PERINATAL JOURNAL

Original Article

Perinatal Journal 2025; 33(2):135-141

https://doi.org/10.57239/prn.25.03320016

Optimizing university exam timetables: A comparative analysis of shuffling algorithms

Ahmad Barghash[1]*

[1]Department of Computer Science, German Jordanian University, Amman, Jordan

Abstract

Modern educational systems are transferring to digital forms in most of their features. For instance, digitalizing exam scheduling has been a complicated task in universities for decades with no clear solutions. In this study, we explore the application of shuffling techniques to generate acceptable schedules following standard university constraints and we present an advanced comparative study between widely used shuffling algorithms. Moreover, we present a judgment scheme to justify the addition or removal of extra examination days. We studied the performance of the Fisher-Yates, Sattolo, BogoSort, MergeShuffle, and Riffle shuffling algorithms on three real registration datasets from the German Jordanian University. The datasets were fully encoded and highly diverse in terms of student counts, courses, multi-sections availability, and registration cases. We report that shuffling significantly enhances the exam scheduling process and increases the chances of finding acceptable schedules. To obtain stable results, we applied each shuffling algorithm up to 100k times for each registration dataset and then analyzed the number of acceptable schedules generated along with the time required to generate them and test their compliance with university examination constraints. Although most shuffling algorithms generate acceptable schedules, MergeShuffle exhibits a distinguished performance under different aspects of diverse datasets.

Keywords: Exam, Scheduling, University, Quality education, Registration, Shuffling, Fisher-Yates, Sattolo, Bogo sort, Merge-Shuffle, and Riffle

1.Introduction

Providing digital services to staff and students is an important objective in universities strategic plans. Exam Scheduling is one of the main features in universities educational systems. It is a complicated task given that the constraints the universities have and their limited scheduling time. Many universities use a relaxed examination mode where exam dates are defined based on a discussion between the instructor and the students. In this scenario, a student decides about the ability of having multiple exams a day or requesting a makeup exam. Other universities define obligatory examination constraints and have acceptable schedules announced to instructors and students early in the semester once the course registration cases are completed. In this study, we discuss creating schedules considering two widely-adapted constraints which are: 1. The schedule should have a maximum of two exams per day for each student 2. Daily exams should not conflict in time for any student. As spacing is required in examinations, room capacity should be considered as a third constraint. Manual preparation of a schedule is a tedious task, and an acceptable schedule is almost impossible. Therefore, universities target digitalized solutions that generate mid-term and final schedules considering mandatory constraints.

Many projects have targeted exam scheduling and have introduced new techniques to automate the process. Recently, a module with two genetic algorithms was introduced to solve the exam scheduling problem. The solution incorporates two techniques: the first is a pure genetic algorithm, and the other combines graph algorithms with variations of the genetic algorithm, and reports substantial improvements over traditional methods [1]. Modules using Memtic Algorithms have been proposed as quality improvement techniques for schedule generation [2]. Moreover, schedule quality was effectively considered in the adaptation of Greedy-Least Saturation Degree (G-LSD) heuristic to evaluate the generated schedules [3]. Generally, Heuristic methods have been considered in exam scheduling problems, where other researchers preferred Hyper-heuristics to metaheuristics and reported noteworthy achievements [4]. However, schedule quality can be enhanced if operational costs are minimized. Therefore, profiling-based algorithms begin by grouping course exams based on their requirements before scheduling, thereby ensuring minimum costs at the scheduling time [5]. Another factor to consider is the flexible versus hard-scheduling constraint. Patrovic et al. used fuzzy

Correspondence: Ahmad Barghash, Department of Computer Science, German Jordanian University, Amman, Jordan; **e-mail:** ahmad.barghash@gju.edu.jo ,
Received: August 30, 2025 **Accepted:** October 23, 2025

constraints to provide a satisfaction measure for the generated schedule and evaluated the tradeoff between soft and hard constraints [6]. However, unnecessary flexibility in soft constraints or ignoring registration data may jeopardize the schedule quality. Therefore, modern approaches have studied the ability to generate schedules with solid base knowledge on the curricula without significant loss in solution quality and have reported a significantly reduced need for rescheduling [7].

The computational time required to complete a schedule is widely reported. Therefore, parallel computation was used in the examination process to reduce schedule generation time[8]. However, heuristic ordering-based methods have reported notable achievements in exam scheduling, even with limited computing resources and reduced time costs[9]. Moreover, the required time can be reduced if the required processing iterations are minimized. Therefore, a multistage schedule generation theme was introduced, where the scheduling process starts with a partial assignment of scheduled courses followed by multiple improvement stages until the final desired schedule is achieved [10,22].

Nonetheless, simple solutions should still be valid for generating a schedule that considers all mandatory constraints. The distribution of courses among daily sessions and the planned schedule period should be considered using simple randomization schemes. We have applied basic shuffling techniques over the past 10 years and achieved exceptional results. However, different shuffling algorithms result in different computational times and numbers of iterations.

1.1 Objective of the study

The aim of this work is to analyze the efficacy of different shuffling algorithms in terms of time and iteration counts when used to generate conflict-free university exam schedules. Moreover, we publish real but encoded registration data for a medium-size university, which can be used as a model for future research in this area.

1.2 Organization

This article is organized into the following sections : Section 1 I introduce the issues universities face when generating conflict-free exam schedules along with the current methods used in the generation process. Section 2 presents the encoded real registration datasets we acquired from the German Jordanian University along with the shuffling methods we are comparing in the process of schedule generation. Section 3 presents the results of applying the shuffling algorithms to generate conflict-free schedules using real registration data along with a detailed comparison of the achievements of the chosen shuffling algorithms. Section 4 concludes his work and suggests future directions.

2. Materials and Methods

In this section, we present the datasets, applied algorithms, and testing methods used to compare the different algorithms while generating the acceptable schedules.

2.1 Data sets

In this work, we present real datasets encoded for privacy constraints from real registration data of bachelor's degree students at German Jordanian University in three consecutive semesters, as explained in Table 1. Fully encoded datasets are found in the supplementary files (First, Second, Summer).

Table 1: Dataset description

Semester	Registration Dataset size	Students	Courses	Calendar exam days
Summer	5597	2497	128	6
First	17909	4001	294	12
Second	17493	3708	309	12

The difference in the dataset size is an important testing factor, where suitable shuffling techniques must exhibit stable behavior in small-, medium-, and large-scale datasets. Another important factor is the

availability of large multi-section courses, where the first semester has only 106 and the second semester has 110 courses, although it has fewer students and a smaller dataset. The summer semester had only 36 multi-section courses. Multi-section courses have increased conflicts with other courses and are extremely difficult to implement in exam schedules.

2.2 Shuffling algorithms

In this paper, we present a detailed analysis of five different shuffling algorithms, namely the Fisher–Yates Shuffle (Knuth Shuffle) [11], Sattolo's Algorithm [12], Bogosort[13], MergeShuffle[14], and Riffle Shuffle (Overhand Shuffle),[15] all of which are explained in this section.

2.2.1 Fisher–yates shuffle (knuth shuffle)

The algorithm was first introduced in 1938 as a card-shuffling technique. In a later version, the algorithm was updated to run in linear time O(n) by eliminating some counterproductive side aspects, and was suggested to be applied in computer applications [16]. The modern version of the algorithm was re-announced as Algorithm P, and has been highly adapted for computer applications [17]. The algorithm aims to shuffle the array elements at a low memory cost. It starts by iterating from the last array element to the second array element, and it swaps the selected element with a randomly chosen element that is located before it or even ensures a uniform probability across all permutations.

2.2.2 Sattolo's Algorithm

Sattolo's algorithm is a well-known variant of Fisher-Yates shuffling algorithm [12]. It was first introduced in 1986 as a random cyclic update, in which array elements can be moved to new positions in a single cycle of length n, which is the original size of the array. The algorithm is widely used in modern computing applications, such as cryptography and mathematical modeling, owing to its effectiveness and linear complexity O(n). However, this algorithm is not suitable for applications that require uniform probability distributions, because cyclic permutations may not contain all possible permutations.

2.2.3 Bogo sort

Bogosort is also known as permutation sort or stupid sort [18,21]. This helps to shuffle the datasets that are not sorted because it repeatedly generates random mutations until the list is sorted. As the end status is a sorted status, it is labelled as a sorting algorithm but is among the least effective because randomizing the data to reach a sorting status leads to unmeasurable conditions in terms of memory, time, and required computational power. Therefore, the expected time complexity is not clear, but the minimum expected complexity is O(n) and the maximum is limitless. However, generating random mutations leads to limited shuffling, which may affect the scheduling. In Bogosort developments, the Fisher–Yates Shuffle is normally applied until a sorted set is achieved. Therefore, in this study, we do not re-peat the Fisher–Yates Shuffle, but test the results of Bogosort and the effect-sorted datasets on the scheduling problem.

2.2.4 Merge shuffle

The MergeShuffle algorithm follows the von Neumann 1945 MergeSort algorithm, [17] particularly in terms of the required running time and di-vide-and-conquer design paradigm. This algorithm is relatively quick when it runs in nlog2(n)+O(1) time and is subject to parallelization procedures [14]. Additionally, the algorithm resulted in a swift performance even in increasing permutation environments compared with the Fisher–Yates Shuffle, which noticeably slowed down in the same environments. Moreover, Fisher–Yates Shuffle uses an increased number of recursive calls, whereas MergeShuffle sets a cutoff threshold to limit such calls.

2.2.5 Riffle shuffle (Overhand Shuffle)

Shuffling play cards are widely performed using the Gilbert-Shannon-Reeds (GSR) model, which is based on the Riffle shuffle algorithm [19]. Riffle shuffle was investigated in terms of the shuffles required to make the deck almost random. In 1992, Bayer and Diaconis suggested that seven shuffles should be adequate for randomizing card decks [20]. However, if the dataset is larger than a standard card deck, Riffle shuffle is not favored as the application complexity increases, and scalable challenges are

not well investigated.

2.3 Testing module

In this study, we built all the algorithms in the R-CRAN environment and applied them sequentially to the datasets listed in Table 1. Each algorithm was granted 1000 runs of application to each dataset, where each run had up to 100 iterations, and a random schedule was generated in each iteration. up to 100k schedules were generated using each algorithm for each dataset. If an iteration leads to an acceptable schedule, the remaining iterations are omitted from the next run. A testing module was created to determine the schedules that fulfilled all the obligatory constraints. Partial fulfillment was not achieved in this study. An acceptable schedule must have all obligatory constraints fulfilled by each student, where a student can have up to two exams per day and the exams must not conflict in time, knowing that each examination day can have up to four time slots. Figure 1 presents a schematic diagram of the testing module. Additionally, we tested for the adequacy of the calendar exam days, where an additional exam day was introduced if all methods failed to find an acceptable schedule in their 100k generated schedules, and we removed one day if an algorithm found a schedule in at least 50% of its runs. Reducing the number of exam days while still applying the obligatory constraints is highly favorable. Finally, to achieve a fair time comparison, we used the same device and testing module for all algorithms.

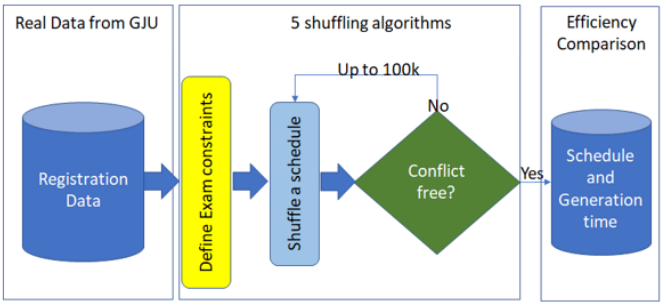

Figure 1: Testing module

73. Results and Discussion

In this section, we present the results of applying the chosen shuffling algorithms to the three-semester datasets. In addition to the time required to generate and test an acceptable schedule, we report the number of acceptable schedules generated by each algorithm. Moreover, the sufficiency of exam dates is well-tested by adding or removing exam days based on the achieved results. For instance, six days were insufficient to create an acceptable schedule among the 100k schedules generated by each algorithm. Therefore, we added a new examination day in which most algorithms led to acceptable schedules. On the other hand, we reduced the number of examination days for the first semester to 11 days, where 12 days led to many acceptable schedules and 11 days led to sufficient acceptable schedules, as presented in Table 2. We report that none of the 100k Fisher-Yates shuffling algorithms led to an acceptable schedule, while other algorithms found at least 1. Surprisingly, Riffle Shuffle reported exceptional performance, indicating that with fewer multi-section courses and an adequate number of exam days, it can be a good choice. Moreover, MergeShuffle exhibited an acceptable performance under the same conditions.

Table 2: Effect of shrinking first semester examination days

Algorithm	12 Days	11 Days
Fisher–Yates Shuffle	923	0
Sattolo's Algorithm	914	1
Merge Shuffle	983	3
Riffle Shuffle	1000	20

To apply Bogosort in our analysis while not repeating the Fisher-Yates shuffle, we sorted (ascending/descending) courses based on the student count as a result of Bogosort and then generated corresponding schedules. However, we reported that sorting never leads to acceptable schedules, indicating that it is not beneficial for scheduling problems. Other algorithms achieved acceptable schedules in most semesters but differed in the number of iterations that found an acceptable schedule and the time required to find them. Table 3 presents the detailed results of applying the selected algorithms to the second semester dataset. The Riffle shuffle could not lead to an acceptable schedule among the 100k generated schedules in the second-semester dataset. Possible reasons for this include the high number of courses, increased number of multi-section courses, and limited examination days.

Other algorithms reported acceptable schedules, where Merge Shuffle reported approximately three times the acceptable schedules compared to the Fisher-Yates and Sattolo algorithms.

Table 2: Results of applying chosen algorithms to second semester dataset

Algorithm	Count of acceptable schedules	Median of detection time (Seconds)
Fisher–Yates Shuffle	69	11.9
Sattolo's Algorithm	65	14.8
Merge Shuffle	198	13.9

The summer semester dataset was relatively small in terms of student count and registered courses. The number of examination days was increased to seven, because none of the algorithms reported an acceptable schedule using six examination days. Table 4 presents the detailed results for the summer semester. Acceptable schedules were achieved in most of the runs for all algorithms; however, Riffle and MergeShuffle required less time.

Table 3: Results of applying chosen algorithms to summer semester dataset

Algorithm	Count of acceptable schedules	Median of detection time (Seconds)
Fisher–Yates Shuffle	821	2.27
Sattolo's Algorithm	802	3.13
MergeShuffle	875	2.0
Riffle Shuffle	827	1.9

In our findings, MergeShuffle and Sattolo's algorithm reported stable performance in the diverse datasets we chose and found acceptable schedules, even with increased numbers of multi-section courses on limited exam days. However, their achievements differ when early schedules are acceptable using MergeShuffle, whereas Sattolo's algorithm can generate many unusable schedules before finding an acceptable schedule. Figure 2 is a histogram presenting the differences in performance between MergeShuffle and Sattolo's algorithm considering the second-semester findings. As assigned before, each run creates up to 100 schedules (x axis). The figure presents the frequency of finding the conflict-free schedule among the 100 created schedules. the figure shows that MergeShuffle is expected to find a conflict-free schedule among the early created schedules while using Sattolo's algorithm might lead to several useless conflicting schedules before finding the desired schedule.

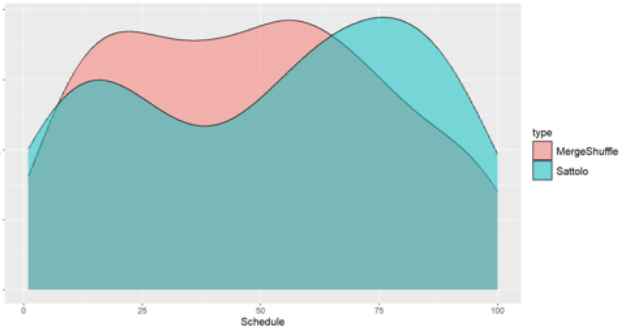

Figure 2: A histogram comparing the performance of MergeShuffle and Sattolo's algorithm

4. Conclusion and Future Scope

In this study, we investigated the effects of applying different shuffling algorithms in the process of scheduling university exams. Where sorted courses based on student counts did not succeed in creating an acceptable schedule, shuffling algorithms reported a significant success. We used real registration datasets from German Jordanian University (GJU). The chosen fully encoded datasets presented diverse environments, as they differed in terms of registration cases, number of students, and number of multi-section courses. To perform proper testing, we repeated the application of each algorithm to each dataset up to 100k times and reported our findings. We found that some shuffling algorithms, such as MergeShuffle and Sattolo's algorithm, worked properly under all environments but with higher time costs. Other algorithms might have a faster action but would work only in certain environments, such as the Fisher-Yates and Riffle algorithms. Moreover, the usage of MergeShuffle or Sattolo's algorithm is highly expected to result in a conflict-free schedule while achieving the same result of other algorithms is less

probable. Based on our findings, we highly advise to use shuffling algorithms in the scheduling problem while considering the features of registration datasets. Finally, our findings suggest that shuffling algorithms especially MergeShuffle and Sattolo's algorithm can be trusted for future developments of university systems to achieve conflict-free exam schedule generations

Author's statements

Acknowledgements I provide my sincere gratitude to the German Jordanian University for providing me with real registration data to complete this work.

Funding source- This study did not receive external funding.

Authors' contributions- Ahmad collected data, researched literature for related work, and was responsible for Conceptualization, Methodology, Scripting, Validation, Writing original draft and the manuscript, and over visualization

Conflict of interest- I declare no conflicts of interest.

Data availability- The encoded registration datasets used in this study are provided in the supplementary materials section.

References

[1] M. Mixer, S. Morrow, and L. Sukherman, "Two genetic algorithms for final exam scheduling," ACMLC 2024 - 2024 6th Asia Conference on Machine Learning and Computing, pp. 102–106, Mar. 2025, doi: 10.1145/3690771.3690798/ASSET/70AEA820-F810-40D2-8EB2-C59BC14A7529/ASSETS/GRAPHIC/ACMLC2024-27-FIG3.JPG.

[2] M. A. Al-Betar, A. T. Khader, and I. A. Doush, "Memetic techniques for examination timetabling," Ann Oper Res, vol. 218, no. 1, pp. 23–50, Nov. 2014, doi: 10.1007/S10479-013-1500-7/TABLES/14.

[3] M. Ayob, A. M. Ab Malik, S. Abdullah, A. R. Hamdan, G. Kendall, and R. Qu, "Solving a Practical Examina-tion Timetabling Problem: A Case Study," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 4707 LNCS, no. PART 3, pp. 611–624, 2007, doi: 10.1007/978-3-540-74484-9_53.

[4] B. Bilgin, E. Özcan, and E. E. Korkmaz, "An Experimental Study on Hyper-heuristics and Exam Timetabling," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 3867 LNCS, pp. 394–412, 2007, doi: 10.1007/978-3-540-77345-0_25.

[5] T. Alrawashdeh, K. G. Al-Moghrabi, and A. M. Al-Ghonmein, "A profiling-based algorithm for exams' scheduling problem," International Journal of Electrical and Computer Engineering (IJECE), vol. 13, no. 5, pp. 5483–5490, Oct. 2023, doi: 10.11591/ijece.v13i5.pp5483-5490.

[6] S. Petrovic, V. Patel, and Y. Yang, "Examination Timetabling with Fuzzy Constraints," Lecture Notes in Com-puter Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 3616 LNCS, pp. 313–333, 2005, doi: 10.1007/11593577_18.

[7] B. Bassimir and R. Wanka, "On the computation of robust examination timetables: methods and experi-mental results," Journal of Scheduling, vol. 28, no. 2, pp. 159–181, Nov. 2024, doi: 10.1007/S10951-024-00815-Y/FIGURES/8.

[8] V. Kolonias, G. Goulas, C. Gogos, P. Alefragis, and E. Housos, "Solving the Examination Timetabling Prob-lem in GPUs," Algorithms 2014, Vol. 7, Pages 295-327, vol. 7, no. 3, pp. 295–327, Jul. 2014, doi: 10.3390/A7030295.

[9] E. K. Burke and J. P. Newall, "Solving examination timetabling problems through adaption of heuristic or-derings," Ann Oper Res, vol. 129, no. 1–4, pp. 107–134, 2004, doi: 10.1023/B:ANOR.0000030684.30824.08/METRICS.

[10] A. K. Mandal, M. N. M. Kahar, and G. Kendall, "Addressing Examination Timetabling Problem Using a Par-tial Exams Approach in Constructive and Improvement," Computation 2020, Vol. 8, Page 46, vol. 8, no. 2, p. 46, May 2020, doi: 10.3390/COMPUTATION8020046.

[11] R. A. , & Y. F. Fisher, Statistical Tables for Biological, Agricultural and Medical Research. London: Hafner Press, 1938.

vol. 22, no. 6, pp. 315–317, May 1986, doi: 10.1016/0020-0190(86)90073-6.

[13] H. Gruber, M. Holzer, and O. Ruepp, "Sorting the Slow Way: An Analysis of Perversely Awful Randomized Sorting Algorithms," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 4475 LNCS, pp. 183–197, 2007, doi: 10.1007/978-3-540-72914-3_17.

[14] A. Bacher, O. Bodini, … A. H. preprint arXiv, and undefined 2015, "Mergeshuffle: a very fast, parallel random permutation algorithm," arxiv.orgA Bacher, O Bodini, A Hollender, J LumbrosoarXiv preprint arXiv:1508.03167, 2015•arxiv.org, 2015, Accessed: Jun. 22, 2025. [Online]. Available: https://arxiv.org/abs/1508.03167

[15] E. W. Weisstein, "Riffle Shuffle", Accessed: Jun. 03, 2025. [Online]. Available: https://mathworld.wolfram.com/RiffleShuffle.html

[16] H. ; Rutishauser, F. L. Bauer, . Rutishauser, N, and S. And, "Algorithm 235: Random permutation," Commun ACM, vol. 7, no. 7, p. 420, Jul. 1964, doi: 10.1145/364520.364540.

[12] S. Sattolo, "An algorithm to generate a random cyclic permutation," Inf Process Lett,

[17] D. E. Knuth, The art of computer programming, 3rd ed., vol. 2. USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[18] E. S. . Raymond, The new hacker's dictionary. MIT Press, 1996.

[19] P. D.-L. notes-monograph series and undefined 1988, "Group representations in probability and statistics," JSTORP DiaconisLecture notes-monograph series, 1988•JSTOR, Accessed: Jun. 10, 2025. [Online]. Available: https://www.jstor.org/stable/4355560

[20] D. Bayer, P. D.-T. A. of A. Probability, and undefined 1992, "Trailing the dovetail shuffle to its lair," JSTOR, vol. 2, no. 2, pp. 294–313, 1992, Accessed: Jun. 10, 2025. [Online]. Available: https://www.jstor.org/stable/295975.

[21] Abbas, M., Khan, T. I., & Jam, F. A. (2025). Avoid Excessive Usage: Examining the Motivations and Outcomes of Generative Artificial Intelligence Usage among Students. Journal of Academic Ethics, 1-20.

[22] Moghavvemi, S., & Jam, F. A. (2025). Unraveling the influential factors driving persistent adoption of ChatGPT in learning environments. Education and Information Technologies, 1-28.