A Min-Cut Approach for Contingency Analysis

Srikanth B. Yoginath, Maksudul Alam, Pratishtha Shukla, and Nils Stenvig
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
{yoginathsb, alamm, shuklap, stenvign}@ornl.gov

***Abstract*—The networked architecture of the power grid makes it resilient against component failures. However, assessing the effects of k component failures in a real-world power grid with tens of thousands of components is computationally expensive even for a small value of k when all combinations of k-component failures need to be evaluated. Using the graph *min-cut* algorithm, we can determine the edge-sets of k edges whose loss could potentially disconnect the connected components from the grid. Determining these edge-sets using graph algorithms significantly reduces the number of combinations to be evaluated for contingency analysis and hence, can be practically significant. However, finding all the *min-cut* edge-sets for a given graph is also computationally expensive, as the computational time significantly increases with the graph size. In this paper, we present a novel algorithm that can not only compute all the *min-cut* edge-sets of real-world power grids but also scale linearly over hundreds of processor cores to reduce the computation time.**

***Index Terms*—Contingency analysis, graph algorithms**

I. Introduction

The reliability and resilience of modern power grids are essential to ensure an uninterrupted electricity supply [1]. Increasing demand variability driven by electrification, changing consumption patterns, and extreme events have intensified the need for more comprehensive reliability assessments [2]. The inherent redundancy of transmission networks should enable them to withstand component failures. They are regularly evaluated for such failures with traditional $N-1$ contingency criterion [3]. However, analyzing multiple simultaneous outages ($N-k$ contingencies) remains computationally challenging, as the number of possible combinations grows rapidly even for small $k=2$ [4].

Among all possible contingencies, those that result in disconnection of buses or loads from the path of power supply are particularly critical because they directly compromise power delivery and voltage stability [5]. By focusing on topologically disruptive contingencies, it is possible to significantly reduce the number of component failure scenarios that must be evaluated.

Traditional $N-k$ contingency analysis is performed by executing power flow simulations [6] for all combinations of k component outages out of N total components, leading to $\binom{N}{k} = \frac{N!}{k!(N-k)!}$ evaluations. By modeling the power grid as a graph, we can use the minimum-cut (*min-cut*) algorithm to identify sets of k edges whose removal would disconnect one or more buses from the rest of the network [7]. These edge sets correspond to physically significant $N-k$ contingencies that compromise grid connectivity. Our graph-based method sidesteps the brute-force approach by identifying only those k-edge sets that result in topological disconnection. This drastically limits the number of critical $N-k$ scenarios to be evaluated, enabling practical analysis of large-scale grids with strict analysis time.

In this work, we introduce novel algorithms to efficiently compute all k-edge min-cuts in large undirected graphs, show that these algorithms are amenable to parallel computing, and thus result in linear speedups of computation times. We demonstrate the algorithms on both IEEE test model and real transmission system data obtained from a Transmission System Operator. Our results show that the proposed method reduces the number of scenarios to be evaluated by several orders of magnitude, making at least $N-2$ contingency analysis for large-scale power grids practical for near-real-time applications.

II. Algorithm

A graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ consists of a set of vertices or nodes ($\mathcal{V}$) and a set of node pairs or edges $\mathcal{E}$. This mathematical abstraction is widely used to model real-world networks such as road systems, communication networks, and power grids [8].

The *min-cut* algorithm is a cut between source s and target t, where capacity of the cut edges is minimum over all cuts of the network. A cut (S,T) in graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ is a partition of $\mathcal{V}$ into S and $T=\mathcal{V}-S$ such that $s\in S$ and $t\in T$. With a complexity of $O(\mathcal{V}^2\mathcal{E})$, the *Push-relabel* algorithm [9] to determine the *min-cut* is widely used. However, currently, as far as we are aware, there exists no algorithm to determine all possible *min-cuts* for all possible s and t combinations of a given input graph $\mathcal{G}$. The Gomory-Hu [10] method comes closest to this task, where a Gomory-Hu tree is constructed to compute the *min-cut* values for all given s and t combinations of graph $\mathcal{G}$, but it does not provide the cut edge-sets.

Here, we present an algorithm that determines all possible cut-edges for a *min-cut* value k for a given graph $\mathcal{G}(\mathcal{V},\mathcal{E})$. Further, this algorithm works for all values of $k=1,2,3,\cdots$ provided that k is the *min-cut* of the input graph.

This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy (DOE). The work is sponsored by DOE Office of Electricity (OE). The U.S. government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for U.S. government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (https://www.energy.gov/doe-public-access-plan).

979-8-3315-8138-1

Algorithm 1 Get all *min-cut* edge-sets

Require: $G \leftarrow$ Input undirected-weighted graph
Require: $m_v \leftarrow$ Input graph G's *min-cut* value
Require: $s_g \leftarrow$ Input graph G's super-source node-id
1: $E_s \leftarrow \emptyset$ ▷ return value, list of all edge-sets for m_v
2: $c_v^n \leftarrow \emptyset$ ▷ return value, next *min-cut* value
3: $L_b \leftarrow 0$ ▷ list of all nodes other than source
4: $c_v \leftarrow 0$ ▷ cut value
5: $e_s \leftarrow \emptyset$ ▷ cut edges
6: $w_j \leftarrow 0$ ▷ weight of edge j
7: $i_f \leftarrow \frac{1}{m_v}$ ▷ increment factor
8:
9: **for** each node b_i in list L_b **do**
10: $g \leftarrow$ Copy (G)
11: **while** $c_v \leq m_v + 1$ **do**
12: $c_v, e_s \leftarrow$ minimum_cut (g, s_g, b_i)
13: $E_s \leftarrow$ append (e_s)
14: **for** each edge j in list e_s **do**
15: **if** $w_j < 1 + i_f$ **then**
16: $w_j = w_j + i_f$
17: **end if**
18: **end for**
19: **end while**
20: **end for**
21: $c_v^n = c_v$
22: return c_v^n, E_s

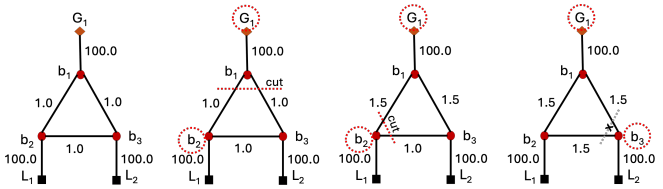
A. Iterative Approach

The pseudocode to obtain all the *min-cut* edges given an input graph is shown in Algorithm 1. We start with an undirected and weighted graph, with the weight of every edge in the graph equaling 1.0. The algorithm takes this graph G, a single super source node s_g of G, and a *min-cut* value of m_v as the inputs. For a given s_g, we iterate through other nodes b_i of G, where $i = 1, 2, 3 \cdots$ (line 9). For every combination of s_g and b_i, the *Push-relabel* algorithm (shown in *minimum_cut* line 12) is iteratively used to determine all the edge-sets for a *min-cut* value of $m_v = k$. At each such iteration, we increment the weight w_j of each edge of the detected *min-cut* edge-set by a constant value of $i_v = \frac{1}{k}$ (line 16). We continue these inner iterations (lines 11–19) with the newly re-weighted graph to obtain the next edge-set of *min-cut* edges (lines 8 – 19). This iterative process continues until the cut value (c_v) is $c_v > m_v + 1$ condition is reached; at this point, all of the edge-sets responsible for the *min-cut* value $m_v = k$ are detected. However, the graph g updated with new edge weights for a given s_g and b_m pair cannot be used as an input for a different s_g and b_n because of the possibility of *information loss* (discussed in next section).

Incrementing the edge weight w_j (line 16) by a factor of $\frac{1}{k}$ ensures that the detected edge-set differs at least by one *edge* from the previously identified edge-sets for a given *min-cut* value. For an example case with $m_v = 3$, if $E_s^1 = e_1, e_2, e_3$, is an identified *min-cut* edge-set, then their corresponding weights w_{e1}, w_{e2}, w_{e3} are incremented by $\frac{1}{m_v} = 0.34$ to 1.34. During the next iteration, the sum of their weights will no longer be less than $m_v + 1$ and hence the same edge-set becomes invalid due to the $c_v \leq (m_v+1)$ condition. Supposing edge-set $E_s^2 = e_1, e_2, e_4$ with $w_{e4} = 1$ exists, then with their aggregated weight being $3.68 < m_v + 1$ will be picked as a *min-cut* edge-set.

B. Information Loss

Consider the graph $\mathcal{G}$ derived from the power grid example shown in Fig. 1a. In this graph $\mathcal{G}$, node G_1 represents the generator, L_1, L_2 represent the loads, and b_1, b_2, b_3 represent the buses. For illustration purposes we ignore single edge *min-cut*, and it is realized by increasing the weights of these edges (G_1, b_1), (b_2, L_1), and (b_3, L_2) to some large value 100.0 as shown in Fig. 1a.



(a) Example Graph (b) Cut 1 (c) Cut 2 (d) Missed Cut 3

Fig. 1: Information loss illustration.

When the *min-cut* operation is performed on this graph using our algorithm, we only obtain two cuts for $m_v = 2$. The first cut is shown in Fig. 1b. The algorithm then increments the weight of the edges by $\frac{1}{m_v} = 0.5$ after detecting this cut, and thus the weights of edges (b_1, b_2) and (b_1, b_3) increase to 1.5 each, as shown in Fig. 1c. The next *min-cut* operation performed on the updated graph results detects edge-set containing (b_1, b_2) and (b_2, b_3) edges, as shown in Fig. 1c. The algorithm then increments the weights of all edges in the *min-cut* edge-set that have not been previously incremented. In this example, the edge weight of (b_2, b_3) is incremented to 1.5. However, our algorithm would be unsuccessful to detect another possible *min-cut* edge-set with edges (b_2, b_3) and (b_1, b_3) because the condition $c_v < m_v + 1$ is not satisfied, as shown in Fig. 1d. Hence, using the graph with altered weights due to previously detected edges results in information loss. The occurrence of such information loss is possible for graph $\mathcal{G}$ with $m_v > 1$

We can overcome the risk of information loss by using the unaltered initial graph as the starting point for exploring different (s, t) pairs. Then the first *min-cut* edge-sets (E_{G_1,b_2}) with ($s = G_1, t = b_2$) pair will be $E_{G_1,b_2} = \{(b_1, b_2)(b_1, b_3)\}, \{(b_1, b_2)(b_2, b_3)\}$. Similarly, for ($s = G_1, t = b_3$) pair we obtain an edge-sets $E_{G_1,b_3} = \{(b_1, b_2)(b_1, b_3)\}, \{(b_1, b_3)(b_2, b_3)\}$. Hence, a duplication of the *min-cut* edge-set $\{(b_1, b_2)(b_1, b_3)\}$ is seen with the approach but all the edge-sets are detected.

Further, this algorithm is guaranteed to iteratively detect the *min-cut* edge-sets as long as they differ from each other by a single edge for a given (s, t) pair; we demonstrated this with a cyclic graph example in Fig. 1. This unmodified input graph $\mathcal{G}$ requirement by the algorithm makes it amenable to parallel computing, since every iterations in between lines 9 –

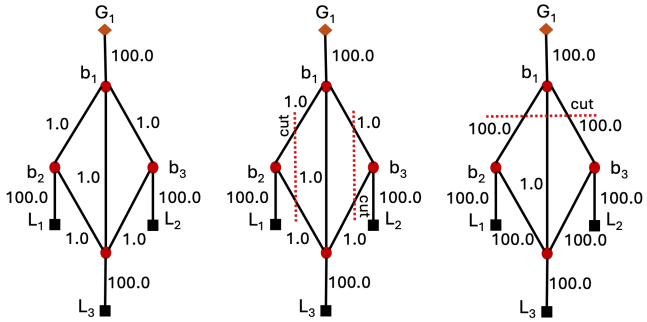
20 become independent of each other. However, the duplicates need to be removed from the returned E_s list of edge-sets that is aggregated from all the parallel processes.

C. Edge-sets for Different Cut-Values

As previously discussed, using Algorithm 1, we can detect all edge-sets for a given *min-cut* value $m_v = k$ for a given graph, regardless of the value of k. In this section, we demonstrate the possibility of computing the edge-sets for all possible cut-values $c_v = 1, 2, 3, \cdots$ for an undirected graph.

Algorithm 2 Get edge-sets for all cut-values

Require: $d \leftarrow$ Input IEEE/Raw/PSSE case file
1: $E_d \leftarrow \emptyset$ ▷ return Edge-set dictionary
2: $G \leftarrow \emptyset$ ▷ Graph generated using d
3: $s_g \leftarrow 0$ ▷ super-source connects all sources
4: $c_v \leftarrow 1$ ▷ cut value
5: $w_j \leftarrow 0$ ▷ weight of edge j
6: $\mathcal{L}_\mathcal{W} \leftarrow 100$
7:
8: $G \leftarrow$ generate_Undirected_Weighted_Graph(d)
9: $s_g, G' \leftarrow$ create_Super_Generator (G)
10: **while** $c_v < \mathcal{L}_\mathcal{W}$ **do**
11: c_v^n, $E_s \leftarrow$ get_all_minimum_cut_edgesets (g, c_v, s_g)
12: $E_d \leftarrow$ insert_to_dictionary $\{c_v\colon E_s\}$
13: ▷ Update G'
14: **for** each edge_set $i \in G'$ in edge_sets E_s **do**
15: **for** each edge j in edge_set i **do**
16: **if** $w_j < \mathcal{L}_\mathcal{W}$ **then**
17: $w_j = \mathcal{L}_\mathcal{W}$
18: **end if**
19: **end for**
20: **end for**
21: $c_v \leftarrow c_v^n$
22: **end while**
23: return E_d



(a) Example Graph (b) $m_v = 2$ (c) $m_v = 3$

Fig. 2: Illustrating the overlap of consecutively computed edge-sets using *min-cut* computations for $m_v > 1$ cases for a given undirected connected graph. Here, the edge-sets for the first *min-cut* value $m_v^1 > 1$ will be complete, while the edge-sets for all later *min-cut* value $m_v^k > m_v^1$ could be partial.

By iteratively using Algorithm 1 while ensuring the increase in the *min-cut* values $m_v = 1, 2, \cdots$, and increasing the weights of every identified edge in the edge-sets to a large weight value $\mathcal{L}_\mathcal{W} = 100$, as described in Algorithm 2, we can partially detect the edge-sets for all the cut-values. By increasing the weights of the edges in the previously detected edge-sets, the proposed method essentially renders the identified edges extremely expensive for the *min-cut* algorithm to consider. Thus, every previously identified edge in the edge-set pertaining to a lower *min-cut* value will not be a part of any edge-set pertaining to a higher min-cut value.

This approach works perfectly for cases where the *min-cut* value of the input graph starts with $m_v = 1$ and the very next $m_v = (2 \text{ or } 3 \text{ or } 4 \cdots)$ value after weight adjustments, but it works only partially for the cases beyond this. We illustrate this with an example in Fig. 2, which consists of multiple 1-edge, 2-edge, and 3-edge cuts. In Fig. 2a, the 1-edge cut edge-sets are identified and their weights are adjusted to a $\mathcal{L}_\mathcal{W} = 100$. In Fig. 2b, possible 2-edge cuts are shown, and their weight adjustments after detection are shown in Fig. 2c. It can be noted that transitioning after the identification of the 1-edge cuts and adjusting their weights to $\mathcal{L}_\mathcal{W}$ does not affect the identification of 2-edge or 3-edge cuts, since the identified 1-edge cut edge-sets will never be a part of 2-edge and 3-edge cut edge-sets. However, the 2-edge cut edge-sets can be a part of 3-edge cut edge-sets, which is illustrated in Fig. 2c, where the edges that are part of the edge-sets for *min-cut* value of 2 are also a part of edge-sets for *min-cut* value 3.

Thus, even though the Algorithm 2 can get successive *min-cuts* up to the largest possible cut value in the input graph, because of the previously discussed limitation, only the edge-sets with *min-cut* =1 and the very next grouping of edge-sets for any *min-cut* value $m_v = k^*$, where $k^* > 1$, are complete. Hence, the Algorithm 2 does not guarantee completeness of the edge-sets for *min-cut* values $m_v > k^*$.

Remark: A successful functioning of Algorithm 2 shows that the sufficient condition to detect and enumerate all the edge-sets is met by Algorithm 1, because the sufficient condition of Algorithm 1 is the necessary condition for Algorithm 2 to function.

III. Implementation

We used Python, specifically its *networkx* module, to implement the algorithms discussed in Section II. We use the *multiprocessing* module of Python for parallel computing.

Undirected Weighted Graph: For constructing a computational graph from the IEEE case files and the state estimation data (PSSE) files, we consider the branch data corresponding to the transmission lines and transformers as edges ($\mathcal{E}$), and other elements such as buses, generators, and loads as nodes ($\mathcal{V}$). More details on graph construction can be found in [11].

We connect all the generators to one super-generator (s_g in line 12 of Algorithm 1) and assign a larger weight $w_e = \mathcal{L}_\mathcal{W}$ for the edges connecting the super-generator to the generators. Similarly, all the edges connecting the loads are also assigned a large weight of $w_e = \mathcal{L}_\mathcal{W}$. The weights of all the other edges in the graph as assigned to $w_e = 1.0$ (line 11 of Algorithm 1). Thus, all the edges connecting the super-generator to the generators and the bus to the load are made irrelevant in

min-cut computations. Further, computations are carried out to identify and re-weight the 1-edge *min-cut* edges.

Min-cut edge-sets: We use *networkx*'s *minimum_cut* function that takes the source and target (s, t), and the graph (G) as input arguments and returns a *min-cut* edge-set. We use this function iteratively as described in Algorithm 1 to obtain all the *min-cut* edges. This implementation is based on the Push-Recall algorithm of complexity $O(\mathcal{V}^2\mathcal{E})$. In the real-world power grids, there exist many cuts with $m_v = 1$ exist. A lower-complexity, depth-first search-based Trajan's bridge-finding algorithm of complexity $O(\mathcal{V} + \mathcal{E})$ is used to detect them. This significantly reduces the computation time to identify *min-cut* $m_v = 1$, (*bridges* function of *networkx*).

IV. Verification

To verify the correctness of our algorithms, we use the IEEE 14-bus system, one of the smallest synthetic power grid planning models.

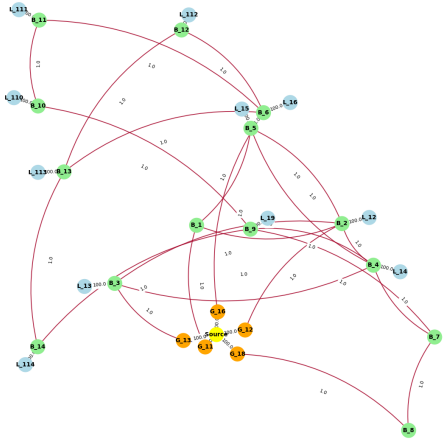
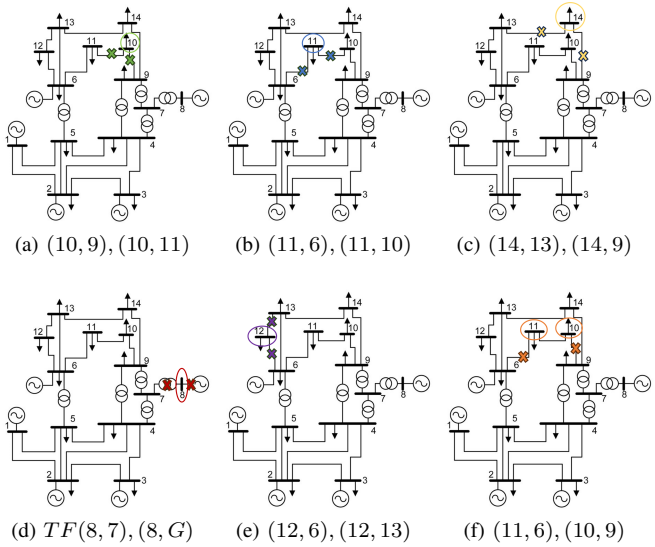


Fig. 3: Undirected-weighted graph of the IEEE 14-bus system.

A. All Minimum Cuts Across All Buses

Single Cut Value: Here, we identify all the minimum cut edge-sets for every $(Source, B_*)$ pair using the IEEE 14-bus system graph shown in Fig. 3. This synthetic power system case file does not contain any single-edge ($c_v = 1$) cuts. The minimum cut value for this case starts with $c_v = 2$. The minimum cut computation for $m_v = c_v = 2$ results in six edge-sets shown in Fig. 4a, 4b, 4c, 4d, 4e, and 4f one-line diagrams. As seen from Fig. 4a, the removal of edges (10, 9) and (10, 11) isolates the bus (B_10) and the associated load from the rest of the grid. Similarly, removal of edges (11,6) and (10,9) isolates buses 11 and 10 along with their associated loads from the rest of the grid, as shown in Fig. 4f. These examples show that one or more buses and loads are affected by the loss of the minimum number of edge cuts or branch losses. Further, these six edge-sets are all possible two-edge cuts in the IEEE 14-bus graph. With this example run, we verify the correctness of the working of Algorithm 1.

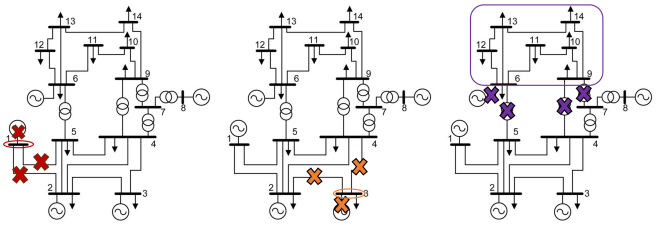
All Cut Values: Here, the functioning of the Algorithm 2 is verified. The Algorithm 2 identifies 6 edge-sets for *min-cut* value of $m_v = 2$ (Fig. 4a- 4f), two edge-sets for $m_v = 3$ (Fig. 5a and 5b), and one edge-set for $m_v = 4$ (Fig. 5c),



(a) $(10, 9), (10, 11)$ (b) $(11, 6), (11, 10)$ (c) $(14, 13), (14, 9)$

(d) $TF(8, 7), (8, G)$ (e) $(12, 6), (12, 13)$ (f) $(11, 6), (10, 9)$

Fig. 4: Algorithm 1 detects all possible edge-sets for *min-cut* $m_v = 2$ for the IEEE 14 bus system. We illustrate the disconnection of one or more buses due to the loss of edges in each individual edge-set.



(a) $m_v = 3$, (b) $m_v = 3$, (c) $m_v = 4$,

Fig. 5: In addition to the edge-sets shown in Fig. 4, the Algorithm 2 on IEEE 14 bus system detected more edge-sets (a) {(1,2), (1,5),(1,G)}, (b) {(8,2),(8,4) (8,G)}, and (c) {(9,7),(9,4),(6,5),(6,G)}, for different *min-cut* values.

before it terminates, as shown in the Fig. 4. Two points must be noted from the results. The fact that the *Push-Recall* algorithm results in a higher *min-cut* value after iterative execution of Algorithm 1 empirically proves that all possible *min-cut* for a prior cut-value have been identified, is the first point. The second point is that there is no overlap of the edges in the edge-sets corresponding to different *min-cut* values ($m_v = 2$, $m_v = 3$, and $m_v = 4$), as seen in Fig. 4. These observations state that the edge-sets for the first *min-cut* value identified after $m_v > 1$ are guaranteed to completely include all possible edge-sets, while the edge-sets for *min-cut* values after this might not be complete. For example: one cut edge-set for $c_v = 3$ is {(B_13,B_12), (B_13,B_6), (B_13,B_14)}. However, it is not detected by Algorithm 2 because the edge (B_13, B_14) is a part of the edge-set corresponding to $m_v = 2$ case (Fig. 4c).

V. Performance

The performance evaluation used a server machine with two AMD EPYC 7742 64-Core processors that provide a total of 128 simultaneous multi-threads, with a TeraByte of memory, and NVMe SSD-based storage devices.

Case	$\mathcal{V}$	$\mathcal{E}$	L	$N = \mathcal{E} - L$	$(N-2)_c$	$(N-2)_m$	Reduction	Runtime(s)
IEEE-14	30	36	11	25	300	6	50	0.41
IEEE-57	106	127	42	85	3,570	61	58.5	0.54
IEEE-118	271	332	99	233	27,028	61	443.1	0.74
IEEE-300	566	675	197	478	114,003	116	982.8	2.42
Utility-PSSE	34366	37359	16421	20938	219,189,453	17045	12859.4	23381.46

TABLE I: Tabulates performance results. The $\mathcal{V}$, $\mathcal{E}$, and L represent the number of nodes, edges and loads in the graph ($\mathcal{G}$) generated from IEEE or PSSE case files, an example graph for IEEE-14 bus is shown in Fig. 3. In our *min-cut* algorithms, we do not include edges connecting to loads, hence $N = \mathcal{E} - L$. $(N-2)_c$ column computes number of combinations of 2-edge removal scenarios $\binom{N}{2}$ required for $N-2$ contingency analysis. $(N-2)_m$ column lists the number of 2-edge removal scenarios identified by the *min-cut* Algorithm 1 for $m_c = 1$ and $m_c = 2$. The *Reduction* column computes $\frac{(N-2)_c}{(N-2)_m}$, quantifying gain through *min-cut* approach. The *Runtime* tabulates the computation time of the *min-cut* algorithm.

We use real-world state estimation data files from a Utility and widely available IEEE grid models for our performance evaluation. In Table I, we show the performance numbers for different IEEE and PSSE case files. As observed from the Table I several orders-of-magnitude of reduction in the scenarios to be evaluated are observed. For example: a four orders-of-magnitude reduction in the number of scenarios is observed from the real-world Utility-provided PSSE case file. Also, the runtime to compute all of 4,573 $m_v = 1$ edge-sets and all of 17045 $m_v = 2$ edge-sets *min-cut* edge-sets using Algorithm 1 for a real-world PSSE case file was observed to be large at 23381 seconds (≈6.5 hours) using 16 processor cores.

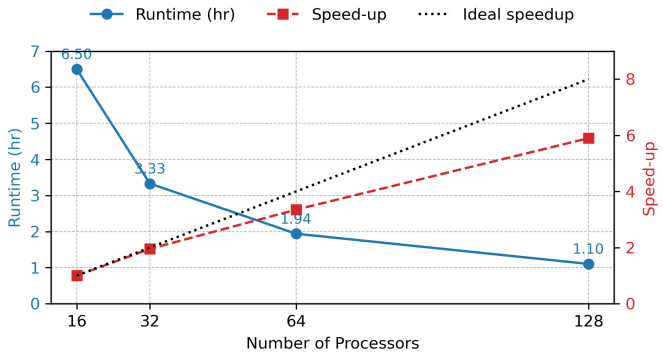


Fig. 6: Runtime and relative speedup of Algorithm 1

However, this runtime was reduced significantly by using additional parallel processing cores. As shown in Fig. 6, Algorithm 1 demonstrates strong scaling characteristics, achieving nearly linear speedup from 16 to approximately 128 cores. With 128 processors, we bring down the runtime from 6.5 hours to 1.1 hours. This can be further reduced with the increase in the number of processors over multi-node clusters, which is a part of our future work.

In addition to the above mentioned complete edge-sets for $m_v = 1$ and $m_v = 2$, Algorithm 2 was able to detect several partial edge-sets, including 417 $m_v = 3$ edge-sets, 85 $m_v = 4$ edge-sets, 13 $m_v = 5$ edge-sets, and 5 $m_v = 6$ edge-sets, for the Utility provided real-world state-estimation data file. Thus, $N-k$ contingency analysis for a real-world Utility for $k = 1, 2, 3, 4, 5, 6$ can be performed to a certain extent by evaluating 22,138 contingency scenarios using the *min-cut*-based algorithms discussed in this paper.

VI. Conclusion

In this paper, we addressed a practical combinatorial problem performing $N-k$ contingency analysis for a large-scale operational grid. We proposed a novel approach of using *min-cut* algorithms to detect combinations of edge failures that result in the disconnection of the nodes that could potentially introduce voltage instability and/or power outages. We developed algorithms to detect all the *min-cut* edge-sets iteratively while increasing the *min-cut* value of the input graph until the graph reaches a certain threshold, where no *min-cut* without previously detected edges becomes impossible. With IEEE case files and real-world state-estimation data from the Utility, we show that this approach results in orders-of-magnitude reduction in the $N-k$ contingency analysis scenarios. We also show that the developed algorithms are amenable to parallel computing and scale linearly with the number of processes, thus making the solution practically applicable for performing $N-k$ contingency analysis on an operational grid in near real time.

References

[1] R. N. Allan *et al.*, *Reliability evaluation of power systems*. Springer Science & Business Media, 2013.
[2] A. C. Gonçalves, X. Costoya, R. Nieto, and M. L. Liberato, "Extreme weather events on energy systems: a comprehensive review on impacts, mitigation, and adaptation measures," *Sustainable Energy Research*, vol. 11, no. 1, p. 4, 2024.
[3] M. Kateri, *Contingency table analysis*. Springer, 2014.
[4] A. Bagheri and C. Zhao, "Distributionally robust reliability assessment for transmission system hardening plan under nk security criterion," *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 653–662, 2019.
[5] L. Chang and Z. Wu, "Performance and reliability of electrical power grids under cascading failures," *International Journal of Electrical Power & Energy Systems*, vol. 33, no. 8, pp. 1410–1419, 2011.
[6] Z. Li, J. Wang, H. Sun, and Q. Guo, "Transmission contingency analysis based on integrated transmission and distribution power flow in smart grid," *IEEE Transactions on Power Systems*, vol. 30, no. 6, pp. 3356–3367, 2015.
[7] A. Dwivedi and X. Yu, "A maximum-flow-based complex network approach for power system vulnerability analysis," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 81–88, 2011.
[8] S. S. Skiena, *The algorithm design manual*. Springer, 2008, vol. 2.
[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.
[10] A. Abboud, J. Li, D. Panigrahi, and T. Saranurak, "All-pairs max-flow is no harder than single-pair max-flow: Gomory-hu trees in almost-linear time," in *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, 2023, pp. 2204–2212.
[11] S. B. Yoginath, P. Shukla, M. Alam, N. Stenvig, and T. Kuruganti, "A novel framework to quantify power grid resilience," in *IECON 2024 - 50th Annual Conference of the IEEE Industrial Electronics Society*, 2024, pp. 1–6.