# Planned Random Algorithm

**Anurag Pant, Kunwar Agrawal and B. K. Tripathy**

**Abstract** Computers are very systemized and none of the procedures conducted by them are random. But computers are seldom required to generate a random number for many practical applications like gaming, accounting, encryption/decryption and many more. The number generated by the computer relies on the time or the CPU clock. A given computer can be programmed to return random number (or character) arrays from a number (or character) data set. The returned dataset can have repeated values. Even though the repeated values are not related with its degree of randomness (in fact, it may be a sign of higher randomization), but still to humans, it appears as biased or not random. We propose an algorithm to minimize repetition of values in the returned data set so as to make it appear more random. The concept proposes returning a data set by using biased or nonrandom procedure in order to make it more random in "appearance".

**Keywords** Apophenia · Planned random · Fisher–Yates algorithm
Random number applications · PR Algorithm

## 1 Introduction

Randomness describes a lack of pattern or predictability in a data set or any activities. Computer-generated random numbers are often required to help in various real-life applications such as gaming, statistics, encryption, and many more. Gaming especially requires computers to generate numbers that are used to make

A. Pant (✉) · K. Agrawal · B. K. Tripathy
School of Computer Science and Engineering, VIT University,
Vellore 632014, Tamil Nadu, India
e-mail: anurag.pant2014@vit.ac.in

K. Agrawal
e-mail: kunwar.agrawal2014@vit.ac.in

B. K. Tripathy
e-mail: tripathybk@vit.ac.in



S. C. Satapathy et al. (eds.), *Information and Decision Sciences*,
Advances in Intelligent Systems and Computing 701,
https://doi.org/10.1007/978-981-10-7563-6_13

games more interesting and life like. The numbers are procured by standard random algorithms, which are predefined in various libraries in almost all programming languages. The numbers generated, even though unpredictable, may seem repetitive or may appear to follow a pattern.

The literal definition of randomness is stated as the lack of predictability or pattern. Computer-generated random numbers are meant to be unpredictable. Even though this is true, they often show some kind of pattern or repetition. Human beings perceive these repeating numbers as not being completely random. This can be attributed to the human tendency to discern patterns out of total randomness, referred to as apophenia or patternicity [1, 3, 6]. This tendency makes it difficult for interactions between humans and computers to actually appear to be random. Thus the visible randomness or apparent randomness is often less for such a data set, generated with the standard random algorithms [4].

*Example 1* in many multiplayer-fighting games (like Injustice: Gods Among Us or WWE, just for reference) there is an option of randomly selecting a player or arena. The random number generated help in choosing the player or the arena. Thus the result directly affects the user.

*Example 2* in cryptography or encryption/decryption applications, random numbers are very useful and their use is inevitable. In such applications the user is not directly affected though.

Example 1 is a clear example of apparent randomness or visible randomness. Such applications are what we are targeting to improve.

For any practical application, the random numbers are often directly affecting the user (human), rather than a client (machine) like in Example 1. For such applications, computer-generated random numbers (using standard algorithms) often fail to keep a high apparent randomness because of the repetition. Thus, we have proposed a custom algorithm to return values which are nonrepeating. The algorithm was devised to return values in a way, that makes them seem less repetitive to the user (prevent values with similar attributes from appearing in quick succession. This introduces an element of planning within the algorithm, thereby, giving way to planned randomness.

## 2 Previous Work

Apple, an American multinational technology company headquartered in Cupertino, California, had designed a shuffle feature for the music software that was installed on their devices to randomize the songs in the device's playlist. However, in 2010, they decided to make the shuffle feature "less random" to increase its apparent randomness since, true randomization often creates counter-intuitively dense clusters which sometimes resulted in the repetition of the same song in quick succession. This decreased the apparent randomness of the shuffle feature.

The CEO of Apple, Steve Jobs had stated, "We are making it (the shuffle) less random to make it feel more random" [2].

### 2.1 Fisher–Yates Algorithm

Ronald Fisher and Frank Yates developed Fisher–Yates algorithm in 1938. The algorithm used very simple explanation and technique and could be described with a pencil and paper. It was also called Pencil and Paper method. A random number was picked from the unshuffled list and was then put into an output list. This was done till all the numbers in the unshuffled list were added to the new shuffled list in a random order. The algorithm was unbiased and allowed for selection from n! permutations while the time complexity of the algorithm was $O(n^2)$ [7].

### 2.2 Modern Fisher–Yates Algorithm (Algorithm P)

Richard Durstenfeld described this algorithm in 1964. This is also known as Knuth's implementation of Fisher–Yates (KFY) Algorithm since it was later popularized by Donald E. Knuth in his book as Algorithm P. The algorithm is exactly like Fisher–Yates Algorithm, except the number which is selected at random from the list is interchanged with the last not-selected number in the same list, thus decreasing the time complexity from $O(n^2)$ to O(n) [7].

### 2.3 Sattolo's Algorithm

Sattolo's Algorithm is similar to KFY Algorithm, except that in this case the random number is chosen from 1 to i-1 instead of from 1 to i (as is done in Fisher–Yates). This biases the algorithm since now the choice has to be made from (n-1)! permutations instead of n! permutations. The time complexity of the algorithm is O(n) [7].

## 3 Applications

The Planned Random algorithm (hereafter, referred to as the PR Algorithm) proposed by us, aims at improving the apparent randomness in various applications. All the activities or applications that require the generation of random numbers and are directly affecting the user, fall into such category. Some are mentioned below.

### 3.1 Gaming

Gaming is a very diverse field, with various platforms, genres, models and graphics type. Even with so many diversities, all the games require random numbers directly or indirectly. These random numbers impact the choices made within the game as well as the gameplay. Therefore, these choices will directly impact the user and hence, the use of the PR Algorithm to shuffle these choices could help to make them look more random to the user thereby, improving the gameplay experience for the user. Examples: For randomly choosing an arena or player in a fighting game, for randomly creating avatars in simulation or RPGs, for map selection within the game, et cetera.

### 3.2 Songs and Shuffling

Most music devices provide the option of shuffling while listening to songs. The order of the songs in the playlist directly impacts the user and affects the overall experience. If songs are played in a similar order again and again, the experience becomes monotonous. Therefore, PR Algorithm can be used to shuffle the songs in a manner that they appear to be more random (increased apparent randomness) to the user. Example: songs can be shuffled in a manner that tends to avoid clubbing songs from the same album or same artist, or songs can be shuffled to avoid same genre songs from being played consecutively, et cetera [5].

## 4 Algorithm

In our proposed shuffling algorithm aimed at increasing the apparent randomness, time-based randomization was used with the help of "srand" and the "rand" function of the "stdlib.h" header file. The algorithm has a time complexity of O(n) which makes it as efficient as the KFY Algorithm.

As the algorithm takes the values (input) in the form of an unshuffled array, the two key elements are the indices and the corresponding values. Indices are used to uniquely identify each item in the array and will always differ for each item in the array. However, the corresponding values of the items can repeat or they can be different. These values are indicative of the properties which may be same for different items, for example, in case of applications in music, these values can represent the same artists or same albums (Figs. 1 and 2).

```
Original List: 1 2 2 5 5 6 7 8 10 10
Shuffled List: 6 5 2 8 10 5 7 1 10 2
Process returned 0 (0x0)   execution time : 0.058 s
Press any key to continue.
```

**Fig. 1** Sample input/ output of the algorithm, implemented in C++ and compiled using code blocks IDE

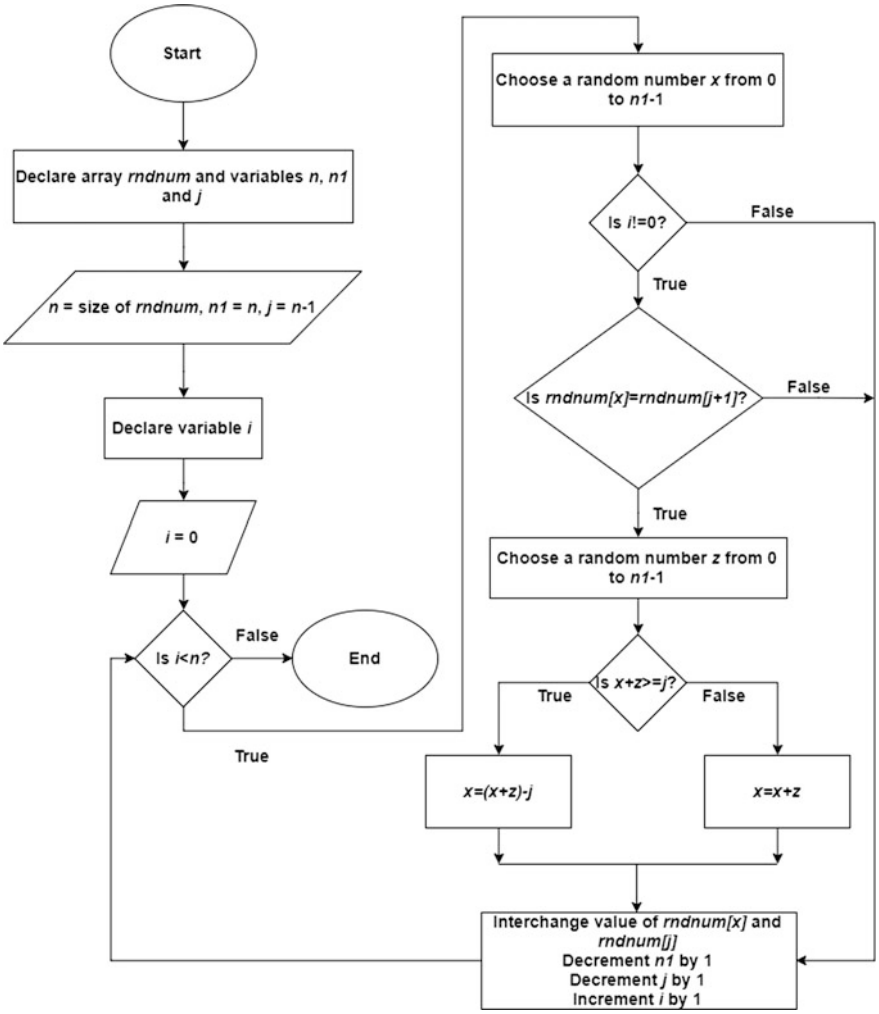


**Fig. 2** Flowchart for the algorithm

```
   Create an array rndnum and set it equal to the
unshuffled list of numbers
   Set n equal to the size of the array
   Initialize n1 to n
   Initialize j to n-1
   Start the while loop for i less than n
     Choose a random number from 0 to n1-1 and store in x
     Check if while loop isn't in its first iteration
   (condition 1)
       If condition 1 is true, check if rndnum[x] is equal
   to rndnum[j+1] (condition 2)
       If condition 2 is true, then:
         Step 1: Choose a random number from 0 to n1-1 and
   store in z
         Step 2: Check if x+z is greater than or equal to
   j (condition 3)
         Step 3: If condition 3 is true, then set x equal
   to (x+z)-j
         Step 4: If condition 3 is false, then set x equal
   to x+z
         End the condition 2 loop
       End the condition 1 loop
     Interchange values of rndnum[x] and rndnum[j]
     Decrement n1 by 1
     Decrement j by 1
   End the while loop
```

## 5 Testing and Comparative Analysis

In order to compare the success of the proposed PR Algorithm in increasing apparent randomness in the shuffling of a given list, it was compared against the KFY Algorithm. The PR Algorithm was not compared against the original Fisher–Yates because then there would be a significant difference between the time complexities of the compared algorithms (original Fisher–Yates has a time complexity of $O(n^2)$ while the PR Algorithm has a time complexity of $O(n)$). It was not further compared to Sattolo's algorithm, since both KFY and Sattolo's algorithms have a very similar implementation except for the fact that Sattolo's algorithm lacks in producing the full set of n! possible permutations (Sattolo's algorithm produces only $(n - 1)!$ set of permutations) and thereby, adds a bias to the results. The aim was to compare the proposed shuffling algorithm, aimed at producing results biased to increase apparent randomness, against an unbiased shuffling algorithm. Therefore, KFY Algorithm was found to be the best contender.

The comparison was done on the basis of 2 tests that were created to check for qualities in the shuffling that could contribute to a decrease in the apparent randomness of the shuffled lists.

A Windows 10 laptop with Intel i5 and 8 GB RAM was used to run the tests. Code::Blocks 13.12 was installed on the system. The code was written in C++ and run using several libraries like “stdlib.h”, “dos.h”, “time.h” and “string.h”.

## 5.1 Test 1

In this test, an array of 100 numbers (kept the same for all the test cases), taken with repetition from the first 10 natural numbers, is shuffled using the algorithm being evaluated. The new shuffled list produced is shuffled again using the algorithm in the next iteration. This is done for a 1000 iterations. The aim of this test is to compare the new shuffled list against the previous shuffled list, to check for a match. If they match, a counter inside the program is incremented by 1 to mark the repetition of patterns. At the end of the 1000 iterations, the counter is displayed. This test is carried out to ensure that the user does not find the patterns produced to be repetitive, for example, a user listening to a playlist of songs that has been shuffled randomly, would want his playlist to be shuffled in an order that is not similar to the order it was shuffled in previously. Repetition of patterns decreases the apparent randomness of the shuffle.

Results: This test was carried out 5 times each for both the PR Algorithm and the KFY Algorithm. Both the PR Algorithm and the KFY Algorithm gave 0 pattern repetitions in all 5 cases of Test 1. Thus, it can successfully be concluded that even though, the PR Algorithm was biased to increase the apparent randomness within the shuffle of a pattern, this bias does not in turn reduce the apparent randomness between shuffles and therefore, doesn’t lead to pattern repetition.

## 5.2 Test 2

For this test, an array of 100 elements (same as the one taken in Test 1), is shuffled using the algorithm being evaluated. The new shuffled list produced is shuffled again using the algorithm in the next iteration. This is done for 1000 iterations. The aim of this test is to evaluate the apparent randomness within the shuffled algorithm. Since the indices of the items are the unique identifying feature of each item and the corresponding values are properties which may be same for different items (same type of character in a game (hero or villain), same album type of different songs), the repetition of items with same common properties consecutively will reduce the apparent randomness of the shuffled list. Therefore, during each shuffle, if the corresponding value of an item in the list matched the corresponding value of the item directly next to it, the counter is incremented by 1. At the end of the 1000 iterations, the counter is displayed. The lesser the value of the counter, the greater the apparent randomness of the shuffled list [4].

**Table 1** Results for test 2

| KFY Algorithm | | PR Algorithm | |
|---|---|---|---|
| Cases | Values | Cases | Values |
| 1 | 8926 | 1 | 1517 |
| 2 | 8967 | 2 | 1424 |
| 3 | 8909 | 3 | 1574 |
| 4 | 8988 | 4 | 1395 |
| 5 | 9030 | 5 | 1409 |
| Total repetitions: 44820 | | Total repetitions: 7319 | |
| Avg. repetitions: 8964 | | Avg. repetitions: 1463.8 | |

Results: This test was carried out 5 times each for both the PR Algorithm and the KFY Algorithm. The results were as follows (Table 1).

Thus, it can be successfully concluded that the PR Algorithm increased the apparent randomness of the shuffle significantly, as it lead to a 83.67% decrease in the average repetitions per case from KFY Algorithm in Test 2 (Fig. 3).

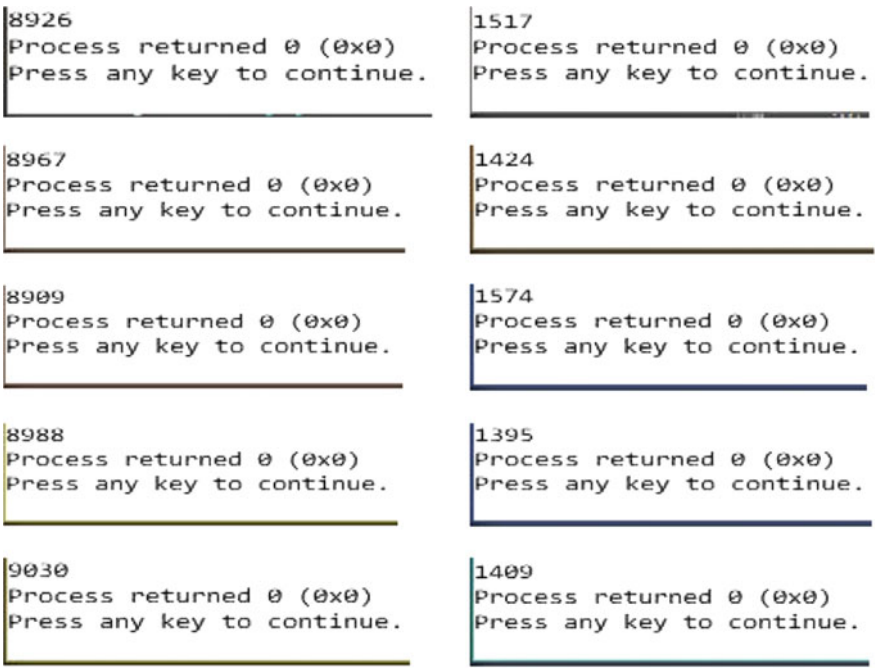


**Fig. 3** Results from KFY Algorithm (left column) and PR Algorithm (right column) for Test 2

## 6 Drawbacks

Even though the algorithm followed O(n) time complexity and gave better results as compared to the KFY Algorithm, there was one drawback. The increased visible randomness came at a price of biasing and altering of the natural random results. Thus, the algorithm may return predictable patterns in smaller sample groups but for longer sample groups it is definitely unpredictable.

## 7 Future Work

The algorithm can be extended to all practical applications and can even be made to learn by using machine learning. The algorithm can grow every day with its use.

For example: In the application of Songs and Shuffling, it can monitor the number of times a song is played. As a human brain will tend to perceive the occurrence of the most played song as highly repetitive, the algorithm will prevent playing of that song in the beginning.

## 8 Conclusion

In this paper, we introduced an approach to increase the apparent randomness of any dataset using an algorithm called Planned Random algorithm. The core of the algorithms lies in the definition of apparent randomness and its increase, by the introduction of the element of planning within the already written code or program. The PR Algorithm can be easily extended and scaled to suit any custom application that interacts with humans to reduce possible patterns and repetitions that may be discerned by humans.

## References

1. Beitman, Bernard D.: Brains seek patterns in coincidences. Psychiatr. Ann. **39**(5), 255 (2009)
2. Dailymail UK., http://www.dailymail.co.uk/home/moslive/article-1334712/Humans-concept-randomness-hard-understand.html
3. Scientific American, https://www.scientificamerican.com/article/patternicity-finding-meaningful-patterns
4. Tech Times, http://www.techtimes.com/articles/34366/20150220/are-spotify-and-itunes-random-shuffle-features-really-random-not-now-but-they-used-to-be.htm
5. The Telegraph, http://www.telegraph.co.uk/technology/11429317/The-biggest-myths-about-technology.html
6. Wikipedia, https://en.wikipedia.org/wiki/Apophenia
7. Wikipedia, http://en.wikipedia.org/wiki/Fisher-Yates_shuffle