Cyber Security Analysis of Power Networks by Hypergraph Cut Algorithms

Yutaro Yamaguchi, Anna Ogawa, Akiko Takeda, and Satoru Iwata

***Abstract*—This paper presents exact solution methods for analyzing vulnerability of electric power networks to a certain kind of undetectable attacks known as false data injection attacks. We show that the problems of finding the minimum number of measurement points to be attacked undetectably reduce to minimum cut problems on hypergraphs, which admit efficient combinatorial algorithms. Experimental results indicate that our exact solution methods run as fast as the previous methods, most of which provide only approximate solutions. We also present an algorithm for enumerating all small cuts in a hypergraph, which can be used for finding vulnerable sets of measurement points.**

***Index Terms*—False data injection, hypergraph, minimum cut, power network, security index, state estimation.**

I. Introduction

MAINTAINING the security and reliability of electric power networks is becoming more crucial due to their increasing scale and complexity. One needs to monitor the condition of a power network based on measurement of meters placed at important area of the power network through e.g., supervisory control and data acquisition (SCADA) system. The system uses state estimation, the process of estimating current states of the power system based on the meter measurement, in order to control power network components such as the operation of power generators.

Recently, various researchers have studied the cyber security of SCADA systems. Conventional techniques for detecting bad data (e.g., including inadvertent measurement errors) utilize the squares of differences between the observed measurements and their corresponding estimates. Those techniques are based on the assumption that the difference becomes significant when bad measurements happen [5], and are useful also in a situation when some attackers inject malicious measurement values. However, Liu *et al.* [6] pointed out that if attackers know the configuration of the power system, they can introduce a new class of attacks, false data injection attacks, to make malicious injection undetectable because the additive measurement values do not affect the difference. This means that an attacker can inject malicious values that will mislead the state estimation process without being detected by difference-based techniques for bad measurement detection. It should be mentioned that there is another approach to bad measurement detection, e.g., using a discrete-time hidden Markov process [18]. For related works in that direction, we refer the readers to [18], which also discussed much of the literature.

The attacker would be interested in finding sparse malicious values (i.e., sparse attacks) because each attack on a measurement point involves risk to be prevented. Sparse attacks reveal vulnerable sets of meters to be attacked in the system. Liu *et al.* [6] showed a cardinality minimization formulation to find a sparsest attack and solved it approximately by matching pursuit method [7]. Sandberg *et al.* [13] formulated a similar optimization problem for finding a sparsest attack including a given measurement point and defined the optimal value as the security index of the point. After Sandberg *et al.* [13] proposed a simple heuristics to find a suboptimal solution for the problem, Hendrickx *et al.* [2] and Sou *et al.* [14], [15] provided efficient solution methods as follows. It was shown in [14] that the problem reduces to a minimum cost node-bipartition problem, which is approximately solved by finding a minimum *s*–*t* cut in the given network. Under a certain strong assumption, it further reduces to a cardinality minimization problem whose constraint matrix is totally unimodular [15]. While ℓ_1-relaxation can indeed solve the cardinality minimization problem, the assumption restricts the applicability of the model. As an improved result of [14], Hendrickx *et al.* [2] gave a reduction of the security index computation to finding a minimum *s*–*t* cut in an auxiliary directed graph, which can be done in polynomial time.

In this paper, we present a new approach to cyber security analysis using hypergraphs. A hypergraph is a generalization of a graph consisting of nodes and hyperedges. Each hyperedge connects an unrestricted number of nodes while each edge in a graph connects just two nodes. Our main contributions are summarized below. The viewpoint of hypergraphs makes it possible, in particular, to find sparse attacks directly as shown in 2) and 3).

1) We show that computing the security index for a given measurement point reduces to finding a minimum *s*–*t* cut in a hypergraph, which can be done efficiently by finding a maximum *s*–*t* flow through the hypergraph. This problem in fact can be reduced to finding a minimum

Manuscript received May 14, 2014; revised August 17, 2014 and November 27, 2014; accepted December 28, 2014. Date of publication February 12, 2015; date of current version August 19, 2015. This work was supported by the Japan Science and Technology Agency (JST) Core Research for Evolutional Science and Technology (CREST). The work of Y. Yamaguchi was supported by the Japan Society for the Promotion of Science Fellowship for Young Scientists. Paper no. TSG-00432-2014.

Y. Yamaguchi, A. Takeda, and S. Iwata are with the Department of Mathematical Informatics, University of Tokyo, Tokyo 113-8656, Japan (e-mail: yutaro_yamaguchi@mist.i.u-tokyo.ac.jp).

A. Ogawa is with the Department of Mathematical Informatics, University of Tokyo, Tokyo 113-8656, Japan, and also with the Department of Administration Engineering, Keio University, Yokohama 223-8522, Japan.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSG.2015.2394791

s–*t* cut in a directed graph, which is almost the same as that suggested in [2]. Our reduction to hypergraphs, however, gives a simpler description with smaller number of nodes, which leads to improved running time in practice.
2) We show that a sparsest attack can be directly obtained by finding a minimum cut in a hypergraph. It should be noted that one can find a sparsest attack in polynomial time by applying the security index computation suggested in [2] to all measurement points. Our method is faster because it only requires one minimum cut computation.
3) We devise a new algorithm for enumerating all small cuts in a hypergraph whose capacities are within a prespecified constant factor of the minimum cut capacity, building on the work of Nagamochi *et al*. [9]. This algorithm can be used to analyze possible sparse attacks in power network systems.

The rest of this paper is organized as follows. Section II is devoted to problem formulations and fundamental properties of the sparsest attacks and security index. In Section III, we show that these problems can be reduced to hypergraph cut problems, which admit efficient combinatorial algorithms. The performance of our solution methods is analyzed through experiments in Section IV. Finally, in Section V, we consider all possible sparse attacks not necessarily the sparsest, and present a new algorithm for enumerating small cuts in a hypergraph which can be used to find all significant sparse attacks.

II. Cyber Security Analysis Problems

A. Sparsest Attacks and Security Index

Let $G = (V, A)$ be a directed graph, which represents a network system such as an electric power network. The node set V is of cardinality n and the arc set A of cardinality m. We assume that G is connected, and hence $n = O(m)$. We denote by B_G the incidence matrix of G. The row and column sets of B_G are indexed by V and A, respectively. For each arc $a = uv \in A$ from u to v, the (u, a)-entry is 1, (v, a)-entry is -1, and (w, a)-entries are 0 for all $w \in V \setminus \{u, v\}$.

In this paper, we consider the dc power flow model (see [1] for the detail), which was considered in [2], [6], and [13]–[15]. Each node has a hidden state (voltage phase angle), and each arc or node has a measurement point. The operator of the system gets a measurement value (active power flow on an arc or injection into a node) in each measurement point, which is determined from the states as follows.

Definition 1 (Measurement Matrix): Let D be an $m \times m$ positive diagonal matrix whose diagonal entry is in proportion[1] to the inverse of the reactance of each arc. The *measurement matrix* H is an $(m + n) \times n$ matrix defined by

$$H := \begin{pmatrix} DB_G^\top \\ B_G D B_G^\top \end{pmatrix}. \quad (1)$$

[1]The factor is the square of the effective voltage, which is assumed to be the same constant (usually regarded as one unit) for all nodes.

Note that the row and column sets are $A \cup V$ and V, respectively.

For a vector $\theta \in \mathbb{R}^V$ that represents the states of all nodes, the measurement values at all measurement points are represented as $z = H\theta \in \mathbb{R}^{A\cup V}$. A nonzero vector $\Delta z \in \mathbb{R}^{A\cup V}$ is called *undetectable* if there exists $\Delta\theta \in \mathbb{R}^V$ such that $\Delta z = H\Delta\theta$. It is shown in [6] that an undetectable additive measurement value Δz does not affect the difference between the observed measurement values and their corresponding estimates and no alarm is triggered in the end. Nobody can detect only by the measurement values whether the system is attacked or not. Such a new class of attacks is known as false data injection attacks [6].

The support $\text{supp}(\Delta z) := \{k \mid k \in A \cup V,\ (\Delta z)_k \neq 0\}$ of an undetectable attack Δz is called an *attackable set*. Let $\|x\|_0$ denote the size of the support of a vector x. The *sparsest attack problem* is to find an undetectable attack with the minimum support size, that is

$$\min_{\Delta\theta\in\mathbb{R}^V} \{\|H\Delta\theta\|_0 \mid H\Delta\theta \neq \mathbf{0}\}. \quad (2)$$

The attacker would be interested in finding a sparsest attack, which is obtained from an optimal solution of the sparsest attack problem, because each attack on a measurement point involves risk to be found. Therefore, a minimum attackable set indicates one of the most vulnerable sets of measurement points to be attacked in the system.

Let $k \in A \cup V$ be a measurement point. Suppose that k is attacked in an undetectable attack Δz, i.e., $k \in \text{supp}(\Delta z)$. The minimum size of an attackable set that contains k is called the *security index* [14] of k. The *security index problem* is to compute the security index of a given measurement point k for a given measurement matrix H, i.e., to compute

$$\min_{\Delta\theta\in\mathbb{R}^V} \{\|H\Delta\theta\|_0 \mid H_k\Delta\theta \neq 0\} \quad (3)$$

where H_k denotes the row vector of H indexed by k.

Let us denote by $G' = (V, E)$ the underlying graph of G, i.e., G' is an undirected graph with edge set $E = \{e_a := \{u, v\} \mid a = uv \in A \text{ or } a = vu \in A\}$ being a multiset. For each node $v \in V$, we denote by $\Gamma_G(v)$ the set of nodes adjacent to v in G' and by $\delta_G(v)$ the set of edges incident to v in G', i.e., $\Gamma_G(v) := \{u \mid \{u, v\} \in E\}$ and $\delta_G(v) := \{e \mid v \in e \in E\}$.

For each arc $a \in A$, let D_a denote the corresponding diagonal entry of D. Then, each entry of the measurement matrix H is given as follows. For each arc $a = uv \in A$, we have $H_{au} = D_a$, $H_{av} = -D_a$, and $H_{aw} = 0$ for every $w \in V \setminus \{u, v\}$. For each node $v \in V$, we have $H_{vv} = \sum_a \{D_a \mid e_a \in \delta_G(v)\}$. For each pair of distinct nodes $u, v \in V$, we have $H_{vu} = -\sum_a \{D_a \mid e_a = \{u, v\} \in E\}$.

Observation 1: For any vector $\Delta\theta \in \mathbb{R}^V$, $X := \text{supp}(H\Delta\theta)$ satisfies the following properties.
1) For each $a = uv \in A$, we have $a \in X$ if and only if $(\Delta\theta)_u \neq (\Delta\theta)_v$.
2) For each node $v \in V$, we have $v \in X$ if and only if $\sum_a \{(\Delta\theta)_v D_a \mid e_a \in \delta_G(v)\} \neq \sum_{u\in\Gamma_G(v)} \sum_a \{(\Delta\theta)_u D_a \mid u \in e_a \in \delta_G(v)\}$.

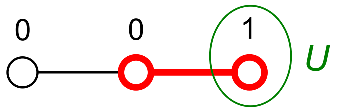


Fig. 1. Example of elementary attacks. The number above each node is the entry of the characteristic vector χ_U. The nodes and arcs in $X = \mathrm{supp}(H\chi_U)$ (i.e., to be attacked) are emphasized with thick lines.

B. Elementary Attacks

Sou *et al.* [14] pointed out a nice property of the security index problem. The following lemma, extracting the core of the property, leads to reduction of the above two problems to hypergraph cut problems, which will be shown later in Section III-B. Let us denote by $\chi_U \in \mathbb{R}^V$ the characteristic vector of a subset $U \subseteq V$, i.e., $\chi_U(u) = 1$ ($\forall u \in U$) and $\chi_U(v) = 0$ ($\forall v \in V \setminus U$).

Lemma 1: For any $\Delta\theta \in \mathbb{R}^V$ with $H\Delta\theta \neq \mathbf{0}$ and any $\alpha \in \mathbb{R}$ with $\min_{v \in V}(\Delta\theta)_v < \alpha \leq \max_{v \in V}(\Delta\theta)_v$, $U := \{v \in V \mid (\Delta\theta)_v \geq \alpha\}$ satisfies $0 < \|H\chi_U\|_0 \leq \|H\Delta\theta\|_0$.

Remark 1: This lemma can be seen in the same way as the proof of [14, Proposition 1], and here, we omit the proof.

Lemma 1 claims that each minimum attackable set has some corresponding assignments of 0 or 1 to each node (i.e., bipartitions of the node set), though the corresponding attackable set itself does not coincide with the original one in general. Hence, the sparsest attack problem (2) and the security index problem (3) are interpreted as node-bipartition problems since these problems are interested only in the sparsest attacks.

Let us call an undetectable attack Δz an *elementary attack* if $\Delta z = H\chi_U$ for some proper nonempty subset U of V. The following observation is easily seen from Observation 1.

Observation 2: For any $U \subseteq V$, $X := \mathrm{supp}\,(H\chi_U)$ satisfies the following properties (cf. Fig. 1).

1) For each arc $a = uv \in A$, we have $a \in X$ if and only if $|U \cap \{u, v\}| = 1$.
2) For each node $v \in V$, we have $v \in X$ if and only if $|U \cap \{u, v\}| = 1$ for some neighbor $u \in \Gamma_G(v)$.

III. Reduction to Minimum Cuts in Hypergraphs

A. Preliminaries for Hypergraphs

A pair $\mathcal{H} = (V, \mathcal{E})$ of a finite set V and a family $\mathcal{E} \subseteq 2^V$ of subsets of V is called a *hypergraph*. Each element $v \in V$ is called a *node*, and each element $e \in \mathcal{E}$ is called a *hyperedge*. Note that each hyperedge $e \in \mathcal{E}$ is a subset of V, and that if every hyperedge is of size 2, then the hypergraph is just an undirected graph which contains no self-loop. To measure the size of $\mathcal{H}$, we use $\|\mathcal{E}\| := \sum_{e \in \mathcal{E}} |e|$ as well as $|V|$.

Let $\mathcal{H} = (V, \mathcal{E})$ be a hypergraph, and $c\colon \mathcal{E} \to \mathbb{R}_{\geq 0}$ a nonnegative function on the hyperedges. We call the pair $\mathcal{N} = (\mathcal{H}, c)$ a *hypernetwork*. A proper nonempty subset U of V is called a *cut* in $\mathcal{H}$ (or in $\mathcal{N}$). The *capacity* of a cut U is defined by

$$\kappa_{\mathcal{N}}(U) := \sum_{e \in \delta_{\mathcal{H}}(U)} c(e)$$

where $\delta_{\mathcal{H}}(U) := \{e \in \mathcal{E} \mid e \cap U \neq \emptyset \neq e \setminus U\}$ denotes the set of hyperedges in $\mathcal{H}$ across U and $V \setminus U$. For each node $v \in V$, we simply denote $\kappa_{\mathcal{N}}(\{v\})$ by $\kappa_{\mathcal{N}}(v)$. The *hypergraph minimum cut problem* is to find a cut in a given hypernetwork $\mathcal{N}$ with the minimum capacity. Let $\lambda(\mathcal{N})$ denote the minimum capacity.

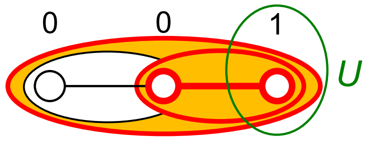


Fig. 2. Example of computing the size of $X := \mathrm{supp}(H\chi_U)$ as the capacity $\kappa_{\mathcal{N}}(U)$ in the auxiliary hypernetwork $\mathcal{N}$. The number above each node is the entry of the characteristic vector χ_U. The nodes and arcs in X (i.e., to be attacked), and the hyperedges cut by U are emphasized.

Based on the concept of MA-ordering, introduced by Nagamochi and Ibaraki [8] for graphs, Klimmek and Wagner [4] presented a simple algorithm for finding a minimum cut in hypernetwork $\mathcal{N}$ in $O(|V| \cdot \|\mathcal{E}\| + |V|^2 \log |V|)$ time.

For two distinct nodes $s, t \in V$, a cut U is called an *s–t cut* if $s \in U$ and $t \notin U$. The hypergraph minimum s–t cut problem is to find an s–t cut in a given hypernetwork $\mathcal{N}$ with the minimum capacity for given distinct nodes $s, t \in V$.

The *hypergraph minimum s–t cut problem* can be reduced, in general, to the minimum s–t cut problem on directed graphs [16], which is a fundamental graph optimization problem solved by maximum flow algorithms. In particular, if the capacity of every hyperedge is equal to one, we can apply a simple algorithm due to Pistorius and Minoux [12]. Its running time is bounded by $O(C \cdot \|\mathcal{E}\|)$, where C is the maximum flow value, which is equal to the minimum s–t cut capacity.

B. Reduction From Cyber Security Analysis Problems

Recall that, in the sparsest attack problem (2) and the security index problem (3), it suffices to consider the elementary attacks. In other words, we have to consider only the values of

$$f(U) := \|H\chi_U\|_0 = |\mathrm{supp}\,(H\chi_U)|$$

over all proper nonempty subset U of V, i.e., over all cuts U in G' (recall that $G' = (V, E)$ denotes the underlying graph of the input graph G). Moreover, by Observation 2, it can be rewritten as

$$f(U) = \kappa_G(U) + |V(\delta_G(U))|$$

where κ_G denotes $\kappa_{\mathcal{N}'}$ for $\mathcal{N}' = (G', \mathbf{1})$ and $\mathbf{1}$ denotes the all one vector. Define $\mathcal{E} := E \cup \{\Gamma_G(v) \cup \{v\} \mid v \in V\}$ and $\mathcal{N} := ((V, \mathcal{E}), \mathbf{1})$ (recall that $\Gamma_G(v)$ denotes the set of nodes adjacent to v in G'). Then, for each cut U in $\mathcal{N}$, we have $f(U) = \kappa_{\mathcal{N}}(U)$ (cf. Fig. 2) since, for each additional hyperedge $e_v := \Gamma_G(v) \cup \{v\}$, we have $\emptyset \neq e_v \cap U \neq e_v$ if and only if $|U \cap e| = 1$ for some $e \in \delta_G(v)$.

The sparsest attack problem immediately reduces to the hypergraph minimum cut problem by the above construction of $\mathcal{N}$. Moreover, the security index problem reduces to the hypergraph minimum s–t cut problem as follows. For an arc $uv \in A$, then let $s := u$ and $t := v$. For a node $v \in V$, then solve the security index problem of arcs $uv \in A$ or $vu \in A$ for all neighbors $u \in \Gamma_G(v)$, and take the minimum among the obtained security indices.

Since each edge in E (obtained from A) is of size 2 and each additional hyperedge e_v in $\mathcal{E} \setminus E$ is of size $|\Gamma_G(v)| + 1$, we have

$$\|\mathcal{E}\| = \sum_{a \in A} 2 + \sum_{v \in V} (|\Gamma_G(v)| + 1) \leq 2|A| + \sum_{v \in V} (|\delta_G(v)| + 1)$$

where $\delta_G(v)$ denotes the set of edges in E incident to v. Note that $\sum_{v \in V} |\delta_G(v)|$ counts each edge in E exactly twice, and hence the value is equal to $2|A|$. Thus, we have $\|\mathcal{E}\| \leq 4|A| + |V| = O(m)$ (recall that we naturally assume that G is connected, which leads to $n = O(m)$, where $n = |V|$ and $m = |A|$).

Using the algorithm of Klimmek and Wagner [4] for finding a minimum cut in a hypergraph (see the Appendix), we obtain the following result. The computational time bound of our method is essentially better than that of computing the security indices of all arcs by the existing exact method [2], which is $O(nm^2)$.

Theorem 1: Given a measurement matrix H, one can find a sparsest attack $\Delta z \in \mathbb{R}^{A \cup V}$ in $O(nm + n^2 \log n)$ time.

Remark 2: Hendrickx *et al.* [2] claimed that the security index problem can be solved by finding a minimum s–t cut in an auxiliary directed graph $\tilde{G}$. Based on this fact, one may consider that the sparsest attack problem can be solved by finding a minimum cut in $\tilde{G}$, but this is not true for the following reason. The auxiliary graph $\tilde{G}$ has three times more nodes than the input graph G. Because of the additional nodes, $\tilde{G}$ always contains a cut with capacity 1 which does not correspond to any elementary attack. Such a cut is always a minimum cut in $\tilde{G}$, while every cut in $\tilde{G}$ that corresponds to some elementary attack is with capacity at least three since the attackable set of each elementary attack contains at least one arc and both of its ends.

Recall that we have $|V| = n$ and $\|\mathcal{E}\| = O(m)$, and we can assume the maximum value of an s–t flow in $\mathcal{N}$ to be $O(n)$ for any distinct $s, t \in V$ as follows. Since the underlying graph has at most constant number of parallel edges at each pair of two nodes in practice, the maximum value of an s–t flow in G' is $O(n)$ (which is obviously at most $\kappa_G(s)$, i.e., the number of arcs incident to s). Moreover, the number of additional hyperedges is n.

Thus the computational time can be bounded as follows.

Theorem 2: Given a measurement matrix H and a measurement point $k \in A \cup V$, one can compute the security index of k in $O(nm)$ time if k is an arc, and in $O(nm|\Gamma_G(v)|)$ if k is a node $v \in V$.

Our reduction works well in a more general setting: when measurement points have different costs to be attacked and the sparsity of each undetectable attack is evaluated by the total cost. This is a natural generalization as discussed also in [2]. Even in such a situation, a sparsest elementary attack can be found by finding a minimum cut in a modified auxiliary hypernetwork discussed below. Moreover, under a certain condition, it is also the sparsest among all undetectable attacks.

It should be noted that this setting includes the following situations: when some arcs and nodes do not have their measurement points, and when some measurement points are protected so that they cannot be attacked. These situations can be represented by using a sufficiently large value $M \in \mathbb{R}$. Let $P \subseteq A \cup V$ be the set of arcs and nodes with no measurement point, and $Q \subseteq A \cup V$ the set of measurement points which are protected. Then, by defining the cost to attack each $k \in P$ as 0 and to attack each $k \in Q$ as M, these situations reduce to the generalized cost setting.

Let $c\colon A \cup V \to \mathbb{R}_{\geq 0}$ be a cost function, which represents the cost to attack each measurement point. The attack on $k \in A \cup V$ takes the cost $c(k)$, and the goal is to find a sparsest attack or to compute the security index of a given measurement point in terms of the total cost, i.e., the objective function $\|H\Delta\theta\|_0$ to be minimized over $\Delta\theta \in \mathbb{R}^V$ is replaced by $\sum_{k \in \text{supp}(H\Delta\theta)} c(k)$. We then construct a hypernetwork $\mathcal{N}' = (\mathcal{H}, c')$ from $\mathcal{N} = (\mathcal{H}, \mathbf{1})$ by replacing its capacity function $\mathbf{1}$ with a nonnegative function $c'\colon \mathcal{E} \to \mathbb{R}_{\geq 0}$ defined as follows: $c'(e_a) := c(a)$ for each arc $a \in A$ and $c'(e_v) := c(v)$ for each node $v \in V$ (recall that $e_a \in E \subseteq \mathcal{E}$ denotes the undirected edge corresponding to a, and that $e_v = \Gamma_G(v) \cup \{v\} \in \mathcal{E} \setminus E$). This modification leads to a reduction of finding a sparsest (minimum cost) elementary attack to finding a minimum cut in $\mathcal{N}'$.

Finally, we give a sufficient condition for a sparsest elementary attack to be the sparsest also among all undetectable attacks in this generalized setting. If $c(v) \leq c(a)$ holds for every pair of a node $v \in V$ and an incident edge $e_a \in \delta_G(v)$, then an analogous claim to Lemma 1 holds (it is not difficult to see that the same proof goes on with respect to the new objective function with the cost function c). Under this condition, Theorem 1 holds as it does. Furthermore, Theorem 2 also carries over this setting with the aid of an advanced maximum flow algorithm [11].

IV. Experiments

We compare our new methods, `hyp.global min.cut` for the sparsest attack problem (2) and `hyp.min.s-t cut` for the security index problem (3), with several existing methods.

1) `min.s-t cut exact` [2]: Find a minimum s–t cut in an auxiliary directed graph, which leads to the security index.
2) `min.s-t cut relax` [14]: Find a minimum s–t cut in the original graph to give an upper bound on the security index.
3) `L1-relax (LP)` [15]: Solve an LP with the ℓ_1-norm objective instead of the ℓ_0-norm in (3).
4) `L0-exact (MIP)`: Solve (3) as a mixed integer programming problem.

We applied these methods to ten power network instances: one real-world data set (case 6) in eastern Japan obtained from [10] and nine data sets obtained from [17] (cases 1–5 are standard IEEE benchmarks, and cases 7–10 are real-world data in Poland). We executed `hyp.min.s-t cut`, `min.s-t cut exact`, `min.s-t cut relax`, and `L1-relax (LP)` for all arcs. We also applied `L0-exact (MIP)` to all arcs for cases 1–6, and to 30 arcs which are randomly chosen for cases 7–10. All computations were done using Python

TABLE I
ACCURACY OF THE TWO RELAXATION METHODS. THE FIRST ROW "ERROR" IN EACH METHOD REPRESENTS THE RATIO OF FAILURE IN OBTAINING THE EXACT SECURITY INDEX. THE SECOND ROW "APPROX." SHOWS THE GEOMETRIC MEAN OF THE RATIOS BETWEEN UPPER BOUNDS OBTAINED BY EACH RELAXATION METHOD AND EXACT SECURITY INDICES

case		1	2	3	4	5	6	7	8	9	10
number of nodes		9	14	30	118	300	246	2383	2746	3012	3375
number of arcs		9	20	41	186	411	682	2896	3514	3572	4161
`min. s-t cut relax`	error	0.00	0.150	0.220	0.210	0.122	0.116	0.124	0.154	0.117	0.112
	approx.	1.000	1.025	1.040	1.032	1.019	1.013	1.022	1.027	1.021	1.020
`L1-relax (LP)`	error	0.667	0.600	0.683	0.726	0.599	0.733	0.644	0.738	0.676	0.655
	approx.	1.368	1.488	1.476	1.813	1.625	2.030	1.442	1.364	1.340	1.345

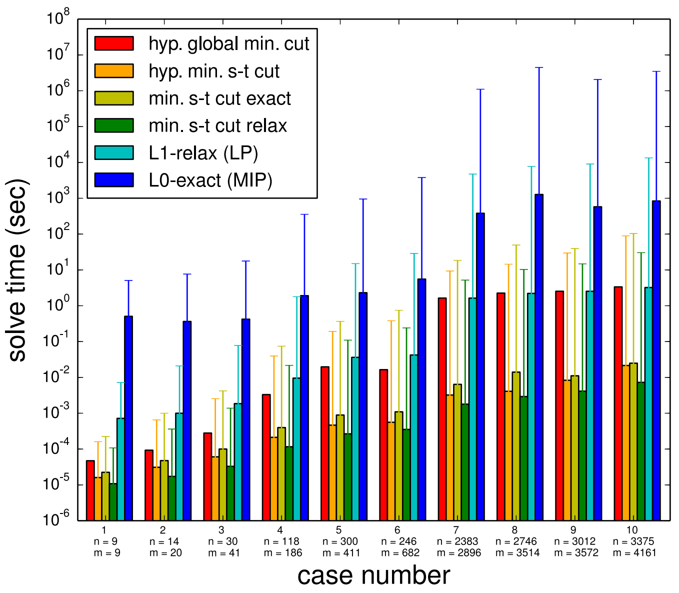


Fig. 3. Experimental results on the running time of the six methods applied to nine benchmarks. Each bar represents the mean running time of each method, and the segment on each bar corresponds to the number of arcs, so that its top represents the total computational time required to find a sparsest attack by computing the security indices for all arcs (or the estimated computational time for the application of `L0-exact (MIP)` to cases 7–10).

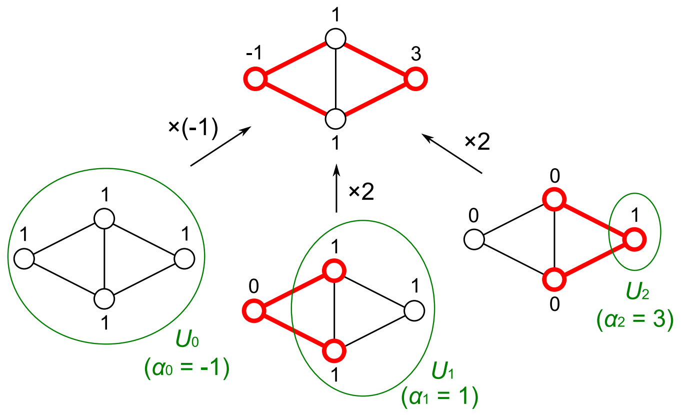


Fig. 4. Example of the representation of an undetectable attack as the nonnegative linear combination of elementary attacks (note that $H\chi_{U_0} = H\mathbf{1} = \mathbf{0}$). The nodes and arcs to be attacked are emphasized with thick lines. We assume that the positive diagonal matrix D is equal to the identity matrix.

(and internally C++) on a Mac desktop with 3.1 GHz Intel CPU Core i7 and 16 GB of memory. CPLEX was used for `L1-relax (LP)` and `L0-exact (MIP)`.

Fig. 3 illustrates the experimental result on the running time of each method. For the security index problem, `hyp. min. s-t cut` as well as `min. s-t cut exact` and `min. s-t cut relax` runs substantially faster than `L1-relax (LP)` and `L0-exact (MIP)`. Furthermore, our `hyp. min. s-t cut` is faster than `min. s-t cut exact` [2] for all instances (about 1.74 times faster on the geometric average). Recall that the sparsest attack problem can be solved by computing the security indices for all measurement points and adopting the minimum among them. A single execution of `hyp. global min. cut` is much faster than this approach.

Table I shows the accuracy of the two relaxation methods. As shown in Fig. 3, `min. s-t cut relax` is the fastest among the five methods for the security index computation. In Table I, this method appears to perform better than `L1-relax (LP)`, but yet it sometimes fails in obtaining an optimal solution.

V. ENUMERATION OF SPARSE ATTACKS

A. *Significance of Elementary Attacks*

Elementary attacks (see Section II-B) play an important role also in considering sparse undetectable attacks that are not necessarily the sparsest.

Recall that Lemma 1 does not depend on the threshold α, i.e., for any undetectable attack $\Delta z = H\Delta\theta$ and any α with $\min_{v\in V}(\Delta\theta)_v < \alpha \le \max_{v\in V}(\Delta\theta)_v$, $U(\alpha) := \{v \in V \mid (\Delta\theta)_v \ge \alpha\}$ satisfies $\|H\chi_{U(\alpha)}\|_0 \le \|\Delta z\|_0$. It is easy to see that, for any $\alpha_1, \alpha_2 \in \mathbb{R}$ with $\alpha_1 \le \alpha_2$, we have $U(\alpha_1) \supseteq U(\alpha_2)$. Hence, by changing α in the domain $[\,\min_{v\in V}(\Delta\theta)_v, \max_{v\in V}(\Delta\theta)_v\,]$, we get a nested family $V = U_0 \supsetneq U_1 \supsetneq \cdots \supsetneq U_r \neq \emptyset$ such that $0 < \left\|H\chi_{U_j}\right\|_0 \le \|\Delta z\|_0$ for each U_j with $1 \le j \le r$ (note that $H\chi_{U_0} = H\mathbf{1} = \mathbf{0}$), where $r + 1$ is the number of different entries of $\Delta\theta$.

For each $j = 0, 1, \ldots, r$, let α_j be the $(j + 1)$th smallest entry of $\Delta\theta$. Then, as seen in Fig. 4, $\Delta\theta$ is written as

$$\Delta\theta = \alpha_0\chi_{U_0} + \sum_{j=1}^{r}(\alpha_j - \alpha_{j-1})\chi_{U_j}.$$

Note that $\chi_{U_0} = \chi_V = \mathbf{1}$ and recall that $H\mathbf{1} = \mathbf{0}$ (see Observation 2). Hence, we have

$$\Delta z = H\Delta\theta = \sum_{j=1}^{r}(\alpha_j - \alpha_{j-1})H\chi_{U_j}.$$

The above discussion implies that any undetectable attack Δz can be written as the nonnegative linear combination of elementary attacks $H\chi_U$ with $0 < \|H\chi_U\|_0 \le \|\Delta z\|_0$. In other words, for any positive integer β and the set Z_β of all undetectable attacks Δz with $\|\Delta z\|_0 \le \beta$, we have

$$Z_\beta \subseteq \left\{ \sum_{U:\, \emptyset \ne U \subsetneq V} \alpha_U H\chi_U \;\middle|\; \begin{array}{l} \alpha_U \ge 0 \ (\emptyset \ne U \subsetneq V) \text{ and} \\ \alpha_U = 0 \text{ if } \|H\chi_U\|_0 > \beta \end{array} \right\}.$$

Based on this fact, what is important for finding sparse undetectable attacks is to find all sparse elementary attacks. The problem of *enumerating all sparse elementary attacks* is to find all elementary attacks Δz with $\|\Delta z\|_0 \le \beta$ for a given measurement matrix and a positive integer β. By the same argument as Section III-B, this problem can be interpreted as to enumerate all small cuts in a given hypergraph. We propose a combinatorial algorithm for this problem, whose overview is shown in the next section.

B. Overview of Enumerating Small Cuts in a Hypergraph

Our goal is to find all cuts U in a given hypernetwork $\mathcal{N}$ with $\kappa_{\mathcal{N}}(U) \le k\lambda(\mathcal{N})$, called *k-small cuts*, for a given real $k \ge 1$ (recall that $\lambda(\mathcal{N})$ denotes the minimum capacity of a cut in $\mathcal{N}$). Based on the idea of the algorithm of Nagamochi *et al.* [9] for enumerating all small cuts in an undirected graph, we devise a new algorithm for enumerating all small cuts in an undirected hypergraph.

Let $\beta := k\lambda(\mathcal{N})$. The overview of the algorithm is as follows. Construct a sequence $\mathcal{N} = \mathcal{N}_n, \mathcal{N}_{n-1}, \ldots, \mathcal{N}_1$ of hypernetworks by isolating and removing a node in each step so that $\mathcal{N}_1$ consists of a single node, and for each $i = 2, 3, \ldots, n$, $\lambda(\mathcal{N}_i) \le \lambda(\mathcal{N}_{i-1})$ and $\kappa_{\mathcal{N}_i}(U) \ge \kappa_{\mathcal{N}_{i-1}}(U)$ for every cut U in $\mathcal{N}_{i-1}$. After the construction, repeat checking the capacity $\kappa_{\mathcal{N}_i}(U)$ of each cut U in $\mathcal{N}_{i-1}$ with $\kappa_{\mathcal{N}_{i-1}}(U) \le \beta$ and the capacity $\kappa_{\mathcal{N}_i}(\{v_i\})$, where v_i is the vertex removed from $\mathcal{N}_i$, i.e., $\{v_i\} = V(\mathcal{N}_i) \setminus V(\mathcal{N}_{i-1})$, and maintain the set of all cuts U in $\mathcal{N}_i$ with $\kappa_{\mathcal{N}_i}(U) \le \beta$. This can be done, since the cuts U and $V(\mathcal{N}_i) \setminus U$ has the same capacity in $\mathcal{N}_i$ and hence it suffices to check one of each two such cuts.

The key idea of our extension is to separate the operation called edge-splitting, which is used to isolate a node in the original algorithm [9], into two types of operations, which we call 1-hyperedge-splitting and 2-hyperedge-splitting.

Finally, we should mention the running time bound of our algorithm. For each $i = 1, 2, \ldots, n$, let $h_k(\mathcal{N}, i)$ denote the number of cuts with capacity at most $\beta = k\lambda(\mathcal{N})$ in $\mathcal{N}_i$, which is constructed through the algorithm, and let $h_k(\mathcal{N}) := \max_i h_k(\mathcal{N}, i)$. One can see that the number of k-small cuts in a hypernetwork $\mathcal{N}$ is $O(n^{kM})$ (i.e., $h_k(\mathcal{N}, n) = O(n^{kM})$), where M denotes the maximum size of a hyperedge in $\mathcal{N}$, as a straightforward extension of Karger's bound [3] for an undirected graph. Our hyperedge-splitting, however, may increase M in constructing $\mathcal{N}_i$ $(i = n-1, n-2, \ldots, 1)$, which makes it difficult to get a simple bound on $h_k(\mathcal{N})$.

Theorem 3: Given a hypernetwork $\mathcal{N} = (\mathcal{H} = (V, \mathcal{E}), c)$ and a real $k \ge 1$, one can enumerate all cuts U in $\mathcal{N}$ with $\kappa_{\mathcal{N}}(U) \le k\lambda(\mathcal{N})$ in $O(|V| \cdot \|\mathcal{E}\|^2 + (h_k(\mathcal{N}) + |V|^2) \cdot \|\mathcal{E}\| \log |V|)$ time.

Remark 3: If $\mathcal{N}$ consists of only one hyperedge containing all nodes, then every cut has the same capacity, which is the minimum. This means that a hypergraph (or a directed graph) may have exponentially many minimum cuts, and hence there is no polynomial-time algorithm to enumerate all small cuts in general. However, it is worth considering whether there exists such an algorithm when the size of hyperedges is bounded by a constant.

In the following sections, we present our algorithm with introducing some concepts and lemmas, which are described analogously to [9]. For full proofs of the lemmas and Theorem 3, see the Appendix.

C. Computing r-Connectivity

For a hypernetwork $\mathcal{N} = (\mathcal{H} = (V, \mathcal{E}), c)$, choose a node $r \in V$ as a designated node. A cut $U \subset V$ is called *r-proper* if U is a cut in $\mathcal{H} - \{r\}$, i.e., $r \notin U$ and $U \ne V - r$. The *r-connectivity* $\lambda_r(\mathcal{N})$ is defined as the minimum capacity of an r-proper cut in $\mathcal{N}$. Hence, we have $\lambda(\mathcal{N}) = \min\{\lambda_r(\mathcal{N}), \kappa_{\mathcal{N}}(\{r\})\}$, since for each cut U that is not r-proper except for $\{r\}$, the complement $V \setminus U$ is r-proper and has the same capacity as U. An r-proper cut U is called *r-tight* if $\kappa_{\mathcal{N}}(U) = \lambda_r(\mathcal{N})$.

Our algorithm often requires an r-tight cut, which can be found by a single application of the algorithm of Klimmek and Wagner [4] for finding a minimum cut in a hypergraph. This fact leads to the following lemma.

Lemma 2: For any hypernetwork $\mathcal{N} = (\mathcal{H} = (V, \mathcal{E}), c)$ with a designated node $r \in V$, the r-connectivity $\lambda_r(\mathcal{N})$ and an r-tight cut can be computed in $O(|V| \cdot \|\mathcal{E}\| + |V|^2 \log |V|)$ time.

D. Weighted Hyperedge-Splitting

In this section, we introduce two types of procedures called hyperedge-splitting, which are used for isolating a designated node. Let $\mathcal{N} = (\mathcal{H} = (V, \mathcal{E}), c)$ be a hypernetwork and $r \in V$ a designated node. Here we assume that $\mathcal{H}$ is a complete hypergraph, whose hyperedge set $\mathcal{E}$ is $\{e \subseteq V \mid |e| \ge 2\}$. This setting does not lose the generality since we can deal with each hyperedge $e \in \mathcal{E}$ as if it did not exist by defining $c(e) = 0$. Let us define $\delta_{\mathcal{N}}(U) := \{e \in \delta_{\mathcal{H}}(U) \mid c(e) > 0\}$ for each $U \subseteq V$ and $\Gamma_{\mathcal{N}}(v) := \{u \in V - v \mid u \in e \in \delta_{\mathcal{N}}(v)\}$ for each $v \in V$.

First, given a hyperedge $e_1 \in \delta_{\mathcal{N}}(r)$ and a nonnegative real $\alpha \le \alpha_{\max} := c(e_1)/2$, we construct a hypernetwork $\mathcal{N}' = (\mathcal{H}, c')$ as follows:

$$c'(e_1) := c(e_1) - 2\alpha,\ c'(e') := c(e') + 2\alpha,$$
$$c'(e) := c(e) \quad (\forall e \in \mathcal{E} \setminus \{e_1, e'\}),$$

where $e' = e_1 - r$, and if $|e'| = 1$ (i.e., $|e_1| = 2$) then there is no update for e'. We say that $\mathcal{N}'$ is obtained from $\mathcal{N}$ by *1-hyperedge-splitting* e_1 of weight α, and denote the resulting hypernetwork $\mathcal{N}'$ by $\mathcal{N}/(e_1, \alpha)$.

Second, given two hyperedges $e_1, e_2 \in \delta_{\mathcal{N}}(r)$ with $e_1 \cap e_2 = \{r\}$ and a nonnegative real $\alpha \le \alpha_{\max} := \min\{c(e_1), c(e_2)\}$, we construct a hypernetwork $\mathcal{N}' = (\mathcal{H}, c')$ as follows:

$$c'(e_1) := c(e_1) - \alpha,\ c'(e_2) := c(e_2) - \alpha,\ c'(e') := c(e') + \alpha,$$
$$c'(e) := c(e) \quad (\forall e \in \mathcal{E} \setminus \{e_1, e_2, e'\}),$$

where $e' = e_1 \triangle e_2 := (e_1 \setminus e_2) \cup (e_2 \setminus e_1) = e_1 \cup e_2 - r$. We say that $\mathcal{N}'$ is obtained from $\mathcal{N}$ by *2-hyperedge-splitting* e_1 and e_2 of weight α, and denote the resulting hypernetwork $\mathcal{N}'$ by $\mathcal{N}/(e_1, e_2, \alpha)$.

Observation 3: After hyperedge-splitting (e_1, α) or (e_1, e_2, α) (in the case of 1-hyperedge-splitting, ignore the second case), for any cut U with $r \notin U$ in $\mathcal{H}$, we have

$$\kappa_{\mathcal{N}'}(U) = \begin{cases} \kappa_{\mathcal{N}}(U) - 2\alpha & (e' \subseteq U) \\ \kappa_{\mathcal{N}}(U) - \alpha & \begin{pmatrix} e_1 \cap U \neq \emptyset \neq e_2 \cap U \\ \text{and } e' \setminus U \neq \emptyset \end{pmatrix} \\ \kappa_{\mathcal{N}}(U) & (\text{otherwise}). \end{cases} \quad (4)$$

This observation implies that the capacity of each r-proper cut does not increase by any hyperedge-splitting. Hence, we have $\lambda_r(\mathcal{N}') \leq \lambda_r(\mathcal{N})$ for any $\alpha \leq \alpha_{\max}$. Let $\alpha_r(e_1; \mathcal{N})$ and $\alpha_r(e_1, e_2; \mathcal{N})$ denote the maximum α such that $\alpha \leq \alpha_{\max}$ and $\lambda_r(\mathcal{N}') = \lambda_r(\mathcal{N})$, i.e., any r-tight cut in $\mathcal{N}$ remains r-tight in $\mathcal{N}/(e_1, \alpha)$ and $\mathcal{N}/(e_1, e_2, \alpha)$ if $0 \leq \alpha \leq \alpha_r(e_1; \mathcal{N})$ and $0 \leq \alpha \leq \alpha_r(e_1, e_2; \mathcal{N})$, respectively.

The following lemma claims that the values of $\alpha_r(e_1; \mathcal{N})$ and $\alpha_r(e_1, e_2; \mathcal{N})$ and a critical r-proper cut can be computed efficiently. This is based on Observation 3, which implies that there are at most two types of r-proper cuts whose capacities decrease by a hyperedge-splitting. Hence, it suffices to find an r-tight cut in at most two hypernetworks obtained from $\mathcal{N}$ by hyperedge-splitting of at most two distinct weight.

Lemma 3: Let $\mathcal{N} = (\mathcal{H} = (V, \mathcal{E}), c)$ be a hypernetwork with a designated node $r \in V$. Then, for any hyperedge $e_1 \in \delta_{\mathcal{N}}(r)$ (or any two hyperedges $e_1, e_2 \in \delta_{\mathcal{N}}(r)$ with $e_1 \cap e_2 = \{r\}$).

1) $\alpha_r(e_1; \mathcal{N})$ (or $\alpha_r(e_1, e_2; \mathcal{N})$) can be computed in $O(|V| \cdot \|\mathcal{E}\| + |V|^2 \log |V|)$ time.
2) If $c'(e_1) > 0$ (or $c'(e_1) > 0$ and $c'(e_2) > 0$), where c' is the capacity function of $\mathcal{N}' := \mathcal{N}/(e_1, \alpha_r(e_1; \mathcal{N}))$ (or $\mathcal{N}' := \mathcal{N}/(e_1, e_2, \alpha_r(e_1, e_2; \mathcal{N}))$), then $\mathcal{N}'$ has an r-tight cut T such that $e' \subseteq T$ (or $e_1 \cap T \neq \emptyset \neq e_2 \cap T$). Which can be found in $O(|V| \cdot \|\mathcal{E}\| + |V|^2 \log |V|)$ time.

The next lemma shows an important property of r-tight cuts, which is used in the algorithm to isolate a designated node $r \in V$ shown in the next section. The proof is simply done by contradiction.

Lemma 4: For any hypernetwork $\mathcal{N} = (\mathcal{H} = (V, \mathcal{E}), c)$ with a designated node $r \in V$ with $\Gamma_{\mathcal{N}}(r) \neq \emptyset$ and any r-tight cuts T and T' in $\mathcal{N}$, we have the following properties.

1) $\Gamma_{\mathcal{N}}(r) \setminus T \neq \emptyset$.
2) If there exists a hyperedge $e \in \delta_{\mathcal{N}}(r)$ such that $e \cap T' \neq \emptyset$, $e \setminus T' \neq \{r\}$, and $e \subseteq T + r$, then $T' \subsetneq T$.
3) If there exist two hyperedges $e_1, e_2 \in \delta_{\mathcal{N}}(r)$ such that $e_1 \cap e_2 = \{r\}$, $e_1 \subseteq T' + r$, $e_2 \cap T' = \emptyset$, $e_1 \cap T \neq \emptyset$, and $e_2 \cap T \neq \emptyset$, then $T' \subsetneq T$.

E. Algorithm to Isolate a Node

Our algorithm to isolate a designated node $r \in V$ is shown as Algorithm 1. The idea is simple: to repeat hyperedge-splitting while there exists a hyperedge $e \in \delta_{\mathcal{N}^*}(r)$ incident to r in the temporary hypernetwork $\mathcal{N}^*$. A more detailed explanation of Algorithm 1 is described later.

Algorithm 1 Node Isolation Technique

```
Input: A hypernetwork $\mathcal{N} = (\mathcal{H} = (V, \mathcal{E}), c)$ and $r \in V$.
Output: A hypernetwork $\mathcal{N}_r = ((V - r, \mathcal{E}'), c')$ with the
    conditions (i) and (ii) in Lemma 5 and a set $Q_r$ of the
    information of all hyperedge-splittings performed.
1: $\mathcal{N}^* \leftarrow \mathcal{N}$, $T^* \leftarrow \emptyset$, $Q_r \leftarrow \emptyset$.
2: while $|\delta_{\mathcal{N}^*}(r)| > 1$ do
3:     if $T^* \cap \Gamma_{\mathcal{N}^*}(r) = \emptyset$ then
4:         $T^* \leftarrow \{u\}$ for some $u \in \Gamma_{\mathcal{N}^*}(r)$.
5:     end if
6:     if $\exists e \in \delta_{\mathcal{N}^*}(r)$ s.t. $e \cap T^* \neq \emptyset$ and $e \setminus T^* \neq \{r\}$ then
7:         Take such $e$, and compute $\alpha := \alpha_r(e; \mathcal{N}^*)$.
8:         $\alpha_{\max} \leftarrow c(e)/2$.
9:         $\mathcal{N}^* \leftarrow \mathcal{N}^*/(e, \alpha)$ (1-hyperedge-splitting), and
           $Q_r \leftarrow Q_r \cup \{(e, \alpha)\}$.
10:    else
11:        Take $e_1, e_2 \in \delta_{\mathcal{N}^*}(r)$ s.t. $e_1 \subseteq T^* + r$ and $e_2 \cap T^* = \emptyset$,
           and compute $\alpha := \alpha_r(e_1, e_2; \mathcal{N}^*)$.
12:        $\alpha_{\max} \leftarrow \min\{c(e_1), c(e_2)\}$.
13:        $\mathcal{N}^* \leftarrow \mathcal{N}^*/(e_1, e_2, \alpha)$ (2-hyperedge-splitting), and
           $Q_r \leftarrow Q_r \cup \{(e_1, e_2, \alpha)\}$.
14:    end if
15:    if $\alpha < \alpha_{\max}$ then
16:        For the $r$-tight cut $T$ in $\mathcal{N}^*$ found by hyperedge-
           splitting (Lemma 3-(ii)), let $T^* \leftarrow T$.
17:    end if
18: end while
19: $\mathcal{N}^* \leftarrow \mathcal{N}^*/(e, c(e)/2)$ (1-hyperedge-splitting), and
    $Q_r \leftarrow Q_r \cup \{(e, c(e)/2)\}$ for unique $e \in \delta_{\mathcal{N}^*}(r)$ if exists.
20: Return $\mathcal{N}_r := \mathcal{N}^* - r$ and $Q_r$.
```

The following lemma guarantees the correctness and efficiency of Algorithm 1. It is rather easy to see that the algorithm returns a desired hypernetwork when it halts, since it just performs hyperedge-splitting without destroying any r-tight cut. The running time bound is much more difficult to see, which is based on the facts that each hyperedge in $\delta_{\mathcal{N}}(r)$ is in one of the four irreversible states, and that each hyperedge-splitting changes the state of at least one hyperedge in $\delta_{\mathcal{N}}(r)$.

Lemma 5: Algorithm 1 correctly isolates $r \in V$ in a hypernetwork $\mathcal{N} = (\mathcal{H} = (V, \mathcal{E}), c)$, i.e., it returns a hypernetwork $\mathcal{N}_r = ((V - r, \mathcal{E}'), c')$ with the following conditions 1) and 2), after at most $3|\delta_{\mathcal{N}}(r)|$ hyperedge-splittings, and runs in $O\left(|\delta_{\mathcal{N}}(r)| \cdot (|V| \cdot \|\mathcal{E}\| + |V|^2 \log |V|)\right)$ time.

1) $\kappa_{\mathcal{N}_r}(U) \leq \kappa_{\mathcal{N}}(U)$ for every cut U in $\mathcal{N}_r$.
2) $\lambda(\mathcal{N}_r) = \lambda_r(\mathcal{N}) \geq \lambda(\mathcal{N})$.

In this algorithm, the variable T^* indicates an r-tight cut in $\mathcal{N}^*$ unless just after line 4 is performed, and it monotonically expands in line 16 [by Lemma 4-2), 3)] until line 4 is performed next. The monotonicity of T^* plays an important role on the irreversibility of the state of each hyperedge in $\delta_{\mathcal{N}}(r)$, which leads to the upper-bound on the number of performances of hyperedge-splittings in Lemma 5. The condition at line 6 is to guarantee this monotonicity of T^*. If there exists a 1-hyperedge-splitting that does not violate the monotonicity of T^*, then the algorithm performs such a 1-hyperedge-splitting (lines 7–9), and otherwise it performs

Algorithm 2 Hypergraph Small Cut Enumeration

```
Input: A hypernetwork $\mathcal{N} = (\mathcal{H} = (V, \mathcal{E}), c)$ and $k \geq 1$.
Output: The set $\mathcal{C}_k(\mathcal{N})$ of all $k$-small cuts in $\mathcal{N}$.
 1: Compute $\lambda(\mathcal{N})$, and set $\mathcal{N}_n \leftarrow \mathcal{N}$ and $\beta \leftarrow k\lambda(\mathcal{N})$.
 2: for $i = n, n-1, \dots, 2$ do
 3:     Take $v_i \in \arg\min_{v \in V(\mathcal{N}_i)} |\delta_{\mathcal{N}_i}(v)|$.
 4:     Isolate $v_i$ in $\mathcal{N}_i$ by Algorithm 1, and $\mathcal{N}_{i-1} \leftarrow \mathcal{N}_{v_i}$.
 5: end for
 6: $r \leftarrow v_1$ (reference node), and $\mathcal{C}_r^{\leq\beta}(\mathcal{N}_1) \leftarrow \emptyset$.
 7: for $j = 2, 3, \dots, n$ do
 8:     $\kappa_{\mathcal{N}_j}(v_j) \leftarrow 2\sum\{\alpha \mid (e, \alpha) \in Q_{v_j}$ or $(e_1, e_2, \alpha) \in Q_{v_j}\}$.
 9:     for each $U \in \mathcal{C}_r^{\leq\beta}(\mathcal{N}_{j-1})$ do
10:         $\kappa_{\mathcal{N}_j}(U) \leftarrow \kappa_{\mathcal{N}_{j-1}}(U) + f_j(U)$ and
            $\kappa_{\mathcal{N}_j}(U + v_j) \leftarrow \kappa_{\mathcal{N}_{j-1}}(U) + g_j(U)$
            where $f_j$ and $g_j$ are defined naturally from (4).
11:     end for
12:     $\mathcal{C}_{+v_j}[\mathcal{C}_r^{\leq\beta}(\mathcal{N}_{j-1})] \leftarrow \mathcal{C}_r^{\leq\beta}(\mathcal{N}_{j-1}) \cup \{U + v_j \mid U \in \mathcal{C}_r^{\leq\beta}(\mathcal{N}_{j-1})\} \cup \{\{v_j\}\}$.
13:     $\mathcal{C}_r^{\leq\beta}(\mathcal{N}_j) \leftarrow \{U \in \mathcal{C}_{+v_j}[\mathcal{C}_r^{\leq\beta}(\mathcal{N}_{j-1})] \mid \kappa_{\mathcal{N}_j}(U) \leq \beta\}$.
14: end for
15: Return $\mathcal{C}_k(\mathcal{N}) := \mathcal{C}_r^{\leq\beta}(\mathcal{N}_n)$.
```

such a 2-hyperedge-splitting (lines 11–13). It is not trivial that such a 2-hyperedge-splitting can be performed, but it can be done in fact unless $T^* \cap \Gamma_{\mathcal{N}}(r) = \emptyset$ (the condition in line 3 to initialize T^*).

Line 19 is performed only when $|\delta_{\mathcal{N}^*}(r)| = 1$. We describe this separately, since the condition in line 6 may not be satisfied in this situation but the 1-hyperedge-splitting $(e, c(e)/2)$ $(e \in \delta_{\mathcal{N}^*}(r))$ can be performed.

F. Enumerating All Small Cuts

Our algorithm to find all small cuts in a hypernetwork is shown as Algorithm 2. The basic idea is the same as the algorithm of Nagamochi *et al.* [9], and the overview is briefly described in Section V-B.

In lines 2–5, the algorithm constructs a sequence of hypernetworks $\mathcal{N}_{i-1}$ $(i = n, n-1, \dots, 2)$ by isolating (applying Algorithm 1 to) a node $v_i \in V(\mathcal{N}_i)$ with $|\delta_{\mathcal{N}_i}(v_i)|$ minimum and removing v_i. Let v_1 denote the node that remains in $\mathcal{N}_1$. After that, in lines 7–14, it enumerates all cuts U in $\mathcal{N}_j$ with $\kappa_{\mathcal{N}_j}(U) \leq \beta = k\lambda(\mathcal{N})$ for $j = 2, 3, \dots, n$. Note that, for the enumeration, it suffices to consider cuts U in $\mathcal{N}_j$ with $v_1 \notin U$ since U and $V(\mathcal{N}_j) \setminus U$ has the same capacity in $\mathcal{N}_j$. Hence, we maintain the set $\mathcal{C}_r^{\leq\beta}(\mathcal{N}_j)$ of all such cuts (U with $\kappa_{\mathcal{N}_j}(U) \leq \beta$ and $v_1 \notin U$) by using the information Q_{v_j} of hyperedge-splittings performed in line 4 (Algorithm 1).

VI. Conclusion

We have presented two exact and efficient solution methods for cyber security analysis problems of power networks using hypergraphs. One is to compute the security index of a specified arc or node, and the other is to find a sparsest attack in the whole network. We have also pointed out the significance of sparse elementary attacks, and presented an algorithm to enumerate sparse elementary attacks by enumerating small cuts in a hypergraph, which extends such an algorithm for an undirected graph. Some theoretical and experimental analyses of our generalized algorithm are left as future works.

Algorithm 3 Hypergraph Minimum Cut Algorithm

```
Input: A hypernetwork $\mathcal{N} = (\mathcal{H} = (V, \mathcal{E}), c)$.
Output: A cut $U$ with $\kappa_{\mathcal{N}}(U)$ minimum.
 1: $K \leftarrow +\infty$, $U \leftarrow \emptyset$, $\mathcal{N}' \leftarrow \mathcal{N}$ $(\mathcal{N}' = (\mathcal{H}' = (V', \mathcal{E}'), c'))$.

 2: while $|V'| \geq 2$ do
 3:     $W \leftarrow \emptyset$.
 4:     while $W \neq V'$ do
 5:         $W \leftarrow W + v$, where $v \in V' \setminus W$ is taken so that
            $\sum\{c'(e) \mid e \in \delta_{\mathcal{N}'}(\{v\}, W)\}$ is maximized.
 6:         if $|V' \setminus W| = 1$ then
 7:             $s \leftarrow v$.
 8:         end if
 9:         if $W = V'$ then
10:             $t \leftarrow v$.
11:         end if
12:     end while
13:     if $\kappa_{\mathcal{N}'}(\{t\}) < K$ then
14:         $U \leftarrow \{v \in V \mid v$ was merged into $t\}$, and
            $K \leftarrow \kappa_{\mathcal{N}'}(\{t\})$.
15:     end if
16:     Update $\mathcal{N}'$ by merging $s$ and $t$ into a single node.
17: end while
18: Return $U$.
```

Appendix

A. Hypergraph Minimum Cut Algorithm

Here we describe the algorithm of Klimmek and Wagner [4] for finding a minimum cut in a hypergraph in order to use for proofs below. Define $\delta_{\mathcal{N}}(U, W) := \{e \in \mathcal{E} \mid e \cap U \neq \emptyset \neq e \cap W$ and $c(e) > 0\}$ for a hypernetwork $\mathcal{N} = ((V, \mathcal{E}), c)$.

The idea is simple: to find a minimum s–t cut in the temporary hypernetwork $\mathcal{N}'$ for some pair of $s, t \in V'$, and to update $\mathcal{N}'$ by merging s and t into a single node, repeatedly. Since the merging does not change the capacity of any cut in $\mathcal{N}'$ that does not separate s and t, a minimum cut among the cuts found above is also a minimum cut in the original hypernetwork. The following lemma guarantees that the cut $\{t\}$ in $\mathcal{N}'$, whose capacity is checked in line 13, is indeed a minimum s–t cut after line 12 if we choose s and t along lines 3–11.

Lemma 6 (Klimmek and Wagner [4]): After line 12 in Algorithm 3, the cut $\{t\}$ in the temporary hypernetwork $\mathcal{N}'$ has the minimum capacity among all s–t cuts in $\mathcal{N}'$.

B. Proofs

Proof of Lemma 2: We use the algorithm of Klimmek and Wagner [4] for finding a minimum cut in a hypergraph shown as Algorithm 3 in the previous section. After line 12 in Algorithm 3, the cut $\{t\}$ is guaranteed to be a minimum s–t cut by Lemma 6. If we always take the designated node r as the first v in line 5 (note that the first v is arbitrary since $\delta_{\mathcal{N}'}(\{v\}, \emptyset) = \emptyset$ for every $v \in V'$), then r remains in $\mathcal{N}'$ until the final iteration. We show that, if we replace line 2 by "**while** $|V| > 2$ **do**" and line 3 by "$W \leftarrow \{r\}$," then the modified algorithm returns an r-tight cut in $\mathcal{N}$.

Since the cut $\{t\}$ is a minimum s–t cut after line 12, the modified algorithm returns a cut U in $\mathcal{N}$ with the minimum capacity such that U is an s–t cut after line 12 for some iteration step (note that some nodes in U was merged into single nodes in line 16). The hypernetwork $\mathcal{N}'$ has exactly two nodes including r when the modified algorithm halts. Hence, for any r-proper cut U in $\mathcal{N}$, we have $|U \cap \{s,t\}| = 1$ in some iteration step, and for any cut U in $\mathcal{N}'$ with $|U \cap \{s,t\}| = 1$ in some iteration step, one of U and $V' \setminus U$ (with the same capacity) is r-proper. Thus, we have proven that the modified algorithm returns an r-tight cut in $\mathcal{N}$, since each cut U in $\mathcal{N}'$ remains with the same capacity after merging s and t if $|U \cap \{s,t\}| \neq 1$. ■

Proof of Lemma 3: Apply Lemma 2 to $\mathcal{N}$ and $\tilde{\mathcal{N}} := \mathcal{N}/(e_1, \alpha_{\max})$ (or $\tilde{\mathcal{N}} := \mathcal{N}/(e_1, e_2, \alpha_{\max})$). If $\lambda_r(\mathcal{N}) = \lambda_r(\tilde{\mathcal{N}})$, then the desired value in (i) is obviously equal to $\alpha_{\max}$ and we have $c'(e_1) = 0$ (or $c'(e_1) = 0$ or $c'(e_2) = 0$) in $\mathcal{N}' = \tilde{\mathcal{N}}$. Otherwise, we obtain an r-tight cut $\tilde{T}$ in $\tilde{\mathcal{N}}$ that may not be r-tight in $\mathcal{N}$.

By Observation 3, the capacity of each cut decreases almost uniformly after a hyperedge-splitting. In the case of a 1-hyperedge-splitting of weight α, since the difference is always 2α if decreases, it is easily seen that

$$\alpha(e_1; \mathcal{N}) = \alpha_{\max} - \frac{\lambda_r(\mathcal{N}) - \lambda_r\left(\tilde{\mathcal{N}}\right)}{2}$$

holds and $\tilde{T}$ is also r-tight in $\mathcal{N}'$. This $\tilde{T}$ must include $e' = e_1 - r$ by (4).

In the case of a 2-hyperedge-splitting of weight α, though there are two possible differences α and 2α, the same idea works well. Let

$$\tilde{\alpha} := \begin{cases} \alpha_{\max} - \dfrac{\lambda_r(\mathcal{N}) - \lambda_r\left(\tilde{\mathcal{N}}\right)}{2} & \left(e_1 \triangle e_2 \subseteq \tilde{T}\right) \\ \alpha_{\max} - \left(\lambda_r(\mathcal{N}) - \lambda_r\left(\tilde{\mathcal{N}}\right)\right) & \text{(otherwise)} \end{cases} \tag{5}$$

and apply Lemma 2 to $\hat{\mathcal{N}} := \mathcal{N}/(e_1, e_2, \tilde{\alpha})$. If $\lambda_r(\mathcal{N}) = \lambda_r(\hat{\mathcal{N}})$, then we have $\alpha(e_1, e_2; \mathcal{N}) = \tilde{\alpha}$ and $\tilde{T}$ is also r-tight in $\mathcal{N}' = \hat{\mathcal{N}}$. This $\tilde{T}$ must satisfy $e_1 \cap \tilde{T} \neq \emptyset \neq e_2 \cap \tilde{T}$ by (4) (note that $e' = e_1 \triangle e_2 \subseteq \tilde{T}$ implies this condition).

Otherwise, we obtain an r-tight cut $\hat{T}$ in $\hat{\mathcal{N}}$ that is not r-tight both in $\mathcal{N}$ and in $\tilde{\mathcal{N}}$. In this case, we have $\kappa_{\mathcal{N}}(\tilde{T}) - \kappa_{\tilde{\mathcal{N}}}(\tilde{T}) = 2\tilde{\alpha}$, $\kappa_{\mathcal{N}}(\hat{T}) - \kappa_{\hat{\mathcal{N}}}(\hat{T}) = \tilde{\alpha}$, and $\kappa_{\mathcal{N}}(U) - \kappa_{\hat{\mathcal{N}}}(U) = \tilde{\alpha}$ for any r-proper cut U with $\kappa_{\hat{\mathcal{N}}}(U) < \lambda_r(\mathcal{N})$, since the possible differences of the cut capacities are only 0, $\tilde{\alpha}$, and $2\tilde{\alpha}$. Hence, we have $\alpha(e_1, e_2; \mathcal{N}) = \alpha_{\max} - \left(\lambda_r(\mathcal{N}) - \lambda_r(\hat{\mathcal{N}})\right)$, and $\hat{T}$ is also r-tight in $\mathcal{N}' = \mathcal{N}/(e_1, e_2, \alpha(e_1, e_2; \mathcal{N}))$ and satisfies $e_1 \cap \hat{T} \neq \emptyset \neq e_2 \cap \hat{T}$. ■

Proof of Lemma 4:

1) Suppose to the contrary that some r-tight cut $T \subsetneq V - r$ contains all neighbors of r in $\Gamma_{\mathcal{N}}(r)$, i.e., there is no hyperedge with positive capacity across $\{r\}$ and $V \setminus T$. Then, for the r-proper cut $R := V \setminus (T + r)$, we have $\kappa_{\mathcal{N}}(T) = \kappa_{\mathcal{N}}(R) + \kappa_{\mathcal{N}}(\{r\}) > \lambda_r(\mathcal{N}) = \kappa_{\mathcal{N}}(T)$, a contradiction.

2) Suppose to the contrary that some hyperedge $e \in \delta_{\mathcal{N}}(r)$ and two r-tight cuts T and T' with $e \cap T' \neq \emptyset$, $e \setminus T' \neq \{r\}$ and $e \subseteq T + r$ violate the property, i.e., $T' \setminus T \neq \emptyset$ or $T = T'$. Then, T and T' are crossing (all of $T \cap T'$, $T \setminus T'$, $T' \setminus T$, and $V \setminus (T \cup T')$ are nonempty), and hence we have

$$\begin{aligned}\kappa_{\mathcal{N}}(T) + \kappa_{\mathcal{N}}\left(T'\right) &\geq \kappa_{\mathcal{N}}\left(T \setminus T'\right) + \kappa_{\mathcal{N}}\left(T' \setminus T\right) + c(e) \\ &> 2\lambda_r(\mathcal{N}) = \kappa_{\mathcal{N}}(T) + \kappa_{\mathcal{N}}\left(T'\right)\end{aligned}$$

a contradiction, where the first inequality is easily checked by enumerating hyperedges that contribute to the cut capacities.

3) Suppose to the contrary that some hyperedges $e_1, e_2 \in \delta_{\mathcal{N}}(r)$ and two r-tight cuts T and T' with $e_1 \cap e_2 = \{r\}$, $e_1 \subseteq T' + r$, $e_2 \cap T' = \emptyset$, $e_1 \cap T \neq \emptyset$, and $e_2 \cap T \neq \emptyset$ violate the property, i.e., $T' \setminus T \neq \emptyset$ or $T = T'$. Then, T and T' are crossing, and hence by the similar observation as the proof of 2), we have

$$\begin{aligned}\kappa_{\mathcal{N}}(T) + \kappa_{\mathcal{N}}\left(T'\right) &\geq \kappa_{\mathcal{N}}\left(T \setminus T'\right) + \kappa_{\mathcal{N}}\left(T' \setminus T\right) + c(e_1) \\ &> 2\lambda_r(\mathcal{N}) = \kappa_{\mathcal{N}}(T) + \kappa_{\mathcal{N}}\left(T'\right)\end{aligned}$$

a contradiction. ■

Proof of Lemma 5: First of all, we show that the output $\mathcal{N}_r$ satisfies the two conditions when Algorithm 1 halts. The condition 1) holds by Observation 3, since the algorithm just performs hyperedge-splittings repeatedly. Each hyperedge-splitting is performed so that it does not destroy any r-tight cut in the temporary hypernetwork $\mathcal{N}^*$ (i.e., the capacity of each r-tight cut does not change and all r-tight cuts remain r-tight) in the while loop (lines 2–18), and hence it suffices to check line 19. Suppose that the r-connectivity $\lambda_r(\mathcal{N}^*)$ decreases by 1-hyperedge-splitting e of weight $c(e)/2$ under the condition that $\delta_{\mathcal{N}^*}(r) = \{e\}$. Then, by Observation 3, we must have $e \subseteq T + r$ for some r-tight cut T in $\mathcal{N}' := \mathcal{N}^*/(e, \alpha_r(e; c(e)/2))$ with $\kappa_{\mathcal{N}'}(T) < \lambda_r(\mathcal{N}^*)$. Since $\alpha_r(e; \mathcal{N}^*) < c(e)/2$, the hyperedge $e \in \delta_{\mathcal{N}^*}(r)$ remains in $\mathcal{N}^*/(e, \alpha_r(e; \mathcal{N}^*))$, which contradicts Lemma 4-1).

Next, we confirm that line 11 can be performed if the condition in line 6 is not satisfied. Since at least one hyperedge $e_1 \in \delta_{\mathcal{N}^*}(r)$ is across $\{r\}$ and T^* just before line 6, it suffices to show that there exists a hyperedge $e_2 \in \delta_{\mathcal{N}^*}(r)$ with $e_2 \cap T^* = \emptyset$. If line 4 has been performed in the current iteration, then it immediately follows from $|\delta_{\mathcal{N}^*}(r)| > 1$ and the inexistency of e satisfying the condition in line 6. During the algorithm, each hyperedge $e \in \delta_{\mathcal{H}}(r)$ is in one of the four possible states: (S1) $e \in \delta_{\mathcal{N}^*}(r)$ and $e \cap T^* = \emptyset$, (S2) $e \in \delta_{\mathcal{N}^*}(r)$, $e \cap T^* \neq \emptyset$ and $e \setminus T^* \neq \{r\}$, (S3) $e \in \delta_{\mathcal{N}^*}(r)$ and $e \subseteq T^* + r$, and (S4) $e \notin \delta_{\mathcal{N}^*}(r)$, i.e., the capacity of e in $\mathcal{N}^*$ is 0. By Lemma 4-1), there exists $e \in \delta_{\mathcal{H}}(r)$ in the state (S1) or (S2). Since the condition in line 6 is equivalent to the existence of a hyperedge in the state (S2), e is in (S1). Hence, we can choose such e as e_2, and thus line 11 can be performed.

We start to prove the bound on the number of hyperedge-splittings performed in Algorithm 1, using the states (S1)–(S4) defined above. In the rest of this proof, we show the following two statements: 1) these states are irreversible, i.e., the state index is monotone nondecreasing for each hyperedge throughout the algorithm and 2) after each hyperedge-splitting in line 9 or 13, for at least one hyperedge, the state index increases. Note that, since the capacity of every edge $e \in \delta_{\mathcal{H}}(r)$ monotonically decreases during the algorithm, we need not to consider any transition from the state (S4).

To see these statements (a) and (b), we first show that T^* monotonically increases unless line 4 is executed. This follows from Lemma 4-2), 3). If $|T^*| > 1$, then T^* was updated in line 16 in the previous iteration, and hence T^* is r-tight. Then, by applying Lemma 4-2), 3) to T and T^* just before line 16 in the current iteration, we confirm $T^* \subsetneq T$. Moreover, by Lemma 4-2), 3), each expansion of T^* involves increasing the state index of one of the selected hyperedges for the hyperedge-splitting in the current iteration as follows. In the case of 1-hyperedge-splitting (e, α) in line 9, e is in the state (S2) before line 9 by the condition in line 6, in (S4) after line 9 if $\alpha = \alpha_{\max}$. and otherwise in (S3) after line 16 by Lemma 4-2). In the case of 2-hyperedge-splitting (e_1, e_2, α) in line 13, e_2 is in the state (S1) before line 13 by the condition in line 11, in (S4) after line 13 if $\alpha = c(e_2)$, and otherwise in (S2) or (S3) after line 16 by Lemma 4-3).

Finally, we consider update of T^* in line 4. If the condition in line 3 is satisfied, i.e., $T^* \cap \delta_{\mathcal{N}^*}(r) = \emptyset$, then every $e \in \delta_{\mathcal{H}}(r)$ is in the state (S1) or (S4). Hence, each update of T^* in line 4 changes the state of some $e \in \delta_{\mathcal{H}}(r)$ in the state (S1) into (S2) and does not change the state of any $e \in \delta_{\mathcal{H}}(r)$ in the state (S4). ■

Proof of Theorem 3: The proof is almost the same as [9, Section 6]. One of the main differences appears in line 10 of Algorithm 2, in which it computes the cut capacities $\kappa_{\mathcal{N}_j}(U)$ and $\kappa_{\mathcal{N}_j}(U + v_j)$ for a small cut U in $\mathcal{N}_{j-1}$ with $\kappa_{\mathcal{N}_{j-1}}(U) \leq \beta$. It is easily seen from (4) that f_j and g_j defined as follows work well:

$$\begin{aligned} f_j(U) := {} & 2\sum\{\alpha \mid (e,\alpha) \in Q_{v_j} \text{ with } e \subseteq U + v_j\} \\ & + 2\sum\{\alpha \mid (e_1, e_2, \alpha) \in Q_{v_j} \text{ with } e_1 \triangle e_2 \subseteq U\} \\ & + \sum\left\{\alpha \;\middle|\; \begin{array}{l} (e_1, e_2, \alpha) \in Q_{v_j} \text{ with } (e_1 \triangle e_2) \setminus U \neq \emptyset \\ \text{and } e_1 \cap U \neq \emptyset \neq e_2 \cap U \end{array}\right\} \\ g_j(U) := {} & 2\sum\{\alpha \mid (e,\alpha) \in Q_{v_j} \text{ with } e \cap U = \emptyset\} \\ & + 2\sum\{\alpha \mid (e_1, e_2, \alpha) \in Q_{v_j} \text{ with } (e_1 \triangle e_2) \cap U = \emptyset\} \\ & + \sum\left\{\alpha \;\middle|\; \begin{array}{l} (e_1, e_2, \alpha) \in Q_{v_j} \text{ with } (e_1 \triangle e_2) \cap U \neq \emptyset \\ \text{and } e_1 \setminus U \neq \{v_j\} \neq e_2 \setminus U \end{array}\right\}. \end{aligned}$$

The other main difference appears in the part of bounding the computational time of lines 2–5. Note that, for each $i = n, n-1, \ldots, 2$, the algorithm chooses $v_i \in V_i$ in line 3 so that $|\delta_{\mathcal{N}_i}(v_i)|$ is minimized, which implies $|\delta_{\mathcal{N}_i}(v_i)| \leq \|\mathcal{E}\|/i$. This fact leads to the following bound on the computational time of lines 2–5:

$$O\left(\sum_{i=1}^{n} \frac{\|\mathcal{E}\|}{i} i \left(\|\mathcal{E}\| + |V| \log |V|\right)\right) = O\left(\|\mathcal{E}\| \cdot |V| \cdot \left(\|\mathcal{E}\| + |V| \log |V|\right)\right).$$

For each $j = 2, 3, \ldots, n$, the construction of the set $\mathcal{C}_r^{\leq\beta}(\mathcal{N}_j)$ of all cuts U in $\mathcal{N}_j$ with $\kappa_{\mathcal{N}_j}(U) \leq \beta$ and $v_1 \notin U$ in lines 8–13 can be done in $O(|\mathcal{C}_r^{\leq\beta}(\mathcal{N}_{j-1})| \cdot |Q_{v_j}|)$ time. Recall that $h_k(\mathcal{N}) = \max_i h_k(\mathcal{N}, i) = \max_i |\mathcal{C}_r^{\leq\beta}(\mathcal{N}_i)|$. We have $|Q_{v_i}| = O(|\delta_{\mathcal{N}_i}(v_i)|)$ by Lemma 5, and hence the computational time of lines 7–14 is bounded by

$$\sum_{i=1}^{n} O\left(|\mathcal{C}_r^{\leq\beta}(\mathcal{N}_i)| \cdot |Q_{v_i}|\right) = \sum_{i=1}^{n} O\left(h_k(\mathcal{N}) \frac{\|\mathcal{E}\|}{i}\right) = O(h_k(\mathcal{N}) \cdot \|\mathcal{E}\| \log |V|).$$

Considering the above bound together with the computational time bound of lines 2–5 shown in the previous section, we finally obtain the total computational time bound shown in Theorem 3. ■

References

[1] A. Abur, *Power System State Estimation: Theory and Implementation*. Boca Raton, FL, USA: CRC Press, 2004.

[2] J. M. Hendrickx, K. M. Johansson, R. M. Jungers, H. Sandberg, and K. C. Sou, "Efficient computations of a security index for false data attacks in power networks," *IEEE Trans. Auto. Cont.*, vol. 59, no. 12, pp. 3194–3208, Aug. 2014

[3] D. R. Karger, "Random sampling in cut, flow, and network design problems," *Math. Oper. Res.*, vol. 24, no. 2, pp. 383–413, 1999.

[4] R. Klimmek and F. Wagner, "A simple hypergraph min cut algorithm," Internal Rep. B 96-02, Bericht FU Berlin Fachbereich Mathematik und Informatik, Freie Universität Berlin, 1996.

[5] J.-M. Lin and H.-Y. Pan, "A static state estimation approach including bad data detection and identification in power systems," in *Proc. IEEE Power Eng. Soc. Gen. Meeting*, Tampa, FL, USA, Jun. 2007, pp. 1–7.

[6] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," in *Proc. 16th ACM Conf. Comput. Commun. Security*, Chicago, IL, USA, Nov. 2009, pp. 21–32.

[7] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3397–3415, Dec. 1993.

[8] H. Nagamochi and T. Ibaraki, "Computing edge-connectivity in multigraphs and capacitated graphs," *SIAM J. Discrete Math.*, vol. 5, no. 1, pp. 54–66, Feb. 1992.

[9] H. Nagamochi, K. Nishimura, and T. Ibaraki, "Computing all small cuts in an undirected network," *SIAM J. Discrete Math.*, vol. 10, no. 3, pp. 469–481, Aug. 1997.

[10] M. Nagata *et al.*, "Phase-model analysis of supply stability in power grid of Eastern Japan," in *Proc. Int. Symp. Nonlinear Theory Appl.*, 2013, pp. 69–72.

[11] J. B. Orlin, "Max flows in $O(nm)$ time, or better," in *Proc. 45th Annu. ACM Symp. Theory Comput.*, 2013, pp. 765–774.

[12] J. Pistorius and M. Minoux, "An improved direct labeling method for the max-flow min-cut computation in large hypergraphs and applications," *Int. Trans. Oper. Res.*, vol. 10, no. 1, pp. 1–11, 2003.

[13] H. Sandberg, A. Teixeira, and K. H. Johansson, "On security indices for state estimators in power networks," in *Proc. 1st Workshop Secure Control Syst. (CPSWEEK)*, Stockholm, Sweden, 2010. [Online]. Available: https://www.truststc.org/conferences/10/CPSWeek/program.htm

[14] K. C. Sou, H. Sandberg, and K. H. Johansson, "Electric power network security analysis via minimum cut relaxation," in *Proc. 50th IEEE Conf. Decis. Control Eur. Control Conf. (CDC-ECC)*, Orlando, FL, USA, Dec. 2011, pp. 4054–4059.

[15] K. C. Sou, H. Sandberg, and K. H. Johansson, "On the exact solution to a smart grid cyber-security analysis problem," *IEEE Trans. Smart Grid*, vol. 4, no. 2, pp. 856–865, Jun. 2013.

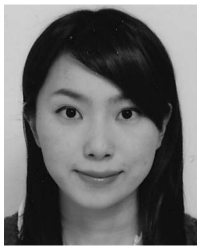
[16] H. H. Yang and D. F. Wong, "Efficient network flow based min-cut balanced partitioning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 15, no. 12, pp. 1533–1540, Dec. 1996.

[17] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[18] S. Zonouz *et al.*, "SCPST: Security-oriented cyber-physical state estimation for power grid critical infrastructures," *IEEE Trans. Smart Grid*, vol. 3, no. 4, pp. 1790–1799, Dec. 2012.

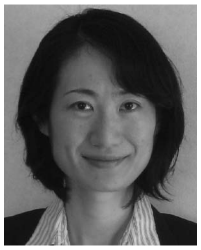
Yutaro Yamaguchi received the M.Sc. degree in mathematical sciences from Kyoto University, Kyoto, Japan, in 2013. He is currently pursuing the Ph.D. degree from the Department of Mathematical Informatics, University of Tokyo, Tokyo, Japan.

His current research interests include combinatorial optimization and graph theory.



Anna Ogawa received the B.E. degree in administration engineering from Keio University, Yokohama, Japan, in 2013, where she is currently pursuing the Master's degree.

Her current research interests include machine learning and its application to renewable energy systems.



Akiko Takeda received the B.E. and M.E. degrees in administration engineering from Keio University, Yokohama, Japan, in 1996 and 1998, respectively, and the Dr.Sc. degree in information science from the Tokyo Institute of Technology, Tokyo, Japan, in 2001.

She is currently an Associate Professor with the Department of Mathematical Informatics, University of Tokyo, Tokyo. Her current research interests include solution methods for decision making problems under uncertainty and nonconvex optimization problems, which appear in financial engineering, machine learning, and energy systems.

Satoru Iwata received the B.E. and M.E. degrees in mathematical engineering from the University of Tokyo, Tokyo, Japan, in 1991 and 1993, respectively, and the Dr.Sc. degree in mathematical sciences from Kyoto University, Kyoto, Japan, in 1996.

He is currently a Professor with the Department of Mathematical Informatics, University of Tokyo. His current research interests include combinatorial optimization, matroid theory, and their applications in systems analysis and control.

Prof. Iwata was the recipient of the Delbert Ray Fulkerson Prize in 2003 for his joint work on submodular function minimization.