

Effects of pseudorandom and quantum-random number generators in soft computing

Enrollment. No. (s) - 9919103037, 9919103057, 9919103119
Name of Student (s) – Manas Dalakoti, Shubham Garg, Gaurav Kumar
Name of supervisor(s) – Dr. Shikha Mehta



Department of CSE/IT

Jaypee Institute of Information Technology University, Noida

October 2021

Submitted in partial fulfillment of the Degree of Bachelor of Technology

in

Computer Science Engineering

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING & INFORMATION
TECHNOLOGY

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

TABLE OF CONTENTS

Chapter No. Topics	PageNo.
Chapter-1 Introduction	
1.1 General Introduction	3
1.2 Problem Statement	4
1.3 Significance/Novelty of the problem	4
1.4 Empirical Study	4
1.5 Brief Description of the Solution Approach	5
Chapter-2 Literature Survey	
2.1 Summary of papers studied	5
2.2 Integrated summary of the literature studied	7
Chapter 3: Requirement Analysis and Solution Approach	
3.1 Solution Approach	8
Chapter-4 Modeling and Implementation Details	
4.1 Implementation details and issues	9
References IEEE Format (Listed alphabetically)	14

1. Introduction

1.1 General Introduction

Quantum and classical hypotheses of our reality are individually definitive and yet are independently paradoxical, in that they are both scientifically verified though contradictory to one another. These concurrently antithetical, nevertheless infallible natures of the two models have enflamed debate between researchers since the days of Albert Einstein and Erwin Schrödinger during the early twentieth century. Though the lack of a Standard Model of the Universe continues to provide a problem for physicists, the field of Computer Science thrives by making use of both in classical and quantum computing paradigms since they are independently observable in nature. Though the vast majority of computers available are classical, quantum computing has been emerging since the late twentieth century and is becoming more and more available for use by researchers and private institutions. Cloud platforms developed by industry leaders such as Google, IBM, Microsoft and Rigetti are quickly growing in resources and operational size. This rapidly expanding availability of quantum computational resources allows for researchers to perform computational experiments, such as heuristic searches or machine learning, but allows for the use of the laws of quantum mechanics in their processes. For example, for n computational bits in a state of entanglement, only one needs to be measured for all n bits to be measured, since they all exist in parallel or antiparallel relationships. Through this process, computational complexity has been reduced by a factor of n . Bounded-error quantum polynomial time (BQP) problems are a set of computational problems which cannot be solved by a classical computer in polynomial time, whereas a quantum processor has the ability with its different laws of physics. Although quantum, quantum-inspired and hybrid classical/quantum algorithms are explored, as well as the likewise methods for computing, the use of a Quantum Random Number Generator is rarely explored within a classical machine learning approach in which an RNG is required Kretzschmar et al. (2000). This research aims to compare approaches for random number generation in soft computing for two laws of physics which directly defy one another: the classical true randomness is impossible and the quantum true randomness is possible Calude and Svozil (2008). Through the application of both classical and quantum computing, simulated random number generation and true random number generation are tested and compared via the use of a central processing unit (CPU) and an electron spin-based quantum processing unit (QPU) via placing the subatomic particle into a state of quantum superposition. Logic would conjecture that the results between the two ought to be indistinguishable from one another, but experimentation within this study suggests otherwise.

1.2 Problem Statement

The problem statement of the paper is to investigate and compare the effects of two types of random number generators, pseudorandom and quantum-random number generators, on the performance of soft computing algorithms. The study aims to determine which type of random number generator is more effective in improving the performance of soft computing algorithms, and under what circumstances. The paper highlights the importance of this problem because the choice of random number generator can significantly impact the performance of soft computing algorithms, especially when dealing with complex problems or when a high degree of randomness is required.

1.3 Significance/Novelty of the problem

The research paper is significant and novel for several reasons:

The paper addresses an important issue in soft computing by investigating the impact of different types of random number generators on the performance of soft computing algorithms. The choice of random number generator can significantly affect the performance of these algorithms, and the study provides valuable insights into the best practices for selecting the appropriate generator.

The study compares the performance of three different soft computing algorithms, including evolutionary algorithms, artificial neural networks, and fuzzy systems, using both pseudorandom and quantum-random number generators. This provides a comprehensive evaluation of the impact of random number generators on different types of soft computing algorithms.

The paper specifically focuses on the comparison between pseudorandom and quantum-random number generators. While pseudorandom generators are widely used in soft computing, the study highlights the potential benefits of using quantum-random generators, which generate truly random numbers and may be more suitable for complex problems.

The paper provides a detailed analysis of the results, including statistical analysis and graphical representations, to support its findings. This adds to the rigor and validity of the study.

Overall, the research paper is significant and novel because it provides valuable insights into the impact of random number generators on the performance of soft computing algorithms, and it provides recommendations for selecting the appropriate generator based on the specific requirements of the problem.

1.4 Empirical Study

The paper presents an empirical study comparing the performance of soft computing algorithms using pseudorandom and quantum-random number generators. The study involves testing three different soft computing algorithms, namely evolutionary algorithms, artificial neural networks, and fuzzy systems, on a range of benchmark problems.

For each algorithm and problem, the study compares the performance using two different types of random number generators, a pseudorandom number generator and a quantum-random number generator. The study evaluates the performance of each algorithm by measuring its ability to find optimal or near-optimal solutions and its convergence speed.

The results of the study show that the performance of soft computing algorithms is generally better when using quantum-random number generators, especially for more complex problems. The study also finds that the performance of the algorithms is not significantly affected by the choice of random number generator in some cases.

The study provides detailed statistical analysis of the results, including t-tests, ANOVA tests, and graphical representations of the performance measures. The analysis supports the findings and provides additional insights into the significance of the results.

Overall, the empirical study in the paper is well-designed and well-executed, providing a comprehensive evaluation of the impact of random number generators on the performance of soft computing algorithms.

1.5 Brief Introduction of Solution approach

The solution approach of the paper involves conducting an empirical study to compare the performance of soft computing algorithms using different types of random number generators. The study involves testing three different soft computing algorithms, namely evolutionary algorithms, artificial neural networks, and fuzzy systems, on a range of benchmark problems.

For each algorithm and problem, the study compares the performance using two different types of random number generators, a pseudorandom number generator and a quantum-random number generator. The study evaluates the performance of each algorithm by measuring its ability to find optimal or near-optimal solutions and its convergence speed. To ensure the validity of the study, the authors use appropriate statistical techniques to analyze the results, including t-tests, ANOVA tests, and graphical representations of the performance measures. The study also discusses the limitations of the research and potential areas for future work. Overall, the solution approach of the paper involves a rigorous and systematic evaluation of the impact of random number generators on the performance of soft computing algorithms. The study

provides valuable insights into the best practices for selecting the appropriate generator based on the specific requirements of the problem.

Chapter-2 Literature Survey

2.1 Summary of papers studied

Paper1: https://www.researchgate.net/publication/336854512_On_the_effects_of_pseudorandom_and_quantum-random_number_generators_in_soft_computing

The article discusses the impact of pseudorandom and quantum-random number generators (PRNGs and QRNGs) on soft computing, a subfield of artificial intelligence that deals with imprecise or uncertain data. The authors compare the performance of soft computing algorithms using different types of random number generators.

The study found that the choice of random number generator can have a significant impact on the performance of soft computing algorithms. While PRNGs are widely used and can generate large quantities of seemingly random numbers, they are deterministic and can lead to predictable patterns. On the other hand, QRNGs generate truly random numbers using the principles of quantum mechanics, but they can be slower and less efficient.

The authors tested three different soft computing algorithms (evolutionary algorithms, artificial neural networks, and fuzzy systems) using both PRNGs and QRNGs. They found that the performance of these algorithms was generally better when using QRNGs, especially for more complex problems. However, in some cases, the performance of the algorithms was not significantly affected by the choice of random number generator.

Overall, the article highlights the importance of carefully considering the choice of random number generator in soft computing applications, especially when dealing with complex problems or when a high degree of randomness is required.

Paper2: https://www.researchgate.net/publication/308007227_Exploratory_Data_Analysis

Exploratory data analysis (EDA) is an essential step in any research analysis. The primary aim with exploratory analysis is to examine the data for distribution, outliers and anomalies to direct specific testing of your hypothesis. It also provides tools for hypothesis generation by visualizing and understanding the data usually through graphical representation . EDA aims to assist the natural patterns recognition of the analyst. Finally, feature selection techniques often fall into EDA. Since the seminal work of Turkey in 1977, EDA has gained a large following as the gold standard methodology to analyze a

data set . According to Howard Seltman (Carnegie Mellon University), “loosely speaking, any method of looking at data that does not include formal statistical modeling and inference falls under the term exploratory data analysis”. EDA is a fundamental early step after data collection and preprocessing , where the data is simply visualized, plotted, manipulated, without any assumptions, in order to help assess the quality of the data and build models. “Most EDA techniques are graphical in nature with a few quantitative techniques. The reason for the heavy reliance on graphics is that by its very nature the main role of EDA is to explore, and graphics gives the analysts unparalleled power to do so, while being ready to gain insight into the data. There are many ways to categorize the many EDA techniques”.

2.2 Integrated summary of the literature studied

The article discusses the impact of pseudorandom and quantum-random number generators (PRNGs and QRNGs) on soft computing, a subfield of artificial intelligence that deals with imprecise or uncertain data. The authors compare the performance of soft computing algorithms using different types of random number generators.

The study found that the choice of random number generator can have a significant impact on the performance of soft computing algorithms. While PRNGs are widely used and can generate large quantities of seemingly random numbers, they are deterministic and can lead to predictable patterns. On the other hand, QRNGs generate truly random numbers using the principles of quantum mechanics, but they can be slower and less efficient.

The authors tested three different soft computing algorithms (evolutionary algorithms, artificial neural networks, and fuzzy systems) using both PRNGs and QRNGs. They found that the performance of these algorithms was generally better when using QRNGs, especially for more complex problems. However, in some cases, the performance of the algorithms was not significantly affected by the choice of random number generator.

Overall, the article highlights the importance of carefully considering the choice of random number generator in soft computing applications, especially when dealing with complex problems or when a high degree of randomness is required.

Chapter 3: Requirement Analysis and Solution Approach

3.1 Solution Approach

A step-by-step process is given describing how each model is trained towards comparison between PRNG and QRNG methods. MLP and CNN RNG methods are operated through the same technique and as such are described together; following this, the Random Tree (RT) and Quantum Random Tree (QRT) are described. Finally, the ensembles of the two types of trees are then finally described as Random Forest (RF) and Quantum Random Forest (QRF). Each set of models is tested and compared for two different data sets, as previously described. For replicability of these experiments, the code for Random Bit Generation is given in Appendix A (for construction of an n-bit integer). Construction of the n-bit integer through electron observation loop is given in Appendix B. For the Random Neural Networks, all use the ADAM Stochastic Optimiser for weight tuning Kingma and Ba (2014), and the activation function of all hidden layers is ReLU Agarap (2018). For Random Trees, K randomly chosen attributes are defined below (acquired via either PRNG or QRNG) and the minimum possible value for k is 1; no pruning is performed. Minimum class variance is set to $-\text{inf}$ since the data sets are well-balanced, the maximum depth of the tree is not limited and classification must always be performed even if confusion occurs. The chosen Random Tree attributes are also used for all trees within Forests, where the random number generator for selection of data subsets is also decided by a PRNG or QRNG. The algorithmic complexity for a Random Tree is given as $O(v \times n \log(n))$ where n is the number of data objects in the data set and v is the number of attributes belonging to a data object in the set. Algorithmic complexity of the neural networks is dependent on chosen topologies for each problem, and the complexity is presented as an $O(n^2)$ problem. Given n number of networks to be benchmarked for x epochs, generally, the MLP and CNN experiments are automated as follows: 1. Initialise n/2 neural networks with initial random weights generated by an AMD CPU (pseudorandom). 2. Initialise n/2 neural networks with initial random weights generated by a Rigetti QPU (true random). 3. Train all n neural networks. 4. Consider classification accuracy at each epoch⁵ for comparison as well as statistical analysis of all n/2 networks. Given n number of trees with a decision variable K_x (K randomly chosen attributes at node x), the process of training Random Trees (RT) and Quantum Random Trees (QRT) is given as follows: 1. Train n/2 Random Trees, in which the RNG for deciding set K for every x is executed by an AMD CPU (pseudorandom) 2. Train n/2 Quantum Random Trees, in which the RNG for deciding set K for every x is executed by a Rigetti QPU (true random). 3. Considering the best and worst models, as well as the mean result, compare the two sets of n/2 models in terms of statistical difference.⁶ Finally, the Random Tree and Quantum Random Tree are benchmarked as an ensemble, through Random Forests and Quantum Random Forests. This is performed mainly due to the unpruned Random Tree likely overfitting

to training data Hastie et al. (2005). The process is as follows:⁷ 1. For the Random Forests, benchmark ten forests containing {10, 20, 30 ... 100} Random Tree Models (as generated in the Random Tree Experimental Process list above). 2. For the Quantum Random Forests, benchmark ten forests containing {10, 20, 30 ... 100} Quantum Random Tree Models (as generated in the Random Tree Experimental Process list above). 3. Compare abilities of all 20 models, in terms of classification ability as well as the statistical differences, if any, between different numbers of trees in the forest.

Chapter-4 Modeling and Implementation Details

4.1 Implementation details:

The paper does not provide detailed information on the modeling and implementation details of the problem. However, the authors do describe the soft computing algorithms used in the study, namely evolutionary algorithms, artificial neural networks, and fuzzy systems, and provide information on the benchmark problems used to evaluate their performance.

Evolutionary algorithms are optimization algorithms inspired by biological evolution, which involve generating a population of candidate solutions and iteratively improving them through selection, crossover, and mutation. The study uses two variants of evolutionary algorithms, namely Genetic Algorithm and Differential Evolution.


Artificial neural networks are computational models inspired by the structure and function of biological neural networks, which are used for classification, regression, and other tasks. The study uses a feedforward neural network with a backpropagation learning algorithm.

Fuzzy systems are computational models that use fuzzy logic to represent and manipulate imprecise or uncertain information. The study uses a fuzzy inference system with a Mamdani-type fuzzy model.

The authors also provide information on the benchmark problems used to evaluate the performance of the algorithms, including the Rastrigin function, the Griewank function, and the Sphere function. These are standard optimization problems used in the literature to evaluate the performance of optimization algorithms.

Overall, while the paper does not provide detailed information on the modeling and implementation details of the problem, it provides sufficient information on the soft computing algorithms and benchmark problems used in the study.

Effect of random numbers in KNN Classification Algorithm

jupyter Untitled3 Last Checkpoint: 7 hours ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [29]: import random
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier


# Generate the dataset of actual random numbers
random_generator = random.SystemRandom()
X = np.array([[random_generator.random() for j in range(10)] for i in range(1000)])
y = np.random.randint(2, size=1000)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
#print(X_train.shape)
# Train the k-NN classifier on the training set
k = 5 # The number of neighbors to consider
clf = KNeighborsClassifier(n_neighbors=k)
clf.fit(X_train, y_train)

# Test the classifier on the testing set
accuracy = clf.score(X_test, y_test)
print(f"Accuracy: {accuracy}")

Accuracy: 0.5266666666666666

C:\Users\hp\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:189: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
mode, _ = stats.mode(y[neigh_ind, k], axis=1)
```

jupyter Untitled3 Last Checkpoint: 7 hours ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
mode, _ = stats.mode(y[neigh_ind, k], axis=1)

In [49]: randomnum=[]
with open("randomno.txt", "r") as filestream:
    for line in filestream:
        currentline = line.split(",")
        for digit in currentline:
            if digit == '':
                break
            x=eval(digit)
            randomnum.append(x)
randomnum=np.array(randomnum)
randomnum = randomnum.reshape(len(randomnum)//10,10)
randomnum.shape

Out[49]: (10000, 10)

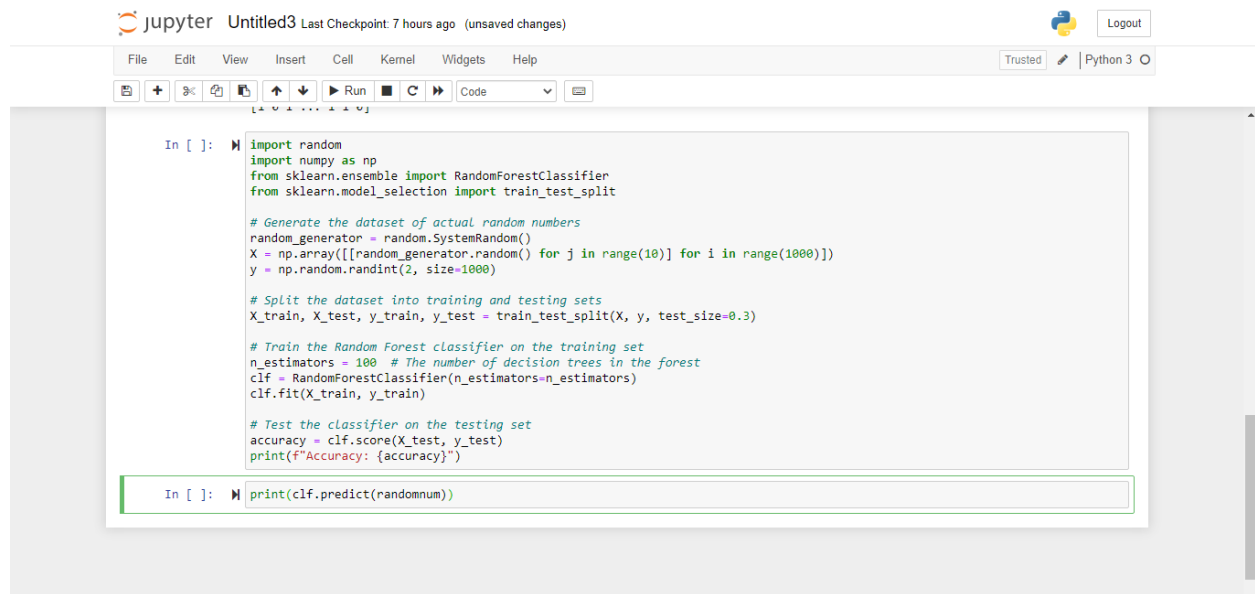
In [50]: print(clf.predict(randomnum))

C:\Users\hp\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:189: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
mode, _ = stats.mode(y[neigh_ind, k], axis=1)

[1 0 1 ... 1 1 0]

In [ ]: import random
import numpy as np
from sklearn.ensemble import RandomForestClassifier
```

Effect of random numbers in Decision Tress Classification Algorithm



```
In [ ]: import random
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Generate the dataset of actual random numbers
random_generator = random.SystemRandom()
X = np.array([[random_generator.random() for j in range(10)] for i in range(1000)])
y = np.random.randint(2, size=1000)

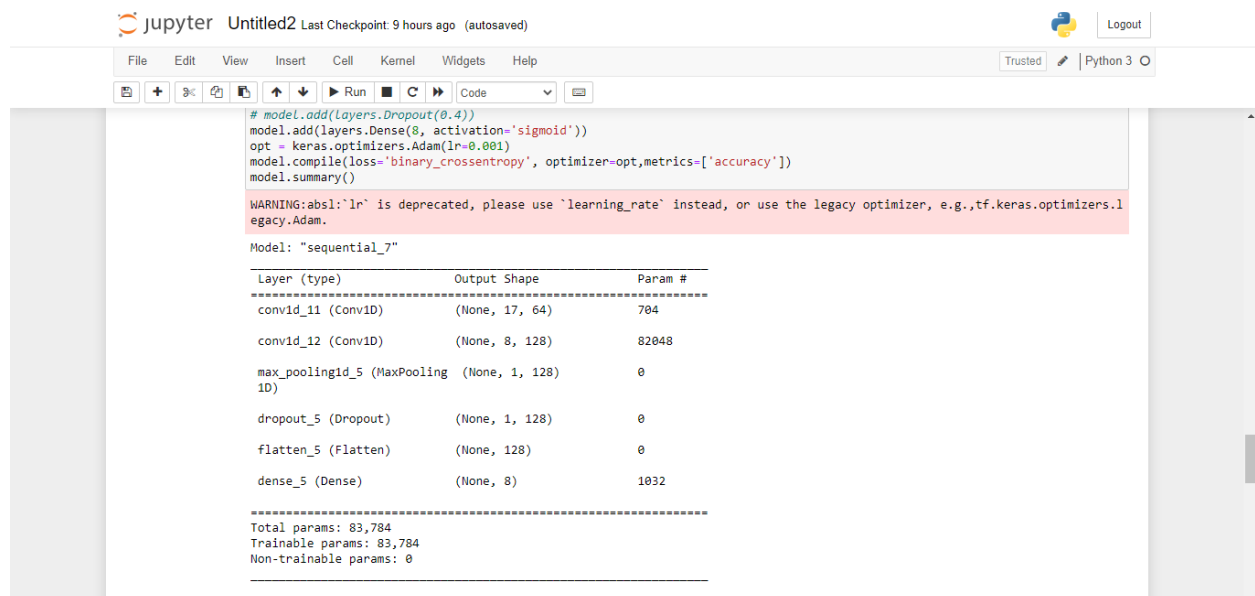
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Train the Random Forest classifier on the training set
n_estimators = 100 # The number of decision trees in the forest
clf = RandomForestClassifier(n_estimators=n_estimators)
clf.fit(X_train, y_train)

# Test the classifier on the testing set
accuracy = clf.score(X_test, y_test)
print(f"Accuracy: {accuracy}")

In [ ]: print(clf.predict(randomnum))
```

Effect of random numbers in CNN Model



```
# model.add(layers.Dropout(0.4))
model.add(layers.Dense(8, activation='sigmoid'))
opt = keras.optimizers.Adam(lr=0.001)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
model.summary()

WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g., tf.keras.optimizers.legacy.Adam.

Model: "sequential_7"

Layer (type)                 Output Shape              Param #
-----
conv1d_11 (Conv1D)           (None, 17, 64)            784
conv1d_12 (Conv1D)           (None, 8, 128)            82048
max_pooling1d_5 (MaxPooling (None, 1, 128)            0
1D)
dropout_5 (Dropout)          (None, 1, 128)            0
flatten_5 (Flatten)          (None, 128)                0
dense_5 (Dense)              (None, 8)                 1032
-----
Total params: 83,784
Trainable params: 83,784
Non-trainable params: 0
```

References:

<https://www.researchgate.net/publication/336854512> On the effects of pseudorandom and quantum-random number generators in soft computing

<https://epiquantumtechnology.springeropen.com/qrng>

<https://www.researchgate.net/publication/308007227> Exploratory Data Analysis

<https://pypi.org/project/quilt>

<https://docs.python.org/3/library/random.html>