Power Grid Reliability

Monte Davityan[1], Celia Langello[2] and Cassandra Ganska[3]

[1]California State University, Fullerton.
[2]Colorado Mesa University.
[3] New Mexico State University.

Contributing authors: monte@csu.fullerton.edu; celangello@mavs.coloradomesa.edu; cganska@nmsu.edu;

Abstract

We explore differing techniques and algorithms to attempt to efficiently compute the probability mass function of an arbitrary power grid. We benchmark these methods and conclude with an efficient algorithm to model for independent power grid forks. We also explore the results and run experiments exploring the relationship between forks and branches, and component aging on the power grid.

1 Introduction

Recently there has been an increase in demand for electricity. This is due to the rise of electrical machines and the need for a continuous flow of electricity. This is widely shown during the COVID-19 pandemic where there was a need for reliability in the power grid for hospitals to run and citizens to transition to at home work [1]. This research may even be effective with utility companies who need to follow certain regulations that are in place by the government such as the ones in the United States [2].

This study began with the main objective to create an efficient algorithm that would compute the probabilities of customers with power. The component being examined is composed of the main power source, forks and branches. The fork and branches have their own probability of success.

While also considering that the electrical power grid infrastructure in the United States contains multiple components who have surpassed their 50 year life expectancy [3], we linked the probability of success to the age of the component by implementing a Weibull aging model (See Section 8).

The rest of the paper was organized with Section 2 as a related works. This includes works that were read in order to help us move further along and a description of the issues we came across. Section 3 will begin the topic of the issues. The process of how the probabilities are computed are discussed in Section 4. Next, the three methods are split up as follows, Section 5 talks about the recursive method, Section 6 goes over the moment generating function method, and Section 7 covers the probability generating function method. Section 8 will cover the failure rates that are dependent on time. The results will be in Section 9 on the methods and the final results. Last, Section 10 will be the conclusion.

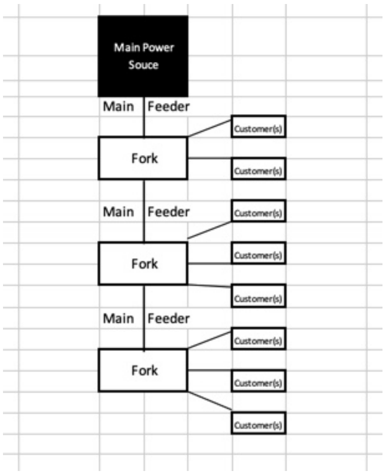


Fig. 1 An example of a simple Power Grid

In order to begin, we first needed a model to follow, which can be seen in figure 1. The main power source is where the electricity is generated, the forks are the transformers, and the customers are where the electricity is being delivered. It is assumed that the main power source and the feeders never fail and that the forks are independent of each other. In order to compute the probabilities, all possible combinations of customers with power must be computed. This topic and process is further explained in Section 3 where an example is given. Next an effective way to handle the data computed is required. This part is crucial due to the large quantity of data that was returned from all the possible combinations. After successfully implementing that, the probabilities were able to be computed.

We explored three methods of computing our probabilities which will be discussed further in detail in Section 5, 6, and 7. All methods were successful in retrieving the probabilities. However, the first method has a high usage of memory which resulted in a high computational time. The second method, was successful in being able to handle more combinations and eventually ran out of memory location. With editing the equation and using a r package we resulted with the final improvement, the third method. Just to note, the previous methods were modified into the newer methods. While we did not stay with the first methods, there are parts of the code that carried on towards the final production.

2 Related Works

We began with the study of implementing Osario and Rojo recursive method as our base (See Section 5). Which leads us to running into the first big issue, storing every combination which made it inefficient in both time and memory. The insufficient memory allocation arose from the fact that the program was saving every possible combination and would run out of RAM. This led us to the study of D.Evans et al, which would talk about using the moment generation function. D. Evans et al. introduces a convolution implementation utilizing efficient algorithmic techniques to calculate the sum of two independent random variables, which can and is adapted to this power grid problem (See Section 7) [4]. In doing so, we were able to cut down on the RAM that was being used.

With that problem now existing, we began researching and found our second method that allowed us to implement less storage which resulted in the ability to add more forks and branches to our power grid. While the second method was a big improvement, it had its own issues. There would eventually the same issue would arise, running out of memory storage (See Section 6) due to the capabilities of ”expand.grid”. Which would be solved by using the probability generating function (See Section 7). By now using the ”polynom” package in R, the program was able to run n values without running into memory storage problems (See Section 8).

With having a stable program, we began looking into including differing probabilities for the components in regards to time. This lead to a Weibull aging model that was discussed by Rodioniv et al. in ”Models and data used for assessing the ageing of systems, structures and components”.

Other approaches that have been used to evaluating the probability mass function of an arbitrary power grid is through simulation techniques, due to the vast number of combinations taken into account when directly calculating the probability mass function. Marsadek et al. used a Monte Carlo simulation to approximate the reliability of the power grid through the evaluation of the failure points [5]. Chun-Lien Su in comparison to the Monte Carlo technique used a point-estimate method introduced in the paper, to calculate the moments of the power grid [6]. A different direct approach to calculating the distribution relies on taking advantage of the networks topology and observing the network to be a Barabasi-Albert network model [7]. Chassin et al. utilize this Barabasi- Albert model to evaluate a scale-free model and estimate the reliability of the power grid [8]. However, we wanted to stay away from a simulation that would be based off of random numbers. Instead, we wanted to write a program that can have values inputted and return the probability of success.

3 Methods

In an attempt to create a computationally efficient algorithm to calculate the probability mass function of the power grid (assuming independence of the forks and assuming the main power line nor the feeders fail), we utilized three methods each with differing degrees of success. The first method as described in Section 5 utilizes the recursive properties of the independent forks to recursively calculate the probability mass function. The second method as explained in Section 6 uses the properties of moment generating functions to obtain the coefficient combinations and determine the moment generating function of the entire power grid along with the associated probability mass function. The final third method, as described in Section 7, builds on the moment generating function method but transforms the exponentials into a polynomial of probability generating functions.

4 One Fork Computation

For purposes of future reference and understanding, in this section we will briefly discuss the method used to calculate the probability mass function of one fork. This method is repeatedly used because of the assumed independence between forks and thus serves as an important backbone to the methods discussed in Sections 5, 6, and 7.
Suppose we have a fork X with three branches with one, two and three [1, 2, 3] customers respectively. Also suppose the associated probability of failure of the respective components are [0.07, 0.07, 0.07] (though they can differ and be dependent on time (See Section 8)). Since there are multiple combinations to the branches, the total possible values of X are [0, 1, 2, 3, 4, 5, 6]. This is obtained by combining all possible combinations. For example, there is the possibility of there being no customers with power in the fork; thus, we have a zero. There is also the possibility of their being six customers with power because we can have all three branches receiving power (1 + 2 + 3). The way we find these combinations is by using the "gtools" package in R [9] which contains the "combn" function. Once these combinations are found, we access the location (index) these combinations were at. For example if the combination we are looking at is four, we get the locations that allow for four to be one combination (in this example it would be indexes 1 and 3 since at index 1 the number 1 exists and at index 3 the number 3 exists, and 1 + 3 = 4). Once these indexes are located, the associated vector of the probability of failure is made into the probability of success at the particular indexes found. So the associated probability of failure ([0.07, 0.07, 0.07]) becomes [0.93, 0.07, 0.93], since we alter the positions at 1 and 3 (since those are the combinations positions that make four). This probability vectors components are then

multiplied to obtain the probability at that specific combination. Since, we also assign probabilities of failure to the transformers, we multiply the result by the transforms probability of success (except for the zero customers with power combination). Duplicate combination probabilities are added together due to the property of independence. For example, we notice in this example that there are two ways to get three customers with power ([3] and [1 + 2]); thus, their individual probabilities are added together. For the above example, we expect a probability mass function for the fork of:

CWP	Operation	Result
0	0.07 + (0.93*prod(c(0.07, 0.07, 0.07))	0.07031899
1	0.93*prod(c(0.93, 0.07, 0.07))	0.00423801
2	0.93*prod(c(0.07, 0.93, 0.07))	0.00423801
3	0.93*prod(c(0.07, 0.07, 0.93))	0.00423801
3	0.93*prod(c(0.93, 0.93, 0.07))	0.05630499
4	0.93*prod(c(0.93, 0.07, 0.93))	0.05630499
5	0.93*prod(c(0.07, 0.93, 0.93))	0.05630499
6	0.93*prod(c(0.93, 0.93, 0.93))	0.74805201

Notice, zero customers with power has an additional addition due to the double combination of either the transformer failing (with a probability of 0.07 for this example) and thus the entire fork failing, or the transformer succeeding but all the branch's failing (thus the multiplication of the entire vector of failures).

5 Recursive Computations

The first method we employed as introduced in Section 3 utilizes the recursive properties of the power grid, as introduced in Osario and Rojo, to determine the probability mass function of the customers with power in the power grid [10]. We base the recursive computations off of this formula:

$$P(Z = z) = P(X + Y = z)$$

$$= \sum_{k=0}^{z} P(X = k, Y = z - k)$$

$$= \sum_{k=0}^{z} P(X = k)P(Y = z - k)$$

due to independence (assuming the main power line doesn't fail). The recursive nature of the problem makes itself known for greater than two forks. For example: Suppose we have three forks X, Y and Z. To find the combinations and thus the probability mass function of all three forks, we require X + Y + Z.

$$P(X + Y + Z = w) = \sum_{k=0}^{w} P(X + Y = w - z)P(Z = w)$$

if we let w - z = d, we notice $P(X+Y = w-z)$ becomes $P(X+Y = d)$ which we notice is the two fork formula from above. Thus, the formula for three forks becomes:

$$P(X+Y+Z=w)=\sum_{k=0}^{w}[\sum_{k=0}^{w-z}P(X=k)P(Y=(w-z)-k)]P(Z=w)$$

This process is similarly generalized to more forks.
Another way of formulating this problem is recursively as examined in [4]. Suppose we have n forks each with distribution X_i for $i=1,2,...,n$. And let $Z_n = X_1 + X_2 + ... + X_n$ where X_i is independent for all X. Then,

$$Z_i = Z_{i-1} + X_i$$

for i = 2, 3, ..., n.
As reviewed in D. Evans et. al, this recursive computation becomes costly for diverse and variable values of customers and increases exponentially in computational time and space for an increasing number of branches and forks. One reason this occurs is because of the vast invalid combinations that are generated. For large, diverse values of customers; w - z, generates many values (some negative) that are not possible values of the forks, and thus have probabilities of zero. Therefore, this constant check to see if w - z is a valid combination creates large inefficiencies for particularly larger combinations. Thus, this recursive method proves to be an inefficient method of computing the probability mass function of the power grid. An implementation in R, yields a maximum of six forks (with an average branch size of five branches), $(X_1, X_2, X_3, X_4, X_5$ and $X_6)$ before running out of memory. However, for six forks the time spent to run is ineffective and impractical for real life power systems (see Section 9 for a summary of the results).

6 Moment Generating Function Computations

The second method we employed as introduced in Section 3 utilizes the methods of Moment Generating Functions and the property of independence for discrete random variables. As introduced in D. Evans et. al, suppose we have X and Y as random variables [4]. For the purposes of our paper, X and Y represent independent forks in the power grid with discrete, positive valued customers. We want all combinations of X and Y, thus we want X + Y. Since, for our purposes X and Y are independent (assuming the main power line doesn't fail), the moment generating function of X + Y is:

$$M_{X+Y}(t) = E[e^{t(X+Y)}]$$

$$= E[e^{tX}e^{tY}]$$
$$= E[e^{tX}]E[e^{tY}]$$
$$= M_X(t)M_Y(t)$$

This process can be shown in an example:
Suppose X has two branches, the first containing one customer and the second containing two customers. Thus, the valid combinations of X are 0, 1, 2, 3 (since there is the possibility of zero customers receiving power and similarly of three customers (1 + 2) receiving power. Also suppose the probability of failure for the two branch's and the transformer connecting the branch's to the main feeder is 0.07 (although this probability can differ and can also be dependent on numerous factors such as time as discussed in Section 8.
Similarly, suppose Y has one branch of three customers; thus, the valid combinations for Y are 0, 3. Similarly, the probability of the branch and transformer failing is 0.07.

The moment generating function of X is:

$$M_X(t) = 0.074557 + 0.060543e^t + 0.060543e^{2t} + 0.804357e^{3t}$$

The moment generating function of Y is:

$$M_Y(t) = 0.1351 + 0.8649^{3t}$$

It can be seen that the valid combinations of X + Y become 0, 1, 2, 3, 4, 5, 6 Thus, the moment generating function of X + Y is:

$$M_{X+Y}(t) = 0.010072651 + 0.008179359e^t + 0.008179359e^{2t}$$

$$+0.173152980e^{3t} + 0.052363641e^{4t} + 0.052363641e^{5t} + 0.695688369e^{6t}$$

We obtain the coefficients by multiplying the combinations. For example: we obtain 0.010072651 by multiplying the coefficient of $M_X(t)$ at e^{0t} by the coefficient of $M_Y(t)$ at e^{0t}. We similarly obtain the exponent coefficient by adding the exponents. For example: 0.010072651 has exponent coefficient e^{0t} because we add e^{0t} from $M_X(t)$ to e^{0t} from $M_Y(t)$ to get e^{0t}. We can then obtain the probability mass function of X + Y. The coefficients are the probabilities with the associated combinations being the exponent coefficients.
This method of obtaining X + Y can be generalized to n forks. The generalized form would be:

$$M_{\sum_{i=1}^{n} X_i}(t) = \prod_{i=1}^{n} M_{X_i}(t)$$

where X_i is the ith fork in the power grid.
Mathematically the generalized form looks concise and easy to determine; however, computationally multiplying exponential polynomials is expensive. Thus, we employ a method of determining only the probability and valid input combinations of the final moment generating function, and thus bypass the need to multiply exponential polynomials.
After computing and storing the individual combinations and associated probabilities of X_i we use the "expand.grid" function in R which obtains all the combinations of all the inputted forks [11, 12]. This is done separately for the combinations and the associated probabilities. Finally, we sum up the rows of

the generated combinations to obtain the final moment generating function, exponent coefficients. And, multiply the rows of the generated probability combinations to obtain the final coefficients of the moment generating function.
In reference to the example above, the individual combinations for X and Y are themselves stored in a list and would be combos = [c(0, 1, 2, 3), c(0, 3)] and similarly the probabilities: probs = [c(0.074557, 0.060543, 0.060543, 0.804357), c(0.1351, 0.8649)]. "expand.grid" is called on combos, then the rows are summed which returns:

X	Y	Sum
0	0	0
0	1	1
0	2	2
0	3	3
3	0	3
3	1	4
3	2	5
3	3	6

Then, "expand.grid" is called on probs, but the rows are multiplied which returns:

X	Y	Multiples
0.1351	0.074557	0.01007265
0.1351	0.060543	0.008179359
0.1351	0.060543	0.008179359
0.1351	0.804357	0.1086686
0.8649	0.074557	0.06448435
0.8649	0.060543	0.008179359
0.8649	0.060543	0.008179359
0.8649	0.804357	0.6956884

It can be seen that the probabilities in the multiples when added, add to one which implies a valid probability mass function. We then associate each row of the multiples to the rows of the rows of the summed combinations. With this association, we generate the coefficients of the exponential polynomial without the need to multiply the exponential polynomials (a computationally intensive process). This method also removes the need to check for invalid values in the forks and thus the need for storing ever expanding zero probabilities is eliminated.
While this method is relatively fast (See Section 9), expanding to more than eight forks possesses computational challenges due to "expand.grid" and R running out of memory. To address this problem, we split up the forks, into a maximum of eight pieces (since "expand.grid" can handle up to eight variables/ forks). We calculate these individual subset moment generating functions using the method explained above, and finally we calculate the final moment generating function using the subset moment generating functions.

An example: Suppose we have twenty forks $A_1, A_2, ..., A_{20}$. Then,

$$M_{\sum_{i=1}^{20}}(t) = \prod_{i=1}^{20} M_{A_i}(t)$$

. However, using "expand.grid" on twenty variables/forks (eg. expand.grid($A_1, A_2, ..., A_{20}$) would fail due to R running out of memory. Therefore, we split up the forks into eight parts. The first subset is the moment generating function of $A_1, A_2, ..., A_8$ and thus would be found by expand.grid($A_1, A_2, ..., A_8$) and respectively summing and multiplying to find the coefficients and exponential coefficients. Similarly, the second subset would be the moment generating functions of $A_9, A_{10}, ..., A_{16}$ and the third and final subset would be from $A_{17}, A_{18}, A_{19}, A_{20}$. Finally, we find the moment generating function using these subsets.
Mathematically, this is equivalent to:

$$M_{\sum_{i=1}^{8}}(t) = \prod_{i=1}^{8} M_{A_i}(t)$$

$$M_{\sum_{i=9}^{16}}(t) = \prod_{i=9}^{16} M_{A_i}(t)$$

$$M_{\sum_{i=17}^{20}}(t) = \prod_{i=17}^{20} M_{A_i}(t)$$

and finally using the subsets:

$$M_{\sum_{i=1}^{20}}(t) = M_{\sum_{i=1}^{8}}(t) M_{\sum_{i=9}^{16}}(t) M_{\sum_{i=17}^{20}}(t)$$

Using this method, we can run twenty forks for diverse, variable customer combinations in a relatively fast period, less than ten minutes (See Section 9. However, for greater than twenty forks, the memory problems associated with "expand.grid" still persist.

7 Probability Generating Functions Computations

The third and final method we employed follows the methodology of the Moment Generating Functions method from Section 6 but with the utilization of probability generating functions. The reason in Section 6 we utilized "expand.grid" to find the combinations (and which as well led to the bottleneck of our algorithm, due to the memory issues associated with "expand.grid") was because multiplying exponential polynomials would be far more computationally costly.

For example, for the two fork power grid we have:

$$M_{X+Y}(t) = M_X(t)M_Y(t)$$

$= (p_{1,1}e^{c_{1,1}t} + p_{2,1}e^{c_{2,1}t} + ... + p_{n,1}e^{c_{n,1}t})(p_{1,2}e^{c_{1,2}t} + p_{2,2}e^{c_{2,2}t} + ... + p_{m,2}e^{c_{m,2}t})$

With no efficient way to multiply these two exponential polynomials we reverted to finding the combinations using "expand.grid" as explained in Section 6.

However, if we set $s = e^t$, our exponential polynomial, becomes:

$$= (p_{1,1}s^{c_{1,1}} + p_{2,1}s^{c_{2,1}} + ... + p_{n,1}s^{c_{n,1}})(p_{1,2}s^{c_{1,2}} + p_{2,2}s^{c_{2,2}} + ... + p_{m,2}s^{c_{m,2}})$$

Now using this method of setting $s = e^t$ and obtaining the polynomial generating functions, allows us to efficiently multiply these polynomials using the R package "polynom" [13]. The algorithm (See Appendix A) comprises of obtaining the individual probability mass functions (as explained in Section 4) of the forks (similar to Section 6). However, unlike the moment generating function method, each individual probability mass function is only calculated for each fork, then forgotten. This helps in avoiding memory issues. Once, the individual forks probability mass function is calculated, it is converted into the s- polynomial as seen above. p_i is the probabilities of the probability mass function of the individual fork and c_i is the associated combination (ie. the number of customers with power). This converted polynomial is multiplied by the main polynomial (which is initially set to one). This is done for all the forks with the final result being the polynomial multiplied by the n forks. A general overview of how this works is shown:

$$finalPolynomial = 1$$

$$finalPolynomial = finalPolynomial * individualPolynomial_1$$
$$finalPolynomial = finalPolynomial * individualPolynomial_2$$
$$= finalPolynomial * individualPolynomial_1 * individualPolynomial_2$$

with this process repeated for n forks. The results are summarized in Section 9 but this process allows for hundreds of forks to be run in a manner of minutes, which most closely resembles the computational requirements of real world power grids (although as mentioned before the assumptions made to grant this computational prowess are independence between the forks with the main power grid not failing, which doesn't represent real world power grids for the most part).

D. Evans et al. touch on the use of this technique as a valid but potentially expensive approach for determining the distributions of the discrete random variables for arbitrary, infinite supports. However, for applications of the power grid, the supports for the forks are positive, integers which reasonably lie below one thousand (since it is unreasonable and potentially impossible for power companies to use one transformer (one branch) to supply a thousand customers or more. Thus, with these finite, positive, integer valued supports; this polynomial algorithm proves to be quite effective in computing the probability mass function for the independent power grid. The further experiments (in Section 9 utilize this polynomial- probability generating function algorithm, unless other wise stated.

8 Time Dependent Failure Rates

In the methods explained above in Sections 5, 6, and 7 we used constant, similar probabilities of failure for the components. However, this is unreasonable as in the real world, components and the associated probabilities of failure are affected by natural disasters, weather related incidents, and the aging and degradation of components. A European report conducted by Rodioniv et al. on modeling component's probability of failure suggest and discuss multiple models on realistic component failure analysis [14]. Let $\lambda(t)$ be the probability of failure for a given component, dependent on the age of the component (in years). One modeling method the report suggests is simply a constant rate of failure (similar to what we assumed in the prior explanation of methods).

$$\lambda(t) = \lambda_0$$

Although this model seems too simplistic, the report establishes this model can be used for a finite, restricted age range of t. Clarotti et al. utilize this constant probability of failure which sufficiently modeled the components failure rate for a specified, finite time period [15]. However, the paper acknowledges that finding the moment that the failure rate is no longer constant possesses a challenge and attempts to determine this failure point using Bayesian techniques.
A second modeling method the paper suggests is using linear aging.

$$\lambda(t) = \lambda_0 + bt$$

Vesely et al. used this linear degradation due to the assumption of a constant increasing of the probability of failure [16]. However, it was found in a statistical survey (Atwood et al.) that linear modeling of real life components was insufficient and did not properly model the log- normal failure rates that occurred in the sample [17]. Similarly to the constant probability model, a finite, restricted range of t must be used to model a portion of the age of the components.
A failure probability model that we explored was:

$$\lambda(t) = \frac{\lambda_0 t}{1 + t}$$

Unlike the linear and constant probability models, this formula does not require constrained or finite time, since the probability of success of the component:

$$1 - \frac{\lambda_0 t}{1 + t}$$

converges to zero and starts at one at t_0. The λ_0 we chose was 0.07, though this should be determined through a careful survey of real world power grid component failure rates.
Finally, the European report discussed a Weibull aging model which we explored as well:

$$\lambda(t) = \lambda_0 e^{\beta \ln t}$$

with β at 0.15 and λ_0 at 0.07. But again these values should be determined through a careful survey of real world power grid component failure rates.

9 Results

9.1 Comparing Methods

We firstly compare the results of the computational methods and algorithms we used. For all bench marking experiments, we use an average of five branches in each fork, with diverse and variable number of customers in each branch. The first method we benchmark was the recursive method explained in Section 5. The second method we benchmark is the moment generating function method

Number of Forks	Time (in seconds)
2	1.372
3	1.389
4	35.729
5	762.3
6	11628

Table 1 Recursive Method Benchmark

as explained in Section 6. Finally, we benchmark the third method of the

Number of Forks	Time (in seconds)
2	0.392
4	0.575
8	5.41
10	42.55
12	107.2

Table 2 Moment Generating Function Method Benchmark

polynomial - probability generating mass function as explained in Section 7. It can be seen that the final method (the Polynomial - Probability Generating Function method) far out competes the other two methods both in speed and in the number of forks run.

Number of Forks	Time (in seconds)
2	0.112
4	0.213
12	0.3097
25	0.6367
100	3.51
300	23.07

Table 3 Polynomial - Probability Generating Function Method Benchmark

9.2 Exploring the Probability Mass Function

It is interesting to note the shape of the distribution for differing number of forks. For three forks we have a probability mass function looking like Figure 2. For ten forks we have a probability mass function looking like Figure 3.

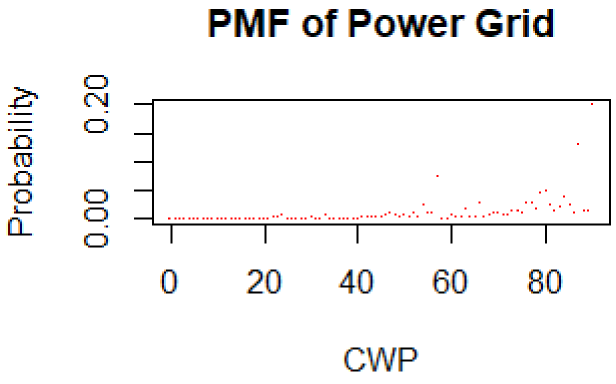


Fig. 2 Three Fork Probability Mass Function

For twenty forks we have Figure 4. And for forty forks we have Figure 5 It is interesting to note the convergence to a skewed binomial distribution for larger values of forks.

9.3 Branch's vs. Forks

Using the polynomial algorithm, we are curious as to the effectiveness of adding more branch's or more forks. This problem would be of practical importance to power companies when developing and deploying power grid systems. The way we measure effectiveness is by the skewness. The more skew to the left the probability mass function of the power grid, the better the system since we want the probability of the customers with power to be larger for more

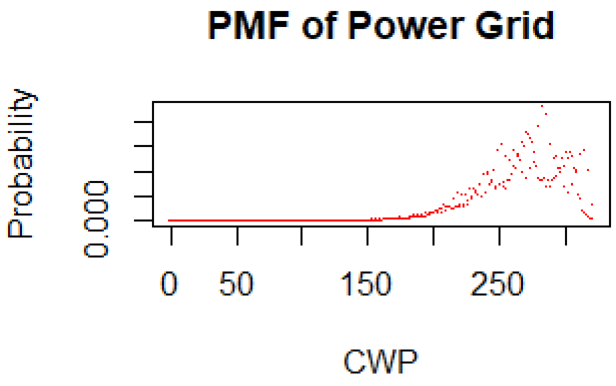


Fig. 3 Ten Fork Probability Mass Function

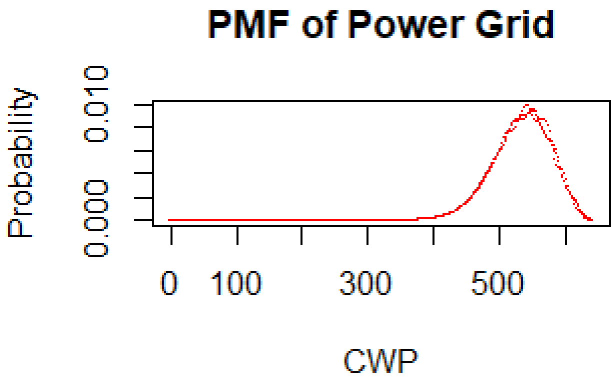


Fig. 4 Twenty Fork Probability Mass Function

customers, implying more customers have power. We measure skewness using:

$$\frac{\sum_{i=1}^{N}(x_i - \bar{x})^3}{Ns^3}$$

We use the "e1071" package in R to perform this computation [18]. Skewness values between -0.5 and 0.5 imply a symmetric distribution. Values between -1 to -0.5 and 0.5 to 1 imply moderately skewed whereas above 1 and below -1 imply heavily skewed distributions. In the comparison between adding more branch's or forks, we wish to see which has a higher skewness value, implying there are more customers with power. To benchmark, we use all values of

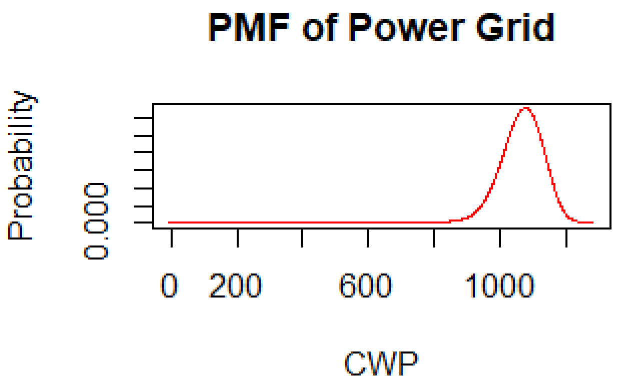
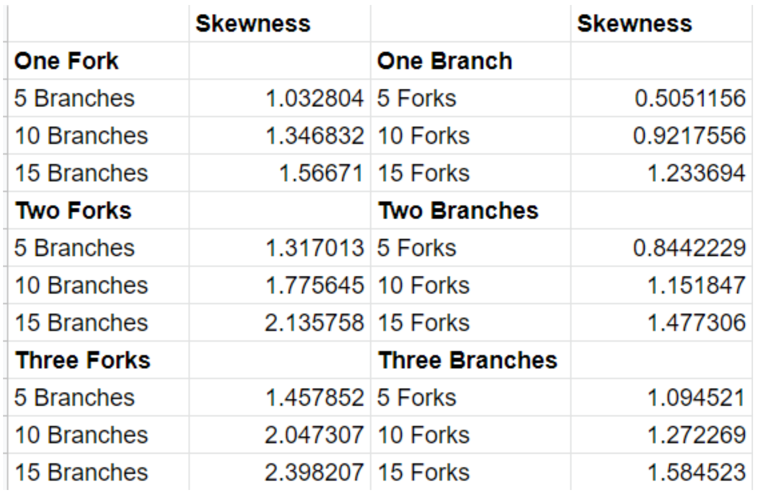


Fig. 5 Forty Fork Probability Mass Function

customers for each branch to be one. Below is the results of our experiment: The first two columns (One Fork, Two Fork, Three Fork) represent having more



	Skewness		**Skewness**
One Fork		**One Branch**	
5 Branches	1.032804	5 Forks	0.5051156
10 Branches	1.346832	10 Forks	0.9217556
15 Branches	1.56671	15 Forks	1.233694
Two Forks		**Two Branches**	
5 Branches	1.317013	5 Forks	0.8442229
10 Branches	1.775645	10 Forks	1.151847
15 Branches	2.135758	15 Forks	1.477306
Three Forks		**Three Branches**	
5 Branches	1.457852	5 Forks	1.094521
10 Branches	2.047307	10 Forks	1.272269
15 Branches	2.398207	15 Forks	1.584523

Fig. 6 Skewness Experiment Results

branches instead of forks. Alternatively, the third and fourth columns (One Branch, Two Branch, Three Branch) represents having more forks instead of branches. Because this experiment does not contain a random sample of the possible fork and branch combinations, we cannot do a significance test of the difference of skewness between the two methods. However, in future studies it would be interesting to randomly sample fork and branch combinations to statistically test the true difference of skewness between the two methods.

Although, we cannot do a significance test, we do notice the more branch method (the left columns) seems to be far more positively skewed than the more forks method (the right columns). With this observation, this suggests having more branches is skewer to the left; thus, there are more customers with power, thus the power grid is more efficient. Therefore, we recommend for the simple model we are using to add more branches before adding more forks, if possible.

9.4 Components with Time

To show how an aging power grid could affect the probability mass function, we ran our algorithm with probabilities of failure dependent on time (as described in Section 8). The result is shown below with the power grid years at 0.01, 1, 5, and 20 years. As expected, the probability mass function of more aged

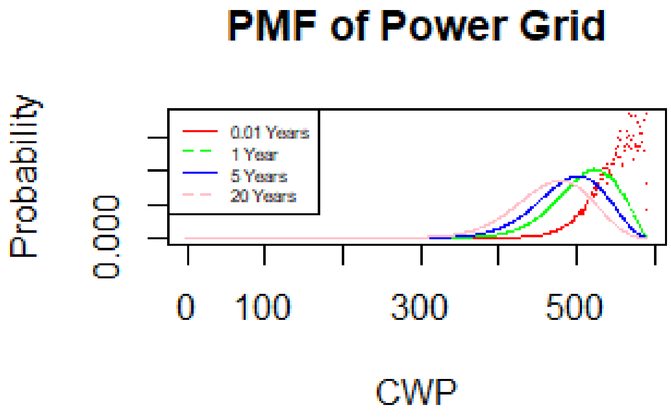


Fig. 7 Aging Power Grid Experiment Results

components is less skewed to the left implying having larger values of customers with power gets more difficult with age.

10 Future Work

Future work of this project would consist of expanding the complexity of the model. Currently, assuming the main power line does not fail and the forks are independent is unreasonable and does not properly model the real world. Thus, finding an efficient algorithm to model this scenario would be beneficial to the industry. It would also be beneficial to establish dependence between the branches in order for it to be more realistic. This would be due by considering the branches in close geographical proximity would be more likely to fail if one of the branches failed.However, these models would add increasing complexity and more of an immense amount of combinations to take into account.

11 Conclusion

In conclusion, we explored numerous algorithms to calculate the probability mass function of the power grid, with the most efficient method being the polynomial - probability generating function method. Using this algorithm, we explored different properties of the power grid such as skewness determined by time and convergence to a skewed binomial distribution. Although, this work consisted of assuming independence between the forks and that the main power line did not fail, future studies should work to develop an efficient algorithm without these assumptions.

Acknowledgments. This work was supported by NSA grant number 71321 and NSF grant DMS-1731082 to Dr. Javier Rojo through the RUSIS@OSU program.

Appendix A Code

The following is the R code of the polynomial- Probability Generating Function model as explained in Section 7:

```
library(polynom)
library(gtools)
library(stringr)
start_time <- Sys.time()

df <- data.frame(CWP=integer(),
                 prob=numeric())

PDF.real = data.frame(CWP=integer(),
                      prob=numeric())

weibullFailureRate <- function(componentAge,
initialF = 0.07, beta = 0.15) {
  return(1 - (initialF*exp(beta*log(componentAge))))
}

CWP.individual <- function(customers,
componentAge, branchProb) {
  # get the probabilities with respect to time
  probs = weibullFailureRate(componentAge)
  branchProb = weibullFailureRate(branchProb)

  x = list()
```

```
for (i in 1:length(customers)) {
  x[[i]] <- c(customers[i], i)
}
allProbs = numeric()
# calculate the zero prob
zeroProb = (1 - branchProb) + (branchProb*
prod(1 - probs))
df[nrow(df) + 1,] = list(CWP=0, prob=zeroProb)
allProbs = append(allProbs, zeroProb)
allCombos = integer()
allCombos = append(allCombos, 0)

for (i in 1:length(x)) {
  # create all i combos of the fork
  combo = combn(x, i)
  for (j in 1:ncol(combo)) {
    comb = combo[,j]
    # get the indexes of the combinations
    indexes = which(x %in% comb)
    s = 0
    for (c in comb) {
      s = s + c[1]
    }
    # change the indices of prob of failure to prob
    # of success
    failure = 1 - probs
    for (ind in indexes) {
      failure[ind] = 1 - failure[ind]
    }
    # multiply the probs
    prob = branchProb*prod(failure)
    if (s %in% allCombos) {
      prior = df$prob[df$CWP == s]
      df$prob[df$CWP == s] <- prior +
      prob
    } else {
      allCombos = append(allCombos, s)
      df[nrow(df) + 1,] = list(CWP=s,
      prob=prob)
      allProbs = append(allProbs, prob)
    }
  }
}
# check for duplicates. If found add the probs up
CWPSeen = integer()
```

```
  for (i in 1:nrow(df)) {
    cwp = df[i,1]
    prob = df[i, 2]
    if (cwp %in% CWPSeen) {
      prior = PDF.real$prob[PDF.real$CWP == cwp]
      PDF.real$prob[PDF.real$CWP == cwp] <- prior +
      prob
    } else {
      CWPSeen = append(CWPSeen, cwp)
      PDF.real[nrow(PDF.real) + 1,] = list(CWP=cwp,
      prob=prob)
    }
  }
  return(PDF.real)
}

CWP.all <- function(...) {
  powerGrid = list(...)
  lPowerGrid = length(powerGrid)
  firstLoopCount = lPowerGrid / 3
  print(paste("Running␣for", firstLoopCount, "forks"))
  # start with polynomial 1
  all.polynomial = polynomial(1)
  count = 1
  for (i in 1:firstLoopCount) {
    if (count %% 1000 == 0) {
      print(paste("On␣Loop:", i))
    }
    # get the PMF of the fork_i
    if (i == 1) {
      individual.pdf = CWP.individual(powerGrid[i][[1]],
                                      powerGrid[i + 1][[1]],
                                      powerGrid[i + 2][[1]])
    } else {
      individual.pdf = CWP.individual(powerGrid[(i*3)-2][[1]],
                                      powerGrid[(i*3)-1][[1]],
                                      powerGrid[(i*3)][[1]])
    }
    # create a vector of zeros
    x.polynomial = integer(max(individual.pdf$CWP)+1)
    # assign CWP indexes + 1 to the probabilities
    x.polynomial[individual.pdf$CWP+1] <- individual.pdf$prob
    # convert the vector of probs to a polynomial
    x.polynomial <- as.polynomial(x.polynomial)
    # multiply the previous polynomial to the current one
    all.polynomial = all.polynomial*x.polynomial
    count = count + 1
  }
  # extract the exponent polynomial (is the CWP)
  expression = as.character(all.polynomial)
  allValues = str_split(expression, "\\+")[[1]]
  zeroMet = FALSE
  allCombos = integer()
  for (exp in allValues) {
    val = str_trim(str_split(exp, "\\^")[[1]][2])
    if (is.na(val)) {
      if (zeroMet == FALSE) {
        zeroMet = TRUE
        allCombos = append(allCombos, 0)
      } else {
        allCombos = append(allCombos, 1)
      }
    } else {
      allCombos = append(allCombos, strtoi(val))
    }
  }
  # get the polynomial coefficients (which is the probabilities)
  allProbs = coef(all.polynomial)
  # removes all zero coefficients from coefficients
  allProbs = unlist(lapply(allProbs, function(x) {x[x!=0]}))
  finalPDF = data.frame(allCombos, allProbs)

  print(format(tail(finalPDF), scientific=F))
  print(paste("Sum␣of␣Probs␣is:", sum(allProbs)))

  plot(allCombos, allProbs, pch = ".", xlab = "CWP", ylab = "Prob
       main = "PMF␣of␣Power␣Grid", col = "red")
  print(skewness(allProbs))
}

# Main Function Call:
# One Fork is: c(customers), c(component ages), fork Age
# This is an example of 6 forks
CWP.all(c(5, 15, 16, 1, 2, 1), c(1, 2, 3, 3, 2, 1), 10,
        c(3, 10, 11, 3, 5, 3), c(1, 2, 3, 3, 2, 1), 10,
        c(5, 7, 9, 1, 2), c(1, 2, 1, 1, 1), 1,
        c(3, 10, 8, 9, 3, 4), c(1, 2, 3, 3, 2, 1), 10,
        c(4, 2, 14, 3, 3, 3), c(1, 2, 3, 3, 2, 1), 10,
        c(3, 15, 1), c(1, 2, 3, 3, 2, 1), 10)
```

```
end_time <- Sys.time()
print(paste("This took:", end_time - start_time))
```

References

[1] Navon, A., Machlev, R., Carmon, D., Onile, A.E., Belikov, J., Levron, Y.: Effects of the covid-19 pandemic on energy systems and electric power grids—a review of the challenges ahead. Energies **14**(4) (2021). https://doi.org/10.3390/en14041056

[2] The Regulatory Assistance Project (2011). https://www.raponline.org/wp-content/uploads/2016/05/rap-lazar-electricityregulationintheus-guide-2011-03.pdf

[3] ASCE: ASCE's 2021 American Infrastructure Report Card: GPA: C- (2021). http://www.infrastructurereportcard.org/

[4] Evans, D.L., Leemis, L.M.: Algorithms for computing the distributions of sums of discrete random variables. Mathematical and Computer Modelling **40**(13), 1429–1452 (2004). https://doi.org/10.1016/j.mcm.2005.01.003

[5] Marsadek, M., Mohamed, A., Z.M, N.: Risk of static security assessment of a power system using non-sequential monte carlo simulation. Journal of Applied Sciences **11** (2011). https://doi.org/10.3923/jas.2011.300.307

[6] Su, C.-L.: Probabilistic load-flow computation using point estimate method. IEEE Transactions on Power Systems **20**(4), 1843–1851 (2005). https://doi.org/10.1109/TPWRS.2005.857921

[7] Barabási, A.-L., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999) https://science.sciencemag.org/content/286/5439/509.full.pdf. https://doi.org/10.1126/science.286.5439.509

[8] Chassin, D.P., Posse, C.: Evaluating north american electric grid reliability using the barabási–albert network model. Physica A: Statistical Mechanics and its Applications **355**(2), 667–677 (2005). https://doi.org/10.1016/j.physa.2005.02.051

[9] Warnes, G.R., Bolker, B., Lumley, T.: Gtools: Various R Programming Tools. (2020). R package version 3.8.2. https://CRAN.R-project.org/package=gtools

[10] Dueñas-Osorio, L., Rojo, J.: Reliability assessment of lifeline systems with radial topology. Computer-Aided Civil and Infrastructure Engineering **26**(2), 111–128

(2011) https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8667.2010.00661.x. https://doi.org/10.1111/j.1467-8667.2010.00661.x

[11] Chambers, J.M., Hastie, T.: Statistical Models in S. CRC Press, ??? (1999)

[12] R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2020). R Foundation for Statistical Computing. https://www.R-project.org/

[13] Venables, B., Hornik, K., Maechler, M.: Polynom: A Collection of Functions to Implement a Class for Univariate Polynomial Manipulations. (2019). R package version 1.4-0. S original by Bill Venables, packages for R by Kurt Hornik and Martin Maechler. https://CRAN.R-project.org/package=polynom

[14] Rodioniv, A., Patrik, M., Atwood, C., Cronval, O.: Models and Data Used for Assessing the Ageing of Systems, Structures and Components (2007). https://publications.jrc.ec.europa.eu/repository/handle/JRC36344

[15] Clarotti, C., Lannoy, A., Odin, S., Procaccia, H.: Detection of equipment aging and determination of the efficiency of a corrective measure. Reliab. Eng. Syst. Saf. **84**, 57–64 (2004)

[16] Vesely, W.E., Corporation, S.A.I., of Nuclear Regulatory Research. Division of Engineering Safety, U.S.N.R.C.O., Idaho, E..G., Laboratory, I.N.E.: Risk Evaluations of Aging Phenomena: The Linear Aging Reliability Model and Its Extensions. EG & G Idaho, ??? (1987). https://books.google.com/books?id=n4NWAAAAMAAJ

[17] Atwood, C.: Parametric estimation of time-dependent failure rates for probabilistic risk assessment. Reliability Engineering & System Safety **37**, 181–194 (1992)

[18] Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., Leisch, F.: E1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. (2021). R package version 1.7-6. https://CRAN.R-project.org/package=e1071