s benchmarks, since it is an under documented metric. Note that this implementation does not use parallelization, and that a population size of 100 was used.

Instance	Avg Time Per Epoch (Seconds)	Makespan After 100 Epochs
VFR10_5_1	0.083	684
VFR10_10_1	0.15	956
VFR10_15_1	0.215	939
VFR10_20_1	0.272	1342
VFR20_10_1	0.286	1330
VFR20_20_1	0.572	1980
VFR30_10_1	0.427	1768
VFR30_20_1	0.829	2184
VFR40_10_1	0.539	2471
VFR40_20_1	1.085	2843
VFR50_10_1	0.679	3053
VFR50_20_1	1.406	3320
VFR60_10_1	0.747	3325
VFR60_20_1	1.5	3773
VFR100_20_1	2.574	5861
VFR100_60_1	9.468	7701
VFR200_20_1	5.504	11157
VFR200_60_1	21.607	13755
VFR400_20_1	13.916	21274
VFR400_60_1	66.446	23923
VFR600_20_1	22.287	31924
VFR600_60_1	125.054	34425
VFR800_20_1	36.836	41987
VFR800_60_1	189.636	44530

Interestingly enough, the number of tasks per job being increased had a proportionally larger effect on the time per epoch as compared to

increasing the total number of jobs. One reason for this may be that increasing the number of tasks per job greatly increases the complexity of the problem, since later tasks and constraints have a cascading effect on the previous tasks and the optimal order in which to schedule. I believe it would be interesting to observe how other learning algorithms deal with this increase in complexity, as compared to my genetic algorithm.

The Problems of Genetic Algorithms for FJSSP

Compared to heuristic algorithms, my genetic algorithm takes much longer to find a good solution. Of course, this is because it continually improves over some number of epochs whereas heuristic methods return their final solution after the first epoch. Regardless, this behavior means that a genetic algorithm such as this one is not a good fit for many situations. For example, in situations where a very quick result is needed, genetic algorithms are not the right choice. Although, I suspect that genetic algorithms are still likely to be faster than other learning algorithms for this problem.

Furthermore, I do not believe a genetic algorithm such as mine would be effective on a large scale. This was one of the trade offs of neural networking over chromosome sequencing. With neural networks, most of the time is spent on training, and after training is complete, a neural network will be able to infer the solution in very little time. Therefore, a lot more initial time is spent on training the solution than inferring the solution, and if the neural network is used on similar data in the future, it will infer the solution very quickly. This is in contrast to my approach, chromosome sequencing, where the algorithm instead will begin immediately inferring the solution, and spends no prep time on training. The effects of this are that neural networks end up being better on a large scale (faster results later but not now), whereas chromosome sequencing is better on a small scale (fast results now but not later).

There is also the issue of consistency. Some aspects of my genetic algorithm are random, leading to variation that may sometimes be

unwanted. While this enables my algorithm to improve, it also means that it is not the right fit for many real world uses. Although, this does not just apply to genetic algorithms, but to all learning algorithms.

Future Work

I believe that it is definitely possible to implement a genetic algorithm such as mine to work with neural networks, but a lot of work will need to be dedicated to the training and testing process. I only did exploratory work on using a neural network for my genetic algorithm, and while it was not effective in the context of my research (since my scope was small), neural networks would be very effective on a large scale and for the real world.

One of the issues with my implementation is that it does not draw correlations between different sets of data and improve between test sets. In essence, my implementation works on a much smaller scale. Therefore, if a proper neural network implementation was created, with large data sets that had some types of correlations, then I believe that such an algorithm would work even more efficiently and on a much larger scale.

Another interesting topic for future work would be a dynamic genetic scheduler, which would allow the user to add jobs and tasks to the current dataset. Due to the constraint of the problem, this would be a very difficult task to accomplish, but I believe that it would lead to interesting results. In the real world, there is often the need for change.

There are many examples of large scale schedulers in the real world, where new jobs get queued onto the system constantly. Therefore, I believe it would be very interesting to see how a genetic algorithm would react to such a problem, and adapt to the newly added jobs and tasks. Whereas heuristic algorithms are used due to this issue of a dynamic workplace with constantly shifting workloads, a learning algorithm may offer some insight on how to further improve the scheduling systems currently used in the real world, if this hurdle was ever overcome.

One final interesting topic to explore for the future would be to adapt this genetic algorithm to work with any variation of the JSSP problem. While this implementation tackles the FJSSP variant, there are many more variants of this problem, such as the permutation job shop, flow shop, and more. Just like in the real world, there are often new or different constraints than the ones outlined in the original job shop scheduling problem, thus a scheduler with the ability to adapt to any number of constraints would be valuable. Work has already been done in this field, but it is not a widely explored topic, even though a reconfigurable scheduler is often what is demanded in the real world [2].

Conclusion

Solving the FJSSP problem with genetic algorithms was an interesting challenge, and led me to explore many different aspects of learning algorithms. There are a lot of tradeoffs between using either a neural network or chromosome sequencing for the genetic algorithm, which bubble down to the scope of scale for the issue. The conclusion I drew regarding this was that chromosome sequencing should be used for small scopes, and neural networks should be used for large scopes.

What I found was that as compared to the greedy heuristic algorithm, genetic would have a worse make span initially, but beat greedy after some number of epochs depending on the size of the instance. For example, for a 60x10 instance, genetic beats greedy after 20 epochs. But for a 100x20 instance, genetic beats greedy after 100 epochs. This behavior continues to scale for larger and larger instances.

I suspect that genetic algorithms should find solutions to this problem faster than other learning algorithms, but this is not a well explored metric. Therefore, I decided to contribute my own datapoints for this metric, in the case that others in the future find use and expand this metric further. The rate of improvement of learning algorithms, and the time they take to find good solutions, are both very important metrics to explore, as there are

many situations in the real world where a better solution is needed, but time is still an important factor.

The genetic algorithm I presented does not work well on a large scale, and is not consistent, but it successfully shows one method of how a genetic algorithm might be used to solve the FJSSP problem. Furthermore, I believe that in the future, it would be interesting to explore topics such as allowing jobs and tasks to be dynamically added to the system.

References

[1] Kulenović, F., & Hošić, A. (n.d.). *Application of genetic algorithm in solving the travelling salesman problem*. ResearchGate. Retrieved November 29, 2022, from https://www.researchgate.net/publication/356946515_Application_of_genetic_algorithm_in_solving_the_travelling_Salesman_problem

[2] Montana, D., Hussain, T., & Vidaver, G. (1970, January 1). *A genetic-algorithm-based reconfigurable scheduler*. SpringerLink. Retrieved November 14, 2022, from https://link.springer.com/chapter/10.1007/978-3-540-48584-1_21

[3] Noor, S., Nawaz, M. S., & Lali, M. I. (2015, August). *Solving job shop scheduling problem with genetic algorithm - researchgate*. ResearchGate. Retrieved November 14, 2022, from https://www.researchgate.net/publication/281545095_SOLVING_JOB_SHOP_SCHEDULING_PROBLEM_WITH_GENETIC_ALGORITHM

[4] Shylo, O. V., & Shams, H. (2018, August 31). Boosting binary optimization via binary classification: A case study of job shop scheduling. arXiv.org. Retrieved November 14, 2022, from https://arxiv.org/abs/1808.10813

[5] Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., & Clune, J. (2018, April 20). Deep Neuroevolution: Genetic Algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv.org. Retrieved November 14, 2022, from https://arxiv.org/abs/1712.06567

[6] Weckman, G. R., Ganduri, C. V., & Koonce, D. A. (2008, January 20). *A neural network job-shop scheduler - Journal of Intelligent Manufacturing*. SpringerLink. Retrieved November 14, 2022, from https://link.springer.com/article/10.1007/s10845-008-0073-9

[7] Cozzolino, C., Christiansen, B., Vallejos, E., Sorce, M., & Visk, J. (n.d.). *Job shop scheduling*. Job shop scheduling - Cornell University Computational Optimization Open Textbook - Optimization Wiki. Retrieved December 5, 2022, from https://optimization.cbe.cornell.edu/index.php?title=Job_shop_scheduling

[8] Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, *64*(2), 278–285. https://doi.org/10.1016/0377-2217(93)90182-m

[9] Vallada, E., Ruiz, R., & Framinan, J. M. (2014, August 12). *New hard benchmark for flowshop scheduling problems minimising makespan*. European Journal of Operational Research. Retrieved February 6, 2023, from https://www.sciencedirect.com/science/article/abs/pii/S0377221714005992?via%3Dihub