Published in AJSE, Vol:22, Issue: 3
Received on 25th July 2023
Revised on 16th October 2023
Accepted on 3rd December 2023

AIUB JOURNAL OF SCIENCE AND ENGINEERING
ISSN: 1608 – 3679 (print) 2520 – 4890 (Online)

Probabilistic Modeling for Conditional Statements

Alaa Ghazi Abdulbaqi, and Yasir Hashim, *Senior Member, IEEE*

***Abstract*—A new mathematical framework is proposed in this study to comprehend the impact of program architecture on input random variables, the IF statement was the main topic. The primary idea that is theoretically and experimentally supported in this study is that the part of the joint pmf of a collection of random variables that represents the condition will be shifted to the part that represents the action. After sorting two random variables, the framework is used with four random variables, and the theoretically produced results were realistically validated. The study's equations can be applied to assessing probabilistic models of various sorting algorithms or other intricate program structures. This may also result in future investigations formalizing more precise execution time expectations.**

***Index Terms*—Sorting Function, Random Variables, Program Structure, Execution Time.**

I. Introduction

THE life cycle of software development includes a crucial activity called software testing. It aids in boosting a developer's confidence that a program accomplishes its goals. To put it another way, we can say that it is a process of running a program with the goal of identifying faults [1-2].

The automotive, avionics, health care, and consumer electronics industries are just a few examples of the application domains for embedded computing systems, in addition to functional accuracy, the safety-critical systems used in the automotive, avionics, and healthcare industries also require good time predictability. For such complex real-time systems with numerous concurrent processes, traditional schedulable analytic techniques can ensure the satisfiability of temporal restrictions [3-4].

The Worst-Case Execution Time (WCET) of each task is a crucial element needed for the schedulability analysis. A task's maximum execution time (WCET) on a target processor is determined by all potential inputs. The contemporary design trend of embedded applications necessitates the analysis and optimization of performance and power in all the system's components, particularly in the early design stages when several solutions are contrasted. The need of a quick examination of the program, performed across the full development/compilation phase, especially from the source level down to the assembly, is being reinforced by the current pervasiveness of microprocessor-based architectures, to this end, the subject of establishing limits on the processing time of a process running on a microprocessor was examined in several research publications [5-7].

Unfortunately, given the capabilities of the present Processor, this work is getting harder and harder. Because the target application is frequently real-time constrained, several analyses approached the issue at a coarse granularity, concentrating primarily on the WCET (Worst Case Execution Time). The issue generally has two sides; the first is analyzing the program routes to find out which instructions will be executed and how they will affect the WCET, while the second is micro-architectural analysis, or modeling of the hardware system running the software [6-7]. The authors in [8] suggested a technique for obtaining probabilistic distributions of execution durations using static analysis. It was assumed that the real-time program in question is broken up into a number of jobs, each of whose source code is known. The suggested approach enables designers to correlate every execution path with an execution time and a probability that it will follow that path, disregarding hardware issues in this study and basing their decisions solely on the source code of the jobs. A source code example was provided to show how the technique works. Similarly, the research in [9] had modeling attempts for the instruction cache. To simulate the evolution of cache content during program execution over a variety of inputs, they developed the concept of probabilistic cache states. The experimental assessment supported the probabilistic cache modeling approach's scalability and correctness.

II. Methodology

A. Mathematical Background

Assuming a discrete random variable X_1 with probability mass function (pmf) $f_1(X_1)$ as [10]:

$$f_1(k) = P(X_1 = k) \qquad (1)$$

Also in order to progress in mathematical evaluation, additional assumption is made by considering the domain of X_1 to be n integers from 0 to (n-1). In this case the pmf of X_1 can be written as [11]:

$$f_1(X_1) = \sum_{k=0}^{n-1} P(X_1 = k)\,\delta(X_1 - k) = \sum_{k=0}^{n-1} f_1(k)\,\delta(X_1 - k) \qquad (2)$$

Where $\delta(k)$ is the unit impulse function defined as [12]:

Alaa Ghazi is with Department of Information Technology, Tishk International University, Kurdistan Region, 44001, Erbil, Iraq. (e-mail: alaa.ghazi@tiu.edu.iq).
Yasir Hashim is with Department of Computer Engineering, Tishk International University, Kurdistan Region, 44001, Erbil, Iraq (e-mail: yasir.hashim@tiu.edu.iq).

$$\delta(k) = \begin{cases} 1 & if\ k = 0 \\ 0 & if\ k \neq 0 \end{cases} \quad (3)$$

In addition to that Cumulative Distribution Function (cdf) is defined as [10]:

$$F_1(k) = P(X_1 \leq k) = \sum_{j=0}^{k} P(X_1 = j) = \sum_{j=0}^{k} f_1(j) \quad (4)$$

The last assumption here is the uniform distribution of all random variables under study with initial independence. Applying this assumption to eq(4), can yield the following equations:

$$f_1(X_1) = \sum_{k=0}^{n-1} \left(\frac{1}{n}\right) \delta(X_1 - k) = \left(\frac{1}{n}\right) \sum_{k=0}^{n-1} \delta(X_1 - k) \quad (5)$$

$$f_1(k) = \left(\frac{1}{n}\right) \quad (6)$$

Also the cdf can be evaluated to be:

$$F_1(k-1) = \sum_{j=0}^{k-1} f_1(j) = \sum_{j=0}^{k-1} \left(\frac{1}{n}\right) = \left(\frac{k}{n}\right) \quad (7)$$

It is required in this research to evaluate theoretically the probability mass function $f_{new1}(X_1)$ after passing the random variable X_1 in if statement and sort function.

B. Simple IF Statement Influence

The first and simplest program structure to be examined is with simple if statement with a condition that matches the value of the random variable with a constant “a” then take an action to assign the constant “b” to X_1 if the condition is True. The code in python is shown in Fig (1). The samplesize refers to number of samples that ae generated for each random variable.

```
for c in range(samplesize):
   if (X[1,c] = =a):
      X[1,c]=b
```

Figure 1 The python code for simple IF statement with constant matching condition.

The argument here is that the occurrence of $X_1 = a$ will be eliminated and its probability will be shifted to $X_1 = b$ so the new pmf $f_{1new}(X_1)$ can be theoretically calculated as:

$$f_{1new}(X_1) = f_1(X_1) - f_1(a)\delta(X_1 - a) + f_1(a)\delta(X_1 - b) = f_1(X_1) - \left(\frac{1}{n}\right)\delta(X_1 - a) + \left(\frac{1}{n}\right)\delta(X_1 - b) \quad (8)$$

From above discussion, the following theorem can be concluded:

Theorem: Applying IF Statement on a single discrete random variable will result in shifting the value of the probability mass function from the condition range to the action range.

C. IF Statement with Less-Than Condition Influence

The next program structure to be examined is an IF statement with a condition that matches the values of the random variable less than a constant “a” then take an action to assign the constant “b” to X_1, if the condition is True. The code in python is shown in Fig (2).

```
for c in range(samplesize):
   if (X[1,c] <a):
      X[1,c]=b
```

Figure 2 The python code for IF statement with less than condition.

The argument here is that the occurrence of $X_1 < a$ will be eliminated and its probability (which is in this case can be expressed by the cdf) will be shifted to $X_1 = b$ so the new pmf $f_{1new}(X_1)$ can be theoretically calculated as below:

$$f_{1new}(X_1) = \begin{cases} 0 & if\ k < a \\ f_1(X_1) + F_1(a-1) & if\ k = b \\ f_1(X_1) & elsewhere \end{cases} \quad (9)$$

Assuming the range from 0 to (n-1) will generate below:

$$f_{1new}(X_1) = f_1(X_1) - \sum_{j=0}^{a-1} f_1(j)\,\delta(X_1 - j) + F_1(a-1)\delta(X_1 - b) \quad (10)$$

After applying the uniform distribution assumption:

$$f_{1new}(X_1) = f_1(X_1) - \left(\frac{1}{n}\right) \sum_{j=0}^{a-1} \delta(X_1 - j) + \left(\frac{a}{n}\right) \delta(X_1 - b) \quad (11)$$

D. Sorting of Two Random Variables

The more complicated program structure to be examined is when IF statement condition consists of two independent random variables X_1 and X_2. The typical example here is when it is required to sort two variables, ie to check if X_1 is larger than X_2 then swap the two values, which results that X_2 to be always greater or equal to X_1.

The python code that can implement this operation is shown in Fig (3). The variable temp1 is used as a swap variable.

```
for c in range(samplesize):
   if (X1[c] > X2[c]):
      temp1=X1[c]
      X1[c] = X2[c]
      X2[c] = temp1
```

Figure 3 The python code for sorting two random variables.

In order to find the individual pmfs of X_1 and X_2 before and after passing through the code, it is needed to resite the laws of joint probability of independent events A and B [13]:

$$P(A \cap B) = P(A)\,P(B) \quad (12)$$

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) = P(A) + P(B) - P(A)\,P(B) \quad (13)$$

Also this can be expanded to multiple d events (where d here is the dimension or the number of random variables):

$$P(A_1 \cap A_2 \cap \ldots \cap A_m) = \prod_{j=1}^{d} P(A_j) \quad (14)$$

$$P(A_1 \cup A_2 \cup \ldots \cup A_m) = \sum_{j=1}^{d} P(A_j) - \prod_{j=1}^{m} P(A_j) \quad (15)$$

In order to calculate the joint pmf of the two independent random variables X_1 and X_2, then from eq(12) and eq(6), it can be conluded that:

$$f_{joint}(k_1,k_2) = f_1(k_1)f_2(k_2) = \left(\frac{1}{n}\right)\left(\frac{1}{n}\right) = \left(\frac{1}{n^2}\right) \quad (16)$$

$$f_{joint}(X_1 = k_1, X_2 = k_2) = \left(\frac{1}{n^2}\right)\sum_{k_1=0}^{n-1}\sum_{k_2=0}^{n-1}\delta(X_1 - k_1)\,\delta(X_2 - k_2)$$

(17) [10]

The theoretical plot for joint pmf described in eq (17) is shown in Fig(4a).

Then to analyze the influence of sorting function, it is needed to find out the joint pmf after passing through the sorting function, then evaluate the marginal pmf of each random variable. The new joint pmf can be derived with the same discussion used in section 2.3 , and by generalizing Theorem 1 to bivariate joint pmf, the joint pmf for the area $X_1 > X_2$ that makes the if condition true will be shifted to the area of the action $X_1 < X_2$, exactly in the revesred sequence of the two variables. This discussion will result into the below equation:

$$f_{joint\ new}(X_1,X_2) = \begin{cases} f_{joint}(X_1,X_2) - f_{joint}(X_1,X_2) = 0 & if\ X_1 > X_2 \\ f_{joint}(X_1,X_2) + f_{joint}(X_2,X_1) & if\ X_1 < X_2 \\ f_{joint}(X_1,X_2) & if\ X_1 = X_2 \end{cases}$$

(18)

Therefore, if the two random variables assume uniform distribution for the range from 0 to (n-1), then new joint pmf resulting from the sorting function can be expressed as:

$$f_{joint\ new}(k_1,k_2) = \begin{cases} \left(\frac{2}{n^2}\right) & if\ k_1 < k_2 \\ \left(\frac{1}{n^2}\right) & if\ k_1 = k_2 \\ 0 & if\ k_1 > k_2 \end{cases} \quad (19)$$

or alternatively:

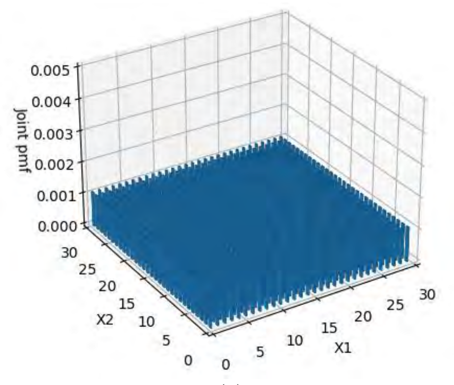
$$f_{joint\ new}(k_1,k_2) = \begin{cases} 0 & if\ k_2 < k_1 \\ \left(\frac{1}{n^2}\right) & if\ k_2 = k_1 \\ \left(\frac{2}{n^2}\right) & if\ k_2 > k_1 \end{cases} \quad (20)$$

Considering the two random independent variables X_1 and X_2 are ranging from 0 to n-1 with uniform pmf, then the plot for this new joint pmf is shown in Fig(4b).

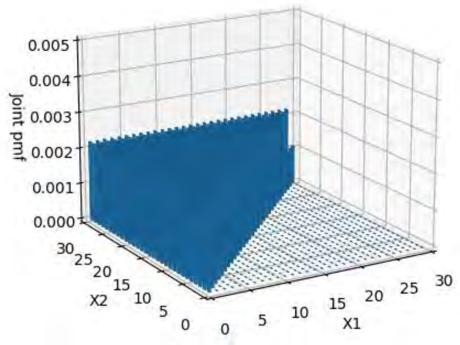
In order to be able to compare the theoretical and practical evaluations of the influence of the various program structures, it is required to evaluate the individual pmf of each random variable. This pmf is called marginal pmf and it is defined in theorem 4.3 from [14] as:

Theorem: For discrete random variable X and Y with joint pmf $P_{X,Y}(x,y)$,

$$P_X(x) = \sum_{y\in S_y} P_{X,Y}(x,y)\ , \qquad P_Y(y) = \sum_{x\in S_x} P_{X,Y}(x,y)$$



(a)



(b)

Figure 4 The joint pmf of two random variables for n=30 (a) before sorting function and (b) after sorting function.

From this theorem and if both variables are ranging from 0 to (n-1), then it can be concluded that:

$$f_1(X_1 = k) = \sum_{j=0}^{n-1} f_{joint}(X_1 = k, X_2 = j) \quad (21)$$

$$f_2(X_2 = k) = \sum_{j=0}^{n-1} f_{joint}(X_1 = j, X_2 = k) \quad (22)$$

So in order to calculate the individual pmfs for both random variables, eq(20) and eq(19) can be substituted in eq(21) and eq(22) respectively:

$$f_{1\ new}(k) = \begin{cases} \sum_{j=0}^{n-1} 0 & if\ j < k \\ \sum_{j=0}^{n-1}\left(\frac{1}{n^2}\right) & if\ j = k \\ \sum_{j=0}^{n-1}\left(\frac{2}{n^2}\right) & if\ j > k \end{cases} \quad (23)$$

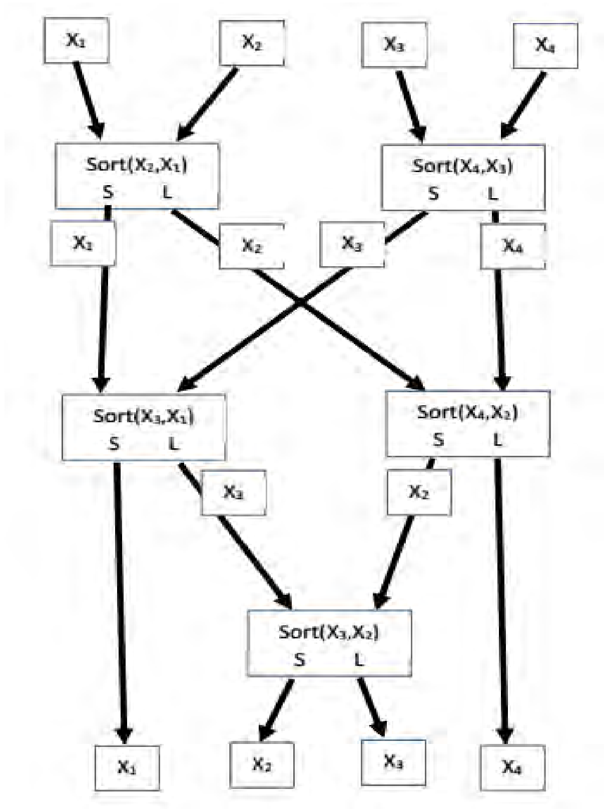
$$f_{1\ new}(k) = \sum_{j=k}^{k}\left(\frac{1}{n^2}\right) + \sum_{j=k+1}^{n-1}\left(\frac{2}{n^2}\right) = \left(\frac{1}{n^2}\right) + \left(\frac{2((n-1)-k)}{n^2}\right) = \left(\frac{2((n-1)-k)+1}{n^2}\right)$$

(24)

$$f_{2\ new}(k) = \begin{cases} \sum_{j=0}^{n-1}\left(\frac{2}{n^2}\right) & if\ j < k \\ \sum_{j=0}^{n-1}\left(\frac{1}{n^2}\right) & if\ j = k \\ \sum_{j=0}^{n-1} 0 & if\ j > k \end{cases} \quad (25)$$

$$f_{2\ new}(k) = \sum_{j=0}^{k-1}\left(\frac{2}{n^2}\right) + \sum_{j=k}^{k}\left(\frac{1}{n^2}\right) = \left(\frac{2(k-1)}{n^2}\right) + \left(\frac{1}{n^2}\right) = \left(\frac{2k+1}{n^2}\right)$$

(26)



(a)

```
def SortRV(L,S)
#L=Large Variable after sort
#S=Small Variable after sort
   for c in range(samplesize):
      if (X[S,c] > X[L,c] ):
          temp1=X[S,c]
          X[S,c]  = X[L,c]
          X[L,c]  = temp1
     Calculate_New_PMFs() #Theoritically
     Measure_New_PMFs() # Practically
     Plot_New_PMFs() #Both Theoritical and
Practical
#Code Stage 1
SortRV(2,1)
SortRV(4,3)
#Code Stage 2
SortRV(3,1)
SortRV(4,2)
#Code Stage 3
SortRV(3,2)
```

(b)

Figure 5 (a) The algorithm for sorting four random variables (b) Python code for sorting four random variables.

E. Sorting Code for Four Random Variables

Further advance in this study is the discussion of the algorithm to sort four independent random variables X_1 , X_2 , X_3 and X_4 . In this case five calls to sorting function should be used as illustrated in the algorithm shown in Fig (5a) and the Python code shown in Fig (5b). The results will satisfy the mathematical inequality:

$(X_1 < X_2 < X_3 < X_4)$

Using the same argument used in section (2.4) for multiple d independent uniformly distributed random variables, it can be concluded that:

$$f_{joint}\,(k_1, \dots, k_d) = \prod_{j=1}^{d} f_j\,(k_j) \;\; = \prod_{j=1}^{d} \left(\frac{1}{n}\right) \; = \; \left(\frac{1}{n^d}\right)$$
(27)

$$f_{joint}\,(X_1 = k_1, \dots, X_d = k_d) = \left(\frac{1}{n^d}\right) \sum_{k_1=0}^{n-1} \dots \sum_{k_d=0}^{n-1} \delta\,(X_1 - k_1) \dots \delta\,(X_d - k_d)$$
(28)

Eq(27) represents the initial values that can be assigned to the joint pmf before starting the experiment with d random variables. The joint pmf here is defined as d dimensional matrix.

Then to analyze the influence of sorting code of the four variables, it is needed to find out the joint pmf after passing through single sorting function, then evaluate the marginal pmf of each individual random variable. The new joint pmf can be derived with the same discussion used in section 2.3, and by generalizing Theorem 1 to multivariate joint pmf, the joint pmf for the d-dimensional region $X_S > X_L$ that makes the IF condition true will be shifted to the d-dimensional region of the action $X_S < X_L$, exactly in the reversed sequence of the two variables. This discussion will result into the below equation:

$$f_{joint\ new}\,(X_1, \dots, X_S, \dots, X_L, \dots, X_d) = \begin{cases} 0 & \text{if } X_S > X_L \\ f_{joint}\,(X_1, \dots, X_S, \dots, X_L, \dots, X_d) + f_{joint}\,(X_1, \dots, X_L, \dots, X_S, \dots, X_d) & \text{if } X_S < X_L \\ f_{joint}\,(X_1, \dots, X_S, \dots, X_L, \dots, X_d) & \text{if } X_S = X_L \end{cases}$$
(29)

$$f_k\,(X_k = k) = \sum_{j_1=0}^{n-1} \dots \sum_{j_k=k}^{k} \dots \sum_{j_d=0}^{n-1} f_{joint}\,(j_1, \dots, j_d)$$
(30)

Eq (29) and eq (30) are the final theoretical conclusion of this study, on which they can be utilized to calculate the new joint pmf each time the sorting function is used to sort two random variables irrelevant of the sorting stage. In results section trials will be done to validate practically those equations which should work even with the existence of dependency between the two sorted variables after passing multi-tier sorting stages.

F. Procedures for the Measurement of Practical pmf

In order to implement those tests, a samplesize=1000000 is used to generate different values of each random variable under study and store them into a two-dimensional matrix [d, samplesize]. This matrix is considered to be the input of the experiment. The practical pmf of each variable is calculated by counting the occurrence of each value and dividing it by the sample size. Plot for both theoretical and practical results are combined for easy comparison. After passing each program structure a new pmf is evaluated theoretically using the relative mathematical equation and practically by counting the occurrence of the values in the range 0 to (n-1) in the same matrix. Again, unified plot is made after each stage. The summary of the methodology was included in the Fig (6).

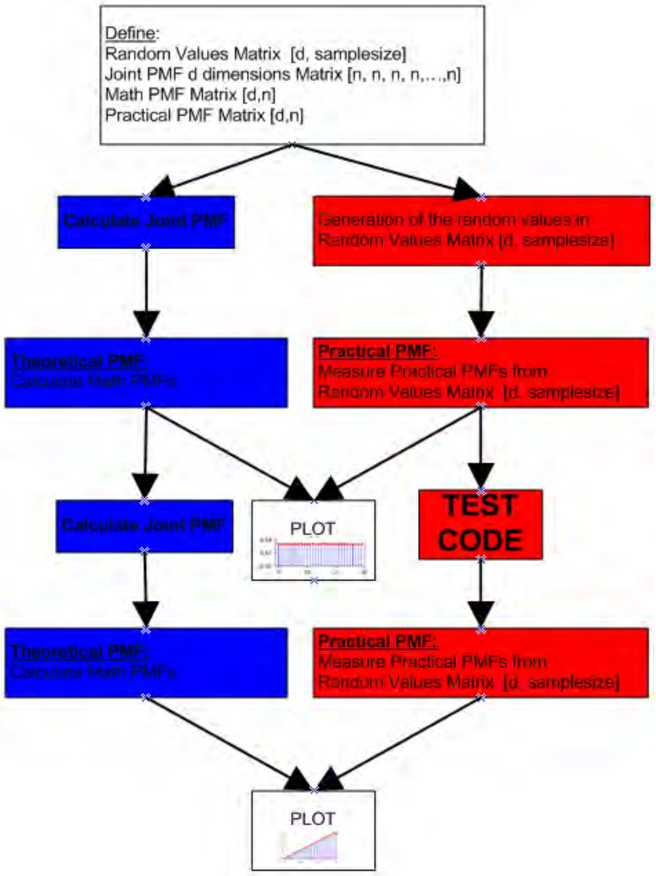

Define:
Random Values Matrix [d, samplesize]
Joint PMF d dimensions Matrix [n, n, n, n,...,n]
Math PMF Matrix [d,n]
Practical PMF Matrix [d,n]
Calculate Joint PMF
Generation of the random values in Random Values Matrix [d, samplesize]
Theoretical PMF: Calculate Math PMFs
Practical PMF: Measure Practical PMFs from Random Values Matrix [d, samplesize]
Calculate Joint PMF
PLOT
TEST CODE
Theoretical PMF: Calculate Math PMFs
Practical PMF: Measure Practical PMFs from Random Values Matrix [d, samplesize]
PLOT

Figure 6 Summary of the study methodology.

III. Results

In this section a comparison and validation of the plot of the theoretical equations obtained in the Methodology sections will be conducted against practical measurement of individual pmfs.

A. Simple IF Statement with Equality Condition Results

The results of passing a single random variable through simple IF statement with equality condition for the code depicted in Fig (1) with value of a=3 and b=10 is shown in Fig(7). As indicated in Theorem 1, the pmf value of the condition range $(X_1 = 3)$ is accumulated over the pmf of the action range $(X_1 = 10)$. In the plotted results indicate practical measurement matches theoretical evaluations.

B. IF Statement with Less than Operation Results

Continuing in the same discussion, the results of passing a single random variable through an IF statement with less than condition for the code of Fig(2) with value of a=3 and b=10 is shown in Fig(8). In this case the pmf values of the condition range $(X_1 < 3)$ which can be represented by cdf $F_1(2)$ is added over the pmf of the action range $(X_1 = 10)$. Again, the plotted results indicate perfect matching between practical measurement matches theoretical evaluations.

C. Results for Sorting Two Variables

The experiment of sorting two variables is more complex than a conditional statement with single variable since the study interaction between the two variables requires analyzing the joint pmf of the two variables before and after the sorting code. In section 2.4 an argument was made for the calculation of the joint pmf and from it the equation for the marginal pmfs where derived in eq (24) and eq (26). Both equations were plotted in Fig (9b). The plot shows excellent matching with practical pmfs obtained by counting the values after passing through the sorting code, which validates the joint pmf calculation along with marginal pmfs derived from it. The behavior of the curves is fitting the rule of each variable so the pmf of the small variable X_1 is tending towrd the minimum value 0, while the pmf of the large variable X_2 is tending toward the maximum (n-1). From other side, it can be noticed that the sorting of two variables has produced two mirrored pmfs around the Y-axis.

D. Results of Sorting Code for Four Random Variables

In Fig (10) the results of sorting two sets of two random variables carry no additional news out of the sorting of two variables discussed in section 3.3 After passing the four random variables in Code Stage 2 where X_3 and X_1 are sorted so the output will force $X_3 > X_1$. In this case X_1 will be assured to hold the least value among the four variables as it was illustrated in Fig(5a). The mathematical evaluation of the pmfs of X_1 and X_3 is based on repeated use of eq (29) and eq (30). Similar discussion can be conducted for the sorting X_4 and X_2 so the output will force $X_4 > X_2$, forcing X_4 to hold the greatest value among the four variables as it was illustrated in the algorithm in Fig(5a) also. The results of Code Stage 2 are show in Fig (11). The final sorting stage is done between X_3 and X_2 to assure that $X_3 > X_2$. The results are shown in Fig (12). Fig (12) will repeate with any kind of additional sorting is forced between any pair of the random variables which indicates that this Figure represents the last results and no further sorting can be done. The graphs depict mirrored image of each other like the discussion in section 3.3.

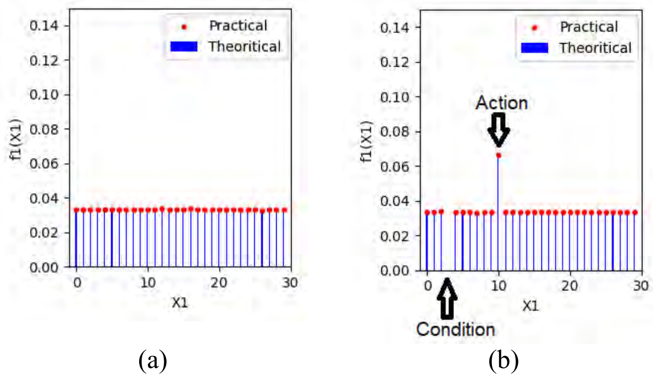
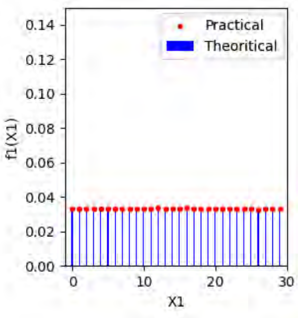
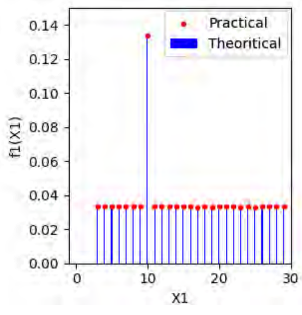


Figure 7 The pmf of X_1(a) before IF statement with equality condition and n=30 (b) after the IF statement

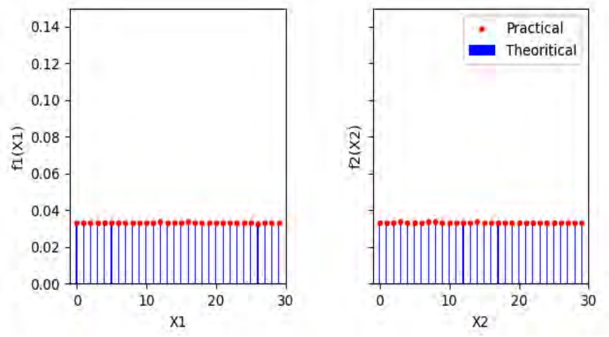


(a)

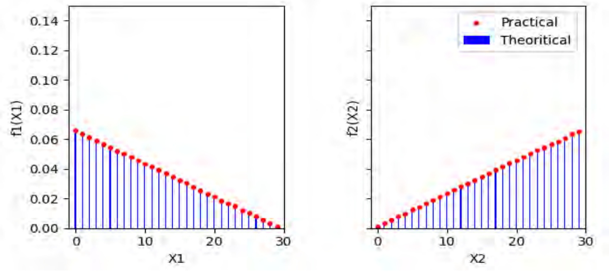


(b)

Figure 8 The pmf of X_1(a) before IF statement with less than condition (b) after the IF statement

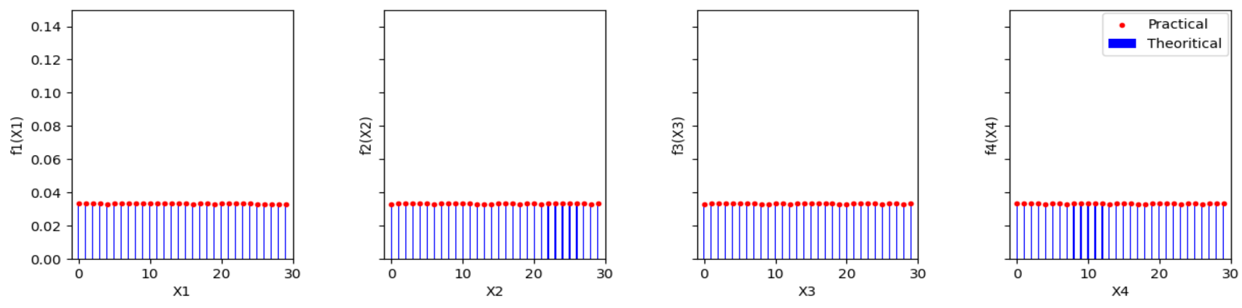


(a)

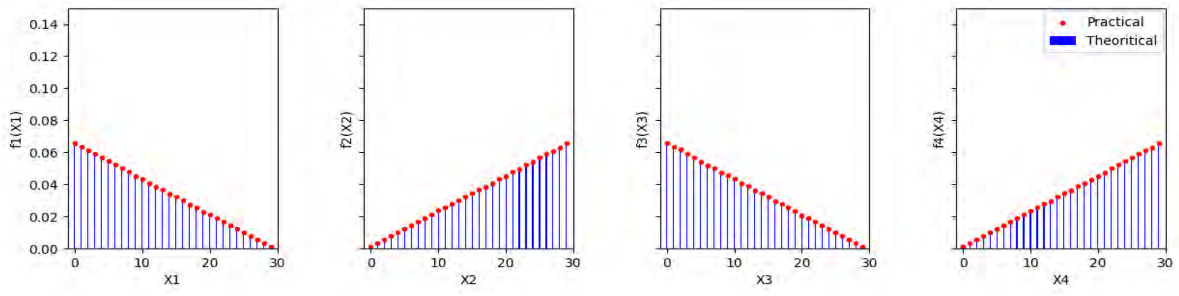


(b)

Figure 9The pmf of the two random variables for n=30 (a) before the sorting code (b) after sorting code.



(a)



(b)

Figure 10. Results of Sorting Four variables (a) Initial input pmfs (b) pmfs after Code Stage 1: SortRV(2,1), SortRV(4,3).

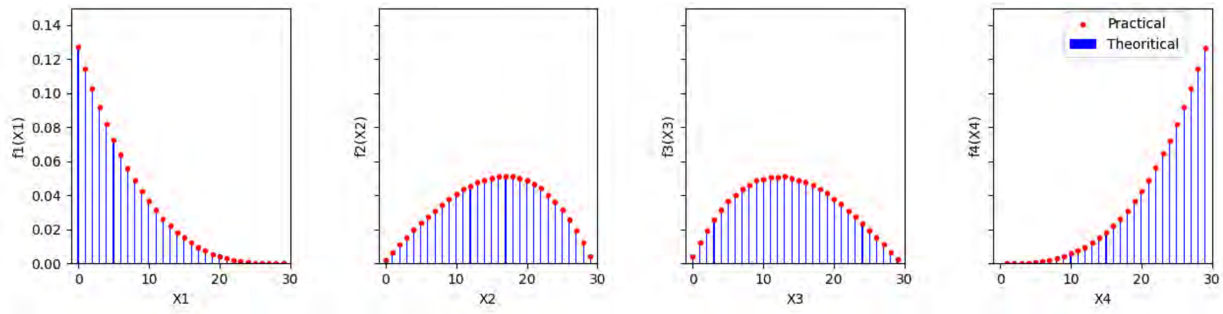


Figure 11 Results of Sorting Four variables (a) Initial input pmfs (b) pmfs after Code Stage 2: SortRV(3,1), SortRV(4,2)

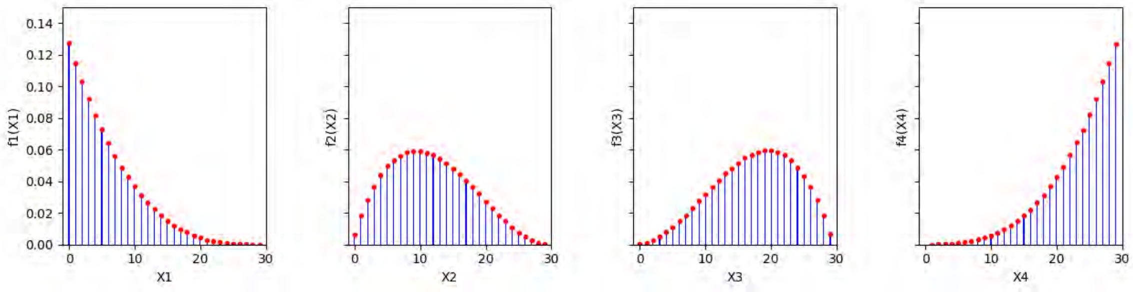


Figure 12 Results of Sorting Four variables (a) Initial input pmfs (b) pmfs after Code Stage 3: SortRV(3,2).

IV. CONCLUSION

This research suggests a new mathematical framework to understand the influence of the program structures on input random variables. The focus was on IF statement. The main principle validated in this paper theoretically and practically is that the portion of joint PMF of set of random variables that represents the condition will be shifted and added with portion that stands for the action. The framework is applied on sorting two random variables and then on four random variables and the theoretical obtained results were validated practically. The equations derived in this study can be useful in evaluating probabilistic model of various sorting algorithms or other complex program structures. Additionally, this can lead in formalizing more exact execution time expectation in future studies

ACKNOWLEDGMENT

The authors would like to thank Tishk International University (TIU) for their support.

REFERENCES

[1] John L., McCallum A., and Fernando C. N., "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data", 18th International Conference on Machine Learning 2001 (ICML 2001), pages 282-289.

[2] Kaufmann S., "Conditional Predictions". Linguistics and Philosophy volume 28, pages181–231 (2005). https://doi.org/10.1007/s10988-005-3731-9

[3] George P., Eric N., Danilo J. R., Shakir M., Balaji L., "Normalizing Flows for Probabilistic Modeling and Inference", Journal of Machine Learning Research, 22(57):1-64, 2021.

[4] A. Arnab, S. Zheng, S. Jayasumana, B. Romera-Paredes, M. Larsson, A. Kirillov, B. Savchynsk, C. Rother, F. Kahl, P. H.S. Torr, "Conditional Random Fields Meet Deep Neural Networks for Semantic Segmentation: Combining Probabilistic Graphical Models with Deep Learning for Structured Prediction," in IEEE Signal Processing Magazine, vol. 35, no. 1, pp. 37-52, Jan. 2018, doi: 10.1109/MSP.2017.2762355.

[5] J. Xu, G. Chen, N. Zhou, W. -S. Zheng and J. Lu, "Probabilistic Temporal Modeling for Unintentional Action Localization," in IEEE Transactions on Image Processing, vol. 31, pp. 3081-3094, 2022, doi: 10.1109/TIP.2022.3163544.

[6] Brückler, F. M., & Milin Šipuš, Ž. (2023). "Pre-service mathematics teachers' understanding of conditional probability in the context of the COVID-19 pandemic". European Journal of Science and Mathematics Education, 11(1), 89-104. https://doi.org/10.30935/scimath/12436

[7] R. Marculescu, D. Marculescu, M. Pedram, "Probabilistic Modeling of Dependencies During Switching Activity Analysis", IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 17, NO. 2, pp. 73-83, 1998.

[8] David L, Puaut I. "Static determination of probabilistic execution times." InProceedings. 16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004. 2004 Jul 2 (pp. 223-230).

[9] Liang Y, Mitra T. "Cache modeling in probabilistic execution time analysis." InProceedings of the 45th annual Design Automation Conference 2008 Jun 8 (pp. 319-324).

[10] Miller, S. and Childers, D., 2012. Probability and random processes: With applications to signal processing and communications. Academic Press.

[11] Chakraborty S., "Some Applications of Dirac's Delta Function in Statistics for More Than One Random Variable," Applications and Applied Mathematics: An International Journal (AAM), Vol. 3, No. 1, Article 4, pp. 42 – 54, 2008.

[12] Alan V. O., Alan S. W., Nawab S. H. "Signals and systems." 2nd edition, Upper Saddle River, NJ: Prentice hall, 1997.

[13] Grimmett G, Stirzaker D. "Probability and random processes." 4th edition, Oxford university press; 2020.

[14] Hwei P. H. , “Theory and problems of probability, random variables, and random processes”, 1st edition, Schaum's Outline Series, McGraw-Hill, 1997.

Alaa Ghazi Abdulbaqi received his B.Sc. and M.Sc. in Computer Engineering from Al Nahrain University, Baghdad, Iraq, in 1995 and 1998. He is currently an Assistant Lecturer in the Department of Information Technology, Faculty of Science, Tishk International University, Erbil-Kurdistan, Iraq

Yasir Hashim received the B.Sc. and M.Sc. in Electronic Engineering from University of Mosul, Mosul, Iraq, in 1991 and 1995 respectively. He completed his Ph.D. in Electronics Engineering-Micro and Nano electronics from Universiti Science Malaysia (USM), Penang, Malaysia, in 2013. He is currently Lecturer in the Department of Computer Engineering, Faculty of Engineering, Tishk University, Erbil-Kurdistan, Iraq.