MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

Cyber Security Analysis of Power Networks by Hypergraph Cut Algorithms

Yutaro YAMAGUCHI, Anna OGAWA,
Akiko TAKEDA and Satoru IWATA

METR 2014–12 May 2014

DEPARTMENT OF MATHEMATICAL INFORMATICS
GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY
THE UNIVERSITY OF TOKYO
BUNKYO-KU, TOKYO 113-8656, JAPAN

Cyber Security Analysis of Power Networks by Hypergraph Cut Algorithms[⋆]

Yutaro Yamaguchi[1⋆⋆], Anna Ogawa[2], Akiko Takeda[1], and Satoru Iwata[1]

[1] Department of Mathematical Informatics, University of Tokyo, Tokyo, Japan
{yutaro_yamaguchi, takeda, iwata}@mist.i.u-tokyo.ac.jp
[2] Department of Administration Engineering, Keio University, Yokohama, Japan
anna_ogawa@opt.mist.i.u-tokyo.ac.jp

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.

1 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 malicious measurement values injected by attackers 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 [4]. However, Liu et al. [5] 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 existing techniques for bad measurement detection.

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 detected. Sparse attacks reveal vulnerable sets of meters to be attacked in the system. Liu et al. [5] showed a cardinality minimization formulation to find a sparsest attack and solved it approximately by matching pursuit method [6]. Sandberg et al. [11] also 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. [11] proposed a simple heuristics to find a suboptimal solution for the problem, Sou et al. [12, 13] and Hendrickx et al. [1] provided efficient solution methods as follows. It was shown in [12] that the problem reduces to a minimum cost node-bipartitioning 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

[⋆] This research is supported by JST CREST.
[⋆⋆] Supported by JSPS Fellowship for Young Scientists.

minimization problem whose constraint matrix is totally unimodular [13]. While ℓ_1 relaxation can indeed solve the cardinality minimization problem, the assumption restricts the applicability of the model. As an improved result of [12], [1] 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 a set of nodes and a set of 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 s–t cut in a directed graph, which is almost the same as that suggested in [1]. 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 [1] for 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. [8]. This algorithm can be used to analyze possible sparse attacks in power network systems.

The rest of this paper is organized as follows. Section 2 is devoted to problem formulations and fundamental properties of the sparsest attacks and security index. In Section 3, 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 4. Finally, in Section 5, we present a new algorithm for enumerating small cuts in a hypergraph.

2 Cyber Security Analysis Problems

2.1 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, e)-entry is 1, (v, e)-entry is -1, and (w, e)-entries are 0 for all $w \in V \setminus \{u, v\}$.

Each node has a hidden *state* (e.g., voltage), and each node or arc has a *measurement point*. The operator of the system gets a measurement value (e.g., current) in each measurement point, which leads to some information of the states.

Definition 1 (Measurement matrix). Let D be an $m \times m$ positive diagonal matrix (each of whose diagonal entry represents the character of each arc, e.g., the inverse of its reactance). The *measurement matrix* H is an $(m + n) \times n$ matrix defined by

$$H := \begin{pmatrix} DB_G^\top \\ B_G DB_G^\top \end{pmatrix}. \tag{1}$$

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 $\varDelta z \in \mathbb{R}^{A \cup V}$ is called *undetectable* if there exists $\varDelta\theta \in \mathbb{R}^V$ such that $\varDelta z = H\varDelta\theta$. It is shown in [5] that an undetectable additive measurement value $\varDelta z$ 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* [5].

The support $\mathrm{supp}(\varDelta z) := \{k \mid k \in A \cup V,\ (\varDelta z)_k \neq 0\}$ of an undetectable attack $\varDelta z$ 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, i.e.,

$$\min_{\varDelta\theta \in \mathbb{R}^V} \{\|H\varDelta\theta\|_0 \mid H\varDelta\theta \neq \mathbf{0}\}. \tag{2}$$

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 $\varDelta z$, i.e., $k \in \mathrm{supp}(\varDelta z)$. The minimum size of an attackable set that contains k is called the *security index* [12] 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

$$\min_{\varDelta\theta \in \mathbb{R}^V} \{\|H\varDelta\theta\|_0 \mid H_k\varDelta\theta \neq 0\}, \tag{3}$$

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 $\varGamma_G(v)$ 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., $\varGamma_G(v) := \{u \mid \{u, v\} \in E\}$ 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 $\varDelta\theta \in \mathbb{R}^V$, $X := \mathrm{supp}(H\varDelta\theta)$ satisfies the following properties.*

- *For each arc $a = uv \in A$, we have $a \in X$ if and only if $(\varDelta\theta)_u \neq (\varDelta\theta)_v$.*
- *For each node $v \in V$, we have $v \in X$ if and only if $\sum_a \{(\varDelta\theta)_v D_a \mid e_a \in \delta_G(v)\} \neq \sum_{u \in \varGamma_G(v)} \sum_a \{(\varDelta\theta)_u D_a \mid u \in e_a \in \delta_G(v)\}$.*

2.2 Elementary Attacks

Sou et al. [12] 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 3.2. 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 $\varDelta\theta \in \mathbb{R}^V$ with $H\varDelta\theta \neq \mathbf{0}$ and any $\alpha \in \mathbb{R}$ with $\min_{v \in V}(\varDelta\theta)_v < \alpha < \max_{v \in V}(\varDelta\theta)_v$, $U := \{v \in V \mid (\varDelta\theta)_v > \alpha\}$ satisfies $\|H\chi_U\|_0 \leq \|H\varDelta\theta\|_0$.*

Proof. By Observation 1, if $(\varDelta\theta)_u = (\varDelta\theta)_v$ for any $u, v \in V$, then we have $H\varDelta\theta = \mathbf{0}$. We then assume that there exist $u, v \in V$ such that $(\varDelta\theta)_u \neq (\varDelta\theta)_v$, and hence we can always take α satisfying the assumption.

Let $H_A := DB_G^\top$ and $H_V := B_G D B_G^\top$ for submatrices in (1). To see $\|H\chi_U\|_0 \leq \|H\varDelta\theta\|_0$, we prove two statements: (i) $\mathrm{supp}(H_A\chi_U) \subseteq \mathrm{supp}(H_A\varDelta\theta)$, and (ii) there exists an injective map ρ on $\mathrm{supp}(H_V\chi_U) \setminus \mathrm{supp}(H_V\varDelta\theta)$ to $\mathrm{supp}(H_A\varDelta\theta) \setminus \mathrm{supp}(H_A\chi_U)$.

(i) By Observation 1, $a \notin \mathrm{supp}(H_A\varDelta\theta)$ implies $(\varDelta\theta)_u = (\varDelta\theta)_v$ for any $a = uv \in A$. Hence we have $U \cap \{u, v\} = \emptyset$ or $\{u, v\}$, which leads to $(\chi_U)_u = (\chi_U)_v$. Thus we have $a \notin \mathrm{supp}(H_A\chi_U)$.

(ii) Take a vertex $v \in \mathrm{supp}(H_V\chi_U) \setminus \mathrm{supp}(H_V\varDelta\theta)$. Take $u \in \arg\max_{x\in\varGamma_G(v)}(\varDelta\theta)_x$ and $w \in \arg\min_{x\in\varGamma_G(v)}(\varDelta\theta)_x$, and let $\alpha := (\varDelta\theta)_u$, $\beta := (\varDelta\theta)_v$, and $\gamma := (\varDelta\theta)_w$. Then, by Observation 1, we have $\alpha > \beta > \gamma$, and we must have $(\chi_U)_u = 1$ and $(\chi_U)_w = 0$ since $v \in \mathrm{supp}(H_V\chi_U)$.

Let a be an arc between v and u if $(\chi_U)_v = 1$, and between v and w otherwise. We then have $a \in \mathrm{supp}(H_A\varDelta\theta) \setminus \mathrm{supp}(H_A\chi_U)$. Let us define $\rho(v) := a$. After this procedure for every $v \in \mathrm{supp}(H_V\chi_U) \setminus \mathrm{supp}(H_V\varDelta\theta)$, we get a map ρ on $\mathrm{supp}(H_V\chi_U) \setminus \mathrm{supp}(H_V\varDelta\theta)$ to $\mathrm{supp}(H_A\varDelta\theta) \setminus \mathrm{supp}(H_A\chi_U)$. We finally prove that ρ is injective.

Suppose to the contrary that ρ is not injective. Then there exists an arc $a = uv \in A$ with $\rho(u) = \rho(v) = a$. By the definition of ρ, we may assume $(\chi_U)_u = (\chi_U)_v = 1$ and $(\varDelta\theta)_u < (\varDelta\theta)_v$ without loss of generality. Then, in the above procedure with respect to v, there exists $u' \in \arg\max_{x\in\varGamma_G(v)}(\varDelta\theta)_x$ with $\rho(v) = u'v \in A$ or $\rho(v) = vu' \in A$. On the other hand, since $(\varDelta\theta)_u < (\varDelta\theta)_v$, we have $u \notin \arg\max_{x\in\varGamma_G(v)}(\varDelta\theta)_x$, a contradiction. □

Lemma 1 suggests that each minimum attackable set has the corresponding bipartition 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-bipartiton problems since these problems are interested only in the sparsest attacks.

Let us call an undetectable attack $\varDelta z$ an *elementary attack* if $\varDelta z = H\chi_U$ for some proper nonempty subset U of V, and such U a *false set* (of $\varDelta z$). The following observation is easily seen from Observation 1.

Observation 2 *For a proper nonempty subset U of V, $X := \mathrm{supp}\,(H\chi_U)$ satisfies the following properties.*

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

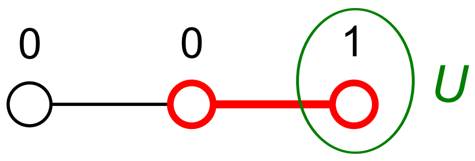

Fig. 1. An example of elementary attacks. The number above each node is the entry of the characteristic vector χ_U. The nodes and arcs in X (i.e., to be attacked) are colored red.

2.3 Enumerating All Sparse Elementary Attacks

In considering sparse undetectable attacks that are not necessarily the sparsest, sparse elementary attacks play an important role. Since Lemma 1 does not depends on the threshold α, for any undetectable attack $\varDelta z = H\varDelta\theta$ and any $\min_{v\in V}(\varDelta\theta)_v < \alpha < \max_{v\in V}(\varDelta\theta)_v$, $U := \{v \in V \mid (\varDelta\theta)_v > \alpha\}$ is a false set of a no denser elementary attack than $\varDelta z$, i.e., $\|H\chi_U\|_0 \leq \|\varDelta z\|_0$. By changing α in the possible range, we get a nested family of false sets of no denser elementary attacks, i.e., $\emptyset \neq U_1 \subset U_2 \subset \cdots \subset U_r \subset V$ such that $\left\|H\chi_{U_j}\right\|_0 \leq \|\Delta z\|_0$ for each U_j with $1 \leq j \leq r$. In other words, only undetectable attacks that can be written as a nonnegative combination of sparse elementary attacks with nested false sets can be sparse.

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 $\|\Delta z\|_0 \leq \beta$ for a given measurement matrix and a positive integer β. As seen in the next section, this problem can be interpreted as to enumerate all small cuts in a given hypergraph, for which a combinatorial algorithm will be given in Section 5.

3 Reduction to Minimum Cuts in Hypergraphs

3.1 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 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 $c : \mathcal{E} \to \mathbb{R}_{\geq 0}$ 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 $\mathcal{N}$), and define $\delta_{\mathcal{H}}(U) := \{e \in \mathcal{E} \mid e \cap X \neq \emptyset,\ e \setminus X \neq \emptyset\}$. The *capacity* of a cut U is defined by

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

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.

Based on the concept of MA-ordering, introduced by Nagamochi and Ibaraki [7] for graphs, Klimmek and Wagner [3] 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 [14], 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 [10]. 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.

3.2 Reduction from Cyber Security Analysis Problems

Recall that $G' = (V, E)$ denotes the underlying graph of the input graph G. By Sections 2.2 and 2.3, we should consider

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

over all proper nonempty subset U of V, i.e., over all cuts U in G'. 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 $\mathcal{E} := E \cup \{\mathit{\Gamma}_G(v) \cup \{v\} \mid v \in V\}$, and $\mathcal{N} := ((V, \mathcal{E}), \mathbf{1})$. Then, for each cut U in $\mathcal{N}$, we have $f(U) = \kappa_{\mathcal{N}}(U)$

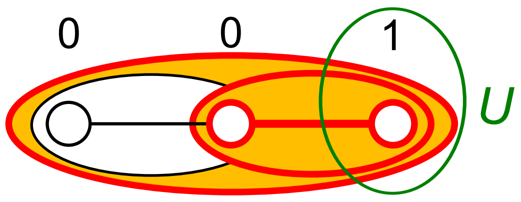

Fig. 2. An example of computing the size of $X := \mathrm{supp}(H\chi_U)$ as the capacity $\kappa_{\mathcal{N}}(U)$ in the auxiliary hypernetwork $\mathcal{N}$. The number above each node is the entry of the characteristic vector χ_U. The nodes and arcs in X (i.e., to be attacked), and the hyperedges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

since, for each $e_v = \Gamma_G(v) \cup \{v\}$, we have $\emptyset \neq e_v \cap U \neq e_v$ if and only if $|U \cap e| = 1$ for some $e \in \delta_G(v)$.

Recall Lemma 1 and Observation 2. Then the sparsest attack problem immediately reduces to the hypergraph minimum cut problem. 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.

By the definition of $\mathcal{E}$ in this reduction, $\|\mathcal{E}\| = 2|A| + \sum_{v \in V}(|\delta_G(v)| + 1) = 4|A| + |V| = O(m)$. Using the algorithm of Klimmek and Wagner [3] for finding a minimum cut in a hypergraph, we obtain the following result.

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.*

We have $|V| = n$ and $\|\mathcal{E}\| = O(m)$. Moreover, the maximum value of an s–t flow for any distinct $s, t \in V$ is $O(n)$ since the underlying graph is practically almost simple (hence the maximum value of an s–t flow in G' is $O(n)$) and the number of added hyperedges is n. Thus the running 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 $v \in V$.*

Remark. Our reduction works well in a slightly more general setting. Suppose that some arcs and some nodes do not have their measurement points. Even in such a situation, if each arc incident to any node with a measurement point has a measurement point, then the same statement as Lemma 1 holds, and hence our reduction works well.

In addition, as shown in [1], one can introduce a cost function $c : A \cup V \to \mathbb{R}_+$ that 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 to be minimized over $\Delta\theta \in \mathbb{R}^V$ is replaced by $\sum_{k \in \mathrm{supp}(H\Delta\theta)} c(k)$. If $c(a) \leq c(v)$ holds for every pair of $v \in V$ and $e_a \in \delta_G(v)$, our reduction is extendable to this setting by regarding c as the capacity function (for each $a \in A$, $c(a)$ is the capacity of the corresponding edge $e_a \in E \subset \mathcal{E}$, and for each $v \in V$, $c(v)$ the corresponding additional hyperedge $\Gamma_{G'}(v) \cup \{v\} \in \mathcal{E}$). In this situation, Theorem 1 holds as it does. Theorem 2 also carries over this setting with the aid of an advanced maximum flow algorithm [9].

4 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.

Table 1. Accuracy of the two relaxation methods. The first row “error” 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
number of nodes		9	14	30	118	300	2383	2746	3012	3375
number of arcs		9	20	41	186	411	2896	3514	3572	4161
`min.s-t cut relax`	error	0.00	0.150	0.220	0.210	0.122	0.124	0.154	0.117	0.112
	approx.	1.000	1.025	1.040	1.032	1.019	1.022	1.027	1.021	1.020
`L1-relax (LP)`	error	0.667	0.600	0.683	0.726	0.599	0.644	0.738	0.676	0.655
	approx.	1.368	1.488	1.476	1.813	1.625	1.442	1.364	1.340	1.345

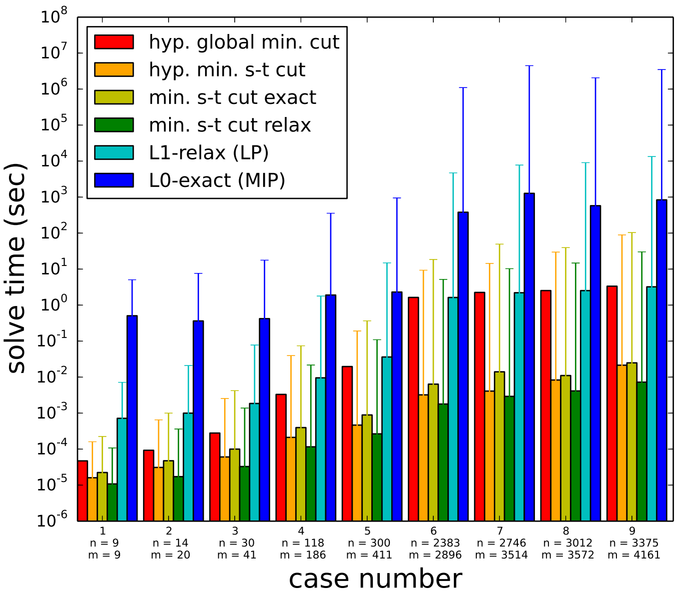

Fig. 3. Running time of the six methods applied to nine benchmarks. The segment on each bar corresponds to the number of arcs, so that its top represents the computationed time required to find a sparsest attack.

- `min.s-t cut exact` [1]: Find a minimum s–t cut in an auxiliary directed graph, which leads to the security index.
- `min.s-t cut relax` [12]: Find a minimum s–t cut in the original graph to give an upper bound on the security index.
- `L1-relax (LP)` [13]: Solve an LP with the ℓ_1-norm objective instead of the ℓ_0-norm in (3).
- `L0-exact (MIP)`: Solve (3) as a mixed integer programming problem.

We applied these methods to nine power network benchmarks obtained from [15]. We executed `hyp.min.s-t cut`, `min.s-t cut exact`, `min.s-t cut relax` and `L1-relax` for all arcs. We also applied `L0-exact` to all arcs for cases 1–5, and to randomly chosen 30 arcs for cases 6–9. All computations were done using Python (and internally C++) on a Mac desktop with 3.1GHz Intel CPU Core i7 and 16GB of memory. CPLEX was used for `L1-relax (LP)` and `L0-relax (MIP)`.

Figure 3 illustrates the mean 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` [1] for all instances. Recall that the sparsest attack problem can be solved by computing the security indices for all measurement points and adopting the minimum among them. The required time for this approach using these five methods for the security index problem can be estimated by multiplying the number of arcs. A single execution of `hyp.global min.cut` is faster than these estimates.

Table 1 shows the accuracy of the two relaxation methods. As shown in Figure 3, `min.s-t cut relax` is the fastest among the five methods for the security index computation. In Table 1, this method appears to perform better than `L1-relax (LP)`, but yet it sometimes fails in obtaining an optimal solution.

5 Enumerating Small Cuts in Hypergraphs

5.1 Overview

Our goal is to find all cuts U in a given hypernetwork $\mathcal{N}$ with $\kappa_{\mathcal{N}}(U) \leq k\lambda(\mathcal{N})$, called *$k$-small cuts*, for a given real $k \geq 1$ (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. [8] 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 $\mathcal{N}_n, \mathcal{N}_{n-1}, \ldots, \mathcal{N}_1$ of hypernetworks from input $\mathcal{N}$ to only one node by isolating and removing a node in each step such that, for each $i = 2, \ldots, n$, $\kappa_{\mathcal{N}_i}(U) \geq \kappa_{\mathcal{N}_{i-1}}(U)$ for every cut U in $\mathcal{N}_{i-1}$ and $\lambda(\mathcal{N}_i) \leq \lambda(\mathcal{N}_{i-1})$. After the construction, repeat checking the capacity $\kappa_{\mathcal{N}_i}(U)$ of each cut U in $\mathcal{N}_{i-1}$ with $\kappa_{\mathcal{N}_{i-1}}(U) \leq \beta$ and the capacity $\kappa_{\mathcal{N}_i}(\{v_i\})$ where v_i is isolated in the i-th step, i.e., $\{v_i\} = V(\mathcal{N}_i) \setminus V(\mathcal{N}_{i-1})$, and maintain the set of all cuts U with $\kappa_{\mathcal{N}_i}(U) \leq \beta$.

The key idea is to separate the operation called *edge-splitting*, which is used to isolate a node, into two types of operations, which we call *1-hyperedge-splitting* and *2-hyperedge-splitting*. The detail is shown later.

Let $h_k(\mathcal{N})$ denotes the maximum number of cuts of capacity at most $\beta = k\lambda(\mathcal{N})$ in $\mathcal{N}_i$, which is constructed through the algorithm, taken over $i = 1, \ldots, n$. One can see that the number of k-small cuts in a hypernetwork $\mathcal{N}$ is $O(n^{kM})$, where M denotes the maximum size of a hyperedge in $\mathcal{N}$, as a straightforward extension of Karger's bound [2] for an undirected graph. Our hyperedge-splitting, however, may increase M in constructing $\mathcal{N}_{n-1}, \ldots, \mathcal{N}_1$, 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 $k \geq 1$, one can enumerate all cuts U in $\mathcal{H}$ with $\kappa_{\mathcal{N}}(U) \leq k\lambda(\mathcal{N})$ in $O(|V| \cdot \|\mathcal{E}\|^2 + (h_k(\mathcal{N}) + |V|^2) \cdot \|\mathcal{E}\| \log |V|)$ time.*

The proof of this theorem is given at the end of this section.

5.2 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 is called *r-proper* if U is a cut in $\mathcal{H} - \{r\}$. The *r-connectivity* $\lambda_r(\mathcal{N})$ is the minimum capacity of an r-proper cut in $\mathcal{N}$. Hence $\lambda(\mathcal{N}) = \min\{\lambda_r(\mathcal{N}), \kappa_{\mathcal{N}}(\{r\})\}$. An r-proper cut U is called *r-tight* if $\kappa_{\mathcal{N}}(U) = \lambda_r(\mathcal{N})$.

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 T can be computed in $O(|V| \cdot \|\mathcal{E}\| + |V|^2 \log |V|)$ time.*

Proof. We use an algorithm of Klimmek and Wagner [3] for finding a minimum cut in a hypergraph shown as Algorithm 3 in Appendix. After the line 12 of Algorithm 3, the cut $\{t\}$ is guaranteed to be a minimum s–t cut [3, Lemma]. If we always take the designated node r as w in Algorithm 3, then r remains in $\mathcal{N}'$ until the final iteration. We claim that, if we replace the line 2 of Algorithm 3 by "**while** $|V| > 2$ **do**" and the line 3 by "$W \leftarrow \{r\}$", then the algorithm returns an r-tight cut in $\mathcal{N}$.

To see this, it suffices to check that each r-proper cut U in $\mathcal{N}$ has the same capacity as an s–t cut for some pair (s, t) after the line 12. Since $\mathcal{N}'$ has exactly two nodes including r when the algorithm halts, for any r-proper cut U in $\mathcal{N}$, in some iteration step we have $|U \cap \{s,t\}| = 1$, i.e., U is an s–t cut. Note that each cut U in $\mathcal{N}'$ remains with the same capacity after merging s and t if $|U \cap \{s,t\}| \neq 1$, which completes the proof. □

5.3 Weighted Hyperedge-Splitting

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 $\{e \subseteq V \mid |e| \geq 2\}$. In order to deal with each hyperedge e with $c(e) = 0$ as if it did not exist, let us define $\delta_{\mathcal{N}}(U) := \{e \in \delta_{\mathcal{H}}(U) \mid c(e) > 0\}$ for each $U \subseteq V$ and $\varGamma_{\mathcal{N}}(v) := \{u \in V - v \mid u \in e \in \delta_{\mathcal{N}}(v)\}$ for each $v \in V$. We define two types of procedures called *hyperedge-splitting*.

One is that, given a hyperedge $e_1 \in \delta_{\mathcal{N}}(r)$ and a nonnegative real $\alpha \leq \alpha_{\max} := c(e_1)/2$, we construct the following hypernetwork $\mathcal{N}' = (\mathcal{H}, c')$:

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

where $e' = e_1 - r$, and if $|e'| = 1$ 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 $\mathcal{N}/(e_1, \alpha)$.

The other is that, given two hyperedges $e_1, e_2 \in \mathcal{E}$ with $e_1 \cap e_2 = \{r\}$ and a nonnegative real $\alpha \leq \alpha_{\max} := \min\{c(e_1), c(e_2)\}$, we construct the following hypernetwork $\mathcal{N}' = (\mathcal{H}, c')$:

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

where $e' = e_1 \varDelta e_2 := (e_1 \setminus e_2) \cup (e_2 \setminus e_1) = e_1 \cup e_2 - r$. 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, e_2, α) (in the case of 1-hyperedge-splitting, regard e_2 as $\emptyset$), for any cut U with $r \notin U$ in $\mathcal{H}$, we have*

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

Hence $\lambda_r(\mathcal{N}') \leq \lambda_r(\mathcal{N})$ holds 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 $\mathcal{N}/(e_1, e_2, \alpha)$ for $0 \leq \alpha \leq \alpha_r(e_1; \mathcal{N})$ and $0 \leq \alpha \leq \alpha_r(e_1, e_2; \mathcal{N})$, respectively.

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 and e_2 with $e_1 \cap e_2 = \{r\}$),*

(i) *$\alpha_r(e_1; \mathcal{N})$ (or $\alpha_r(e_1, e_2; \mathcal{N})$) can be computed in $O(|V| \cdot \|\mathcal{E}\| + |V|^2 \log |V|)$ time, and*

(ii) *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 $e_1 \cap T \neq \emptyset$ and $e_2 \cap T \neq \emptyset$), which can be found in $O(|V| \cdot \|\mathcal{E}\| + |V|^2 \log |V|)$ time.*

Proof. 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 of **(i)** is obviously $\alpha_{\max}$ 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 get an r-tight cut $\tilde{T}$ in $\tilde{\mathcal{N}}$ that may not be r-tight in $\mathcal{N}$.

Because of Observation 3, the decrease of the capacity of each cut by hyperedge-splitting is almost uniform. In the case of 1-hyperedge-splitting, since the difference is always 2α if decreases, it is easily seen that

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

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

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

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 the desired value of **(i)** is $\tilde{\alpha}$ and $\tilde{T}$ is also r-tight in $\mathcal{N}' = \hat{\mathcal{N}}$. This $\tilde{T}$ must either include $e' = e_1 \Delta e_2$ or satisfy $e_1 \cap \tilde{T} \neq \emptyset$ and $e_2 \cap \tilde{T} \neq \emptyset$ by (4).

Otherwise, we get an r-tight cut $\hat{T}$ in $\hat{\mathcal{N}}$ that is not r-tight in $\mathcal{N}$. Note $\kappa_{\mathcal{N}}(\tilde{T}) - \kappa_{\hat{\mathcal{N}}}(\tilde{T}) \neq \kappa_{\mathcal{N}}(\hat{T}) - \kappa_{\hat{\mathcal{N}}}(\hat{T}) =: \alpha'$ and that we have $\kappa_{\mathcal{N}}(T') - \kappa_{\hat{\mathcal{N}}}(T') = \alpha'$ for any r-proper cut T' with $\kappa_{\hat{\mathcal{N}}}(T') < \lambda_r(\mathcal{N})$. Therefore, $\alpha(e_1, e_2; \mathcal{N})$ is equal to the right-hand side of (5) with $\hat{\mathcal{N}}$ and $\hat{T}$ instead of $\tilde{\mathcal{N}}$ and $\tilde{T}$ respectively, and $\hat{T}$ is also r-tight in $\mathcal{N}' = \hat{\mathcal{N}}$ with $e' = e_1 \Delta e_2 \subseteq \hat{T}$ or $e_1 \cap \hat{T} \neq \emptyset$ and $e_2 \cap \hat{T} \neq \emptyset$. □

The following lemma will be used later in the algorithm to isolate a node $r \in V$.

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

(i) *$\Gamma_{\mathcal{N}}(r) \setminus T \neq \emptyset$ holds.*

(ii) *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 $T' \subset T$ holds.*

(iii) *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 $T' \subset T$ holds.*

Proof. **(i)** Suppose to the contrary that some r-tight cut $T \subset V - r$ contains all neighbours of r. 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.
(ii) 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. 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

$$\kappa_{\mathcal{N}}(T) + \kappa_{\mathcal{N}}(T') \geq \kappa_{\mathcal{N}}(T \setminus T') + \kappa_{\mathcal{N}}(T' \setminus T) + c(e) > 2\lambda_r(\mathcal{N}) = \kappa_{\mathcal{N}}(T) + \kappa_{\mathcal{N}}(T'),$$

where the first inequality is easily checked by enumerating hyperedges that contribute to the capacities, a contradiction.
(iii) 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. Then T and T' are crossing, and hence by the similar observation as the proof of **(ii)**, we have

$$\kappa_{\mathcal{N}}(T) + \kappa_{\mathcal{N}}(T') \geq \kappa_{\mathcal{N}}(T \setminus T') + \kappa_{\mathcal{N}}(T' \setminus T) + c(e_1) > 2\lambda_r(\mathcal{N}) = \kappa_{\mathcal{N}}(T) + \kappa_{\mathcal{N}}(T'),$$

a contradiction. □

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 (i) and (ii) in Lemma 5 and a set Q_r of the information of all hyperedge-splittings.

```
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 $u \in \Gamma_{\mathcal{N}}(r)$.
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, 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 $Q_r$.
```

5.4 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 until there is no hyperedge in the temporary hypernetwork $\mathcal{N}^*$ that contains r. Note that the variable T^* usually indicates an r-tight cut in $\mathcal{N}^*$.

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

(i) *$\kappa_{\mathcal{N}_r}(U) \leq \kappa_{\mathcal{N}}(U)$ for every cut U in $\mathcal{N}_r$, and*

(ii) *$\lambda(\mathcal{N}_r) = \lambda_r(\mathcal{N}) \geq \lambda(\mathcal{N})$.*

Proof. First of all, we show that the output $\mathcal{N}_r$ satisfies the two conditions if the algorithm halts. Since the algorithm just performs hyperedge-splitting without destroying any r-tight cut in the while loop, it suffices to check the line 19. Suppose that 1-hyperedge-splitting e of weight $c(e)/2$ destroys some r-tight cut T. Then we must have $e \subseteq T + r$ by Observation 3, and the hyperedge $e \in \delta_{\mathcal{H}}(r)$ remains in $\mathcal{N}' := \mathcal{N}^*/(e, \alpha_r(e; \mathcal{N}^*))$ since $\alpha_r(e; \mathcal{N}^*) < c(e)/2$, contradicting **(i)** of Lemma 4.

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 is 0. These states are irreversible, i.e., the state index is monotone nondecreasing for each hyperedge throughout the algorithm. Moreover, after each hyperedge-splitting in the line 9 or 13, for at least one hyperedge, the state index

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 $i = n, n-1, \ldots, 2$ do
 3:     Take $v_i \in \arg\min_{v \in V_i} |\delta_{\mathcal{N}_i}(v)|$, where $V_i$ is the node set of $\mathcal{N}_i$.
 4:     Isolate $v_i$ in $\mathcal{N}_i$ by applying Algorithm 1, and let $\mathcal{N}_{i-1} \leftarrow \mathcal{N}_{v_i}$.
 5: end for
 6: $r \leftarrow v_1$ (reference node), and $\mathcal{C}_r^{\leq\beta}(\mathcal{N}_1) \leftarrow \emptyset$.
 7: for $j = 2, 3, \ldots, n$ 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 $\mathcal{C}_k(\mathcal{N}) := \mathcal{C}_r^{\leq\beta}(\mathcal{N}_n)$.
```

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 the irreversibility, we first show that T^* monotonically increases unless the line 4 is executed. This follows from **(ii)** and **(iii)** of Lemma 4. If $|T^*| > 1$, then T^* was updated in the line 16 in the previous iteration, and hence T^* is r-tight. Then, by applying Lemma 4 to T and T^* in the line 16 of the current iteration, we confirm $T^* \subset T$. Moreover, by Lemma 4, each extension of T^* involves increasing the state index of one of the selected hyperedges for hyperedge-splitting.

Finally, we consider update of T^* by the line 4. If the condition of the line 3 is true, i.e., $T^* \cap \delta_{\mathcal{N}^*}(r) = \emptyset$, then every $e \in \delta_{\mathcal{H}}(r)$ is in state (S1) or (S4), which completes the proof. □

5.5 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. [8], and it is briefly described in Section 5.1. Here we conclude this section with the proof of Theorem 3.

Proof of Theorem 3. The proof is almost the same as [8, Section 6]. One of the main differences appears in the line 10 of Algorithm 2, in which we compute the 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 the equation (4) that f_j and g_j defined as follows work well:

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

Another main difference appears in the part of bounding the running time of the lines 2–5. Note that we choose v_i as a minimizer of $\delta_{\mathcal{N}_i}(v)$ and hence $|\delta_{\mathcal{N}_i}(v_i)| \le \|\mathcal{E}\|/i$. This fact implies that the running time of the lines 2–5 is bounded as

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

Moreover, our update of the temporary family $\mathcal{C}_r^{\le\beta}(\mathcal{N}_i)$ of k-small cuts in $\mathcal{N}_i$ can be done $O(|\mathcal{C}_r^{\le\beta}(\mathcal{N}_i)| \cdot |Q_{v_i}|)$. Recall that $h_k(\mathcal{N}) = \max_i |\mathcal{C}_r^{\le\beta}(\mathcal{N}_i)|$. We have $|Q_{v_i}| = O(|\delta_{\mathcal{N}_i}(v_i)|)$ by Lemma 5, and hence the total computation is bounded by

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

which leads to the total running time shown in Theorem 3. □

References

1. 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, *arXiv:1204.6174*, 2012.
2. D. R. Karger: Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, **24** (1999), 583–614.
3. R. Klimmek and F. Wagner: A simple hypergraph min cut algorithm. *Internal Report B 96-02*, Bericht FU Berlin Fachbereich Mathematik und Informatik, 1995.
4. J.-M. Lin and H.-Y. Pan: A static state estimation approach including bad data detection and identification in power systems. *Proceedings of the IEEE Power Engineering Society General Meeting*, 1–7, 2007.
5. Y. Liu, P. Ning and M. K. Reiter: False data injection attacks against state estimation in electric power grids. *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 21–32, 2009.
6. S. G. Mallat and Z. Zhang: Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 3397–3415, 1993.
7. H. Nagamochi and T. Ibaraki: Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM J. Discrete Mathematics*, **5** (1992), 54–66.
8. H. Nagamochi, K. Nishimura and T. Ibaraki: Computing all small cuts in an undirected network. *SIAM J. Discrete Mathematics*, **10** (1997), 469–481.
9. J. B. Orlin: Max flows in $O(nm)$ time, or better. *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC 2013)*, 765–774, 2013.
10. J. Pistorius and M. Minoux: An improved direct labeling method for the max-flow min-cut computation in large hypergraphs and applications. *International Transactions in Operational Research*, **10** (2003), 1–11.
11. H. Sandberg, A. Teixeira and K. H. Johansson: On security indices for state estimators in power networks. *First Workshop on Secure Control Systems*, (CPSWeek 2010), 2010.
12. K. C. Sou, H. Sandberg and K. H. Johansson: Electric power network security analysis via minimum cut relaxation. *Proceedings of 50th IEEE Conference on Decision and Control and European Control Conference*, 4054–4059, 2011.
13. K. C. Sou, H. Sandberg and K. H. Johansson: On the exact solution to a smart grid cyber-security analysis problem. *IEEE Transactions on Smart Grid*, **4** (2013), 856–865.
14. H. H. Yang and D. F. Wong: Efficient network flow based min-cut balanced partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuit and Systems*, **15** (1996), 1533–1540.
15. 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 Transactions on Power Systems*, **26** (2011), 12–19.

Appendix

A Hypergraph Minimum Cut Algorithm

Here we describe an algorithm of Klimmek and Wagner [3] for finding a minimum cut in a hypergraph in order to use for proofs in Section 5. Here let $\delta_{\mathcal{N}}(U, W) := \{e \in \mathcal{E} \mid e \cap U \neq \emptyset,\ e \cap W \neq \emptyset,\ c(e) > 0\}$ for a hypernetwork $\mathcal{N} = ((V, \mathcal{E}), c)$.

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: $K \leftarrow +\infty,\ U \leftarrow \emptyset,\ \mathcal{N}' \leftarrow \mathcal{N}$.
 2: while $|V| \geq 2$ do
 3:     $W \leftarrow \emptyset$.
 4:     while $W \neq V$ do
 5:         Take a node $v \in \mathrm{argmax}_{u \in V \setminus W} \sum\{c'(e) \mid e \in \delta_{\mathcal{N}'}(\{u\}, W)\}$ and $W \leftarrow W + v$.
 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 $K > \kappa_{\mathcal{N}'}(\{t\})$ then
14:         Update $K \leftarrow \kappa_{\mathcal{N}'}(\{t\}),\ U \leftarrow \{v \in V \mid v$ was merged into $t\}$.
15:     end if
16:     Renew $\mathcal{N}'$ by merging $s$ and $t$ into a new node.
17: end while
```