Removing Undesirable Flows by Edge Deletion

Gleb Polevoy[1(✉)], Stojan Trajanovski[1,2], Paola Grosso[1], and Cees de Laat[1]

[1] University of Amsterdam, Amsterdam, The Netherlands
G.Polevoy@uva.nl
[2] Philips Research, Eindhoven, The Netherlands

Abstract. Consider mitigating the effects of denial of service or of malicious traffic in networks by deleting edges. Edge deletion reduces the DoS or the number of the malicious flows, but it also inadvertently removes some of the desired flows. To model this important problem, we formulate two problems: (1) remove all the undesirable flows while minimizing the damage to the desirable ones and (2) balance removing the undesirable flows and not removing too many of the desirable flows. We prove these problems are equivalent to important theoretical problems, thereby being important not only practically but also theoretically, and very hard to approximate in a general network. We employ reductions to nonetheless approximate the problem and also provide a greedy approximation. When the network is a tree, the problems are still MAX SNP-hard, but we provide a greedy-based $2l$-approximation algorithm, where l is the longest desirable flow. We also provide an algorithm, approximating the first and the second problem within $2\sqrt{2\,|E|}$ and $2\sqrt{2(|E| + |\text{undesirable flows}|)}$, respectively, where E is the set of the edges of the network. We also provide a fixed-parameter tractable (FPT) algorithm. Finally, if the tree has a root such that every flow in the tree flows on the path from the root to a leaf, we solve the problem exactly using dynamic programming.

1 Introduction

Attacks such as the (distributed) Denial of Service (DDoS) [14] are widespread [19] and heavily impede the functionality, especially when the system is required to be quick (soft real time, for example) [13]. One of the options to fight the problem is deleting network edges or disabling them (anyway, deleting from the network graph) [11]. Another practically important problem is having malicious connections, like Trojans. The danger is not only the bandwidth these connections take but primarily the information they transfer.

We define a flow as a path from the source to the sink and we model DoS or merely malicious communication as a set of undesirable (name them *bad*) flows. The system has also desirable (call them *good*) flows. We aim to remove the undesirable flows by deleting some edges on their paths, while minimizing the resulting damage to the desirable flows. The model assumes no rerouting of

D. Kim et al. (Eds.): COCOA 2018, LNCS 11346, pp. 217–232, 2018.
https://doi.org/10.1007/978-3-030-04651-4_15

flows. If we delete an edge on the path of a bad flow, we consider that flow to be removed, but we also inadvertently remove all the good flows that pass through the deleted edge.

We can identify the DoS flows by frequent access trials from the same IP group, and the malicious flows can be identified by information leaks. Therefore, we assume we know which flows are good and which are bad.

Judicious deletion is crucial, as Example 1 shows. We need to autonomously decide which edges to delete as suggested in [11]. This calls for an algorithm to find which edges to delete. In order to cope with large instances in real time, the algorithm has to be polynomial.

Example 1. In Fig. 1, removing all the bad flows by deleting their only common edge e would remove no good flows. This is infinitely better than removing each bad flow b_i by deleting any of its edges other than e, because that would also remove g_i.

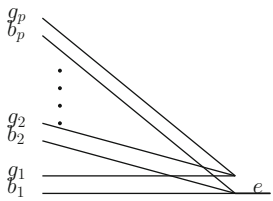

Fig. 1. The bad flows are denoted by b with an index, while the good ones are denoted by g with an index. The paths of flow b_i and g_i coincide, besides edge e, belonging only to the bad flows.

Consider this example of using an algorithm that finds which edges to delete.

Example 2. In a (for example, software-defined) network, assume that the intrusion detection system discovers a DDoS attack, and determines which flows are attacking. We need to respond quickly and efficiently by removing certain links from the network. Assuming we know which flows are desirable and which are attacking, we first estimate how important it is not to remove each desirable flow by deleting a link. Now, we run our algorithm to obtain an (approximately) easiest set of the edges to delete, such that all the attacking flows are removed (disconnected) while the minimum damage is inflicted on the desirable flows.

We present the necessary background in Sect. 1.1. To solve the problem, in Sect. 2 we model the situation as two possible problems. First, we show in Sect. 3 that our problems are equivalent w.r.t. approximation to notorious hard problems. Those hard and important problems admit polynomial approximations, but those are extremely loose. We then reduce the problems to submodular set cover to use approximation known for that problem and also suggest a greedy

approximation in Sect. 4. We approximate the important particular case when the network is a tree in Sect. 5, which is still MAX SNP-hard. We assume in Sect. 5.3 that the tree has a root such that every flow is on the path from the root to a leaf, as often happens when communicating through service providers to the backbones, and solve this case exactly using dynamic programming. Finally, we provide a fixed parameter tractable (FPT) algorithm for trees in Sect. 5.4. These results suggest determining the kind of network we have at hand and subsequently applying the best applicable algorithm. We summarize our approach and suggest further research directions in Sect. 6.

We approximate the problem in general and provide additional approximation and exact algorithms for special cases such as trees.

1.1 Related Work

We study edge deletion that removes the bad flows and does not remove too many good ones, which is a new problem in the realm of edge deletion problems. The simpler problem of deleting the minimum number of edges that can disconnect all the flows from a source to a sink is a famous problem, and Menger's theorem [2, Chap. 3.2] characterizes the minimum number of edges one has to remove in order to disconnect the source from the sink. Finding a minimum cut in a graph [4, Chap. 26] and disconnecting it is optimal for this problem.

There are many other problems of edge deletion, such as deleting the minimum number of edges to obtain certain properties like no cycles of various lengths [18], or removing forbidden graphs [17]. Similar network design studies include other problems, such as edge addition [10]. In practice, edge deletion can be especially easily implemented in a software-defined network. See [11] for a practical example.

There are many important vertex deletion problems as well, such as the famous vertex cover problem [7]. Other vertex deletion problems include the feedback vertex set, where we aim to break all the cycles in a graph [1], and the problem of breaking all the cycles of a given length or at most a given length [18].

2 Model

Let the flow network be a directed graph $G = (N, E)$. A flow f from node a to node z in this network is a path from source a to sink z, each of which edges carries the flow. Formally, $f = (P(f))$, where $P(f)$ is the set of the edges of the path that the flow takes from a to z. Flow in this paper are not splittable, meaning that a flow takes a single path. This can also model a splitting flow as separate flows with partially overlapping paths. We do not model the capacities of the edges and the values of the flows, because these notions are irrelevant to the problem. Of course, real flows have values and real edges (say, wires or roads) have capacities, and the total flow value on an edge cannot exceed the capacity of the edge.

We model removal of undesirable flow as two problems. The first one follows:

Definition 1. *The* Bad Flow Removing (BFR) *problem receives the input* $(G = (N, E), F, GF, BF, w\colon GF \to \mathbb{R}_+)$. *Here,* $G = (N, E)$ *is a network with flows* $F = \{f_i\}$, *where some flows, denoted* $GF = \{g_i\} \subseteq F$, *are marked as good (desirable), and the rest, denoted* $BF = \{b_i\} \triangleq F \backslash GF$, *are bad (undesirable). Every good flow* f *is endowed with a weight* $w(f)$, *designating how important the good flow is.*

A solution $S \subseteq E$ *is a subset of edges to delete.*

Flow f *is* removed *by* S *if* $S \cap P(f) \neq \emptyset$; *otherwise, it is remaining. A* feasible solution *is a solution such that all the bad flows are removed.*

We aim *to find a feasible solution with the minimum total weight of the removed good flows. Intuitively, we aim to remove all the bad flows while minimizing the weight of the removed good flows.*

BFR assumes we have to get rid of all the bad flows. For instance, when flows can spread malicious content or steal information, leaving even few flows can harm the network. Since sometimes bad flows are mostly taking resources and are not so dangerous, we may merely want to remove most of bad flows while still not losing too many good ones, so we relax the requirement to remove all the bad flows and allow leaving them for a cost:

Definition 2. *The* Balanced Bad Flow Removing (BBFR) *problem receives* $(G = (N, E), F, GF, BF, w\colon F \to \mathbb{R}_+)$; *the difference from the BFR is that all the flows are weighted.*

Here, a solution *is the same as in BFR, but any solution is* feasible *here.*

We aim *to find a feasible solution such that the total weight of the remaining bad flows plus the total weight of the removed good flows is minimized. Intuitively, we aim to balance removing the bad flows and not removing too many good ones.*

Denote by $D(S)$ the flows removed from the set of flows D by deleting the edges S. We define the weight of a set of flows $D \subset F$ as $\sum_{f \in D} w(f)$, which we denote as $w(D)$, abusing the notation w. Therefore, BFR removes all the bad flows ($BF(S) = BF$) while aiming to minimize $w(GF(S))$, while BBFR targets to minimize the $w(BF \backslash BF(S)) + w(GF(S))$. We do not add a balancing parameter such as in $w(BF \backslash BF(S)) + \alpha w(GF(S))$, because the weights can be defined already with balancing in mind, modeling such a balancing.

This formalism allows to prune all the edges that do not belong to good flows, so we can assume that all edges belong to at least one good flow. On the other hand, if one wants to avoid deleting edges lightly, one can model this by introducing dummy good flows of length 1 through all the edges.

3 Equivalence Resulting in Hardness and Approximation

We prove that Problems BFR and BBFR, are equivalent w.r.t. approximation to the classical Red-Blue Set Cover [3] and to the Partial Set Cover [12] problems, respectively. This immediately implies strong inapproximability and provides some approximation algorithms.

3.1 BFR

We first formally define the RBSC problem.

Definition 3. *The* Red-Blue Set Cover (RBSC) *problem* [3] *receives the input* $(R, B, \mathcal{S}, w\colon R \to \mathbb{R}_+)$*, where* R *and* B *are two disjoint sets of red and blue elements and* $\mathcal{S} \subseteq 2^{R \cup B}$*, i.e. every set* S *in the collection* $\mathcal{S}$ *is a subset of* $R \cup B$*. These subsets can cover all the blue elements, i.e.* $B \subseteq \cup_{S \in \mathcal{S}} S$*. Finally,* w *denotes the weight of the red elements.*

A solution *is a subcollection* $\mathcal{C}$ *of* $\mathcal{S}$*.*

A feasible solution *is a solution* $\mathcal{C}$ *that covers all the blue elements, i.e.* $B \subseteq \cup_{S \in \mathcal{C}} S$*. The aim is to find a feasible solution with the minimum total weight of the covered red elements,* $w(\mathcal{C}) \triangleq \sum_{r \in R \cap \{\cup_{S \in \mathcal{C}} S\}} w(r)$*.*

Theorem 1. *BFR is equivalent w.r.t. approximation to RBSC. Reducing from RBSC to BFR, we can ensure that* $|E| = 2\,|\mathcal{S}|$*,* $|GF| = |R| + 1$ *and* $|BF| = |B|$*. Reducing from BFR to RBSC,* $|\mathcal{S}| = |E|\,, |R| = |GF|$ *and* $|B| = |BF|$*.*

Proof. We first reduce RBSC to BFR. For each set $S \in \mathcal{S}$, define edge e_S and let all the edges have a common node. Define good flow g_i for each red element $r_i \in R$, with the same weight, and bad flow b_j for each blue element $b_j \in B$, such that the path of a flow contains edge s_S if and only if the element from which the flow has been created is included in S. To be able to route flows through the required edges from $E(\mathcal{S}) \triangleq \{e_S | S \in \mathcal{S}\}$, we add edges, called E', that connect the non-common nodes of the edges from $E(\mathcal{S})$, as illustrated in Fig. 2. To prevent choosing the edges from E', we define an additional good flow that contains every edge from E', and we give this good flow a prohibitively high weight, say $2 \sum_{g \in GF} w(s)$. The solutions to the constructed BFR are directly transformed to solutions for RBSC, besides the case when a solution for BFR contains an edge from E'. Such solutions are transformed to the trivial solution for RBSC that contains all the sets. Unless edges from E' are selected for the solution for BFR, the weight of the covered red elements is equal to the weights of the removed good flows. If edges from E' are selected, then the cost of that solution is at least twice higher than the cost of the corresponding solution for RBSC. Therefore, the reduction preserves approximation.

We now reduce BFR to RBSC. Make a red element from a good flow, a blue element from a bad flow and a set from an edge, such that the set contains exactly the flows that have the edge on their paths. The weights are transferred as they are. The solution to the obtained RBSC instance is mapped to a solution for the original BFR in the reverse manner. This mapping preserves being a solution, feasibility and the weights. ■

This theorem immediately implies that all the hardness and all the positive results for RBSC transfer to BFR. In particular, Theorems 3.1 and 3.2 from Sect. 3 of [3] imply the following.

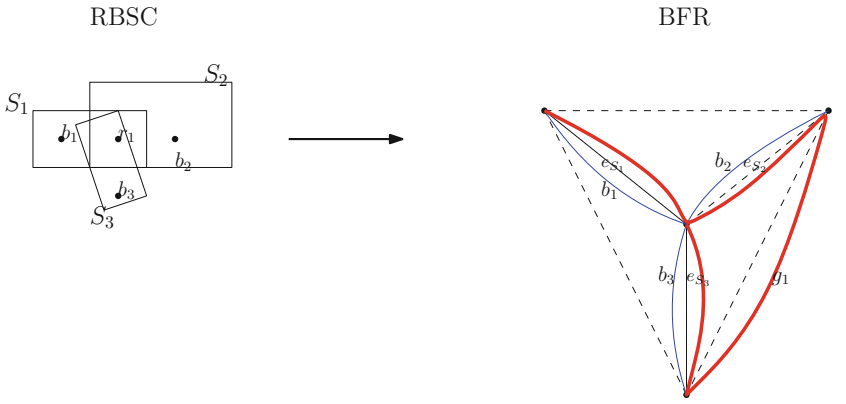

Fig. 2. On the right, the bad flows are denoted by b with an index, while the good ones are denoted by g with an index. The dashed lines are the E' edges.

Corollary 1. *1. Unless NP* $\subseteq$ *DTIME*$(n^{\text{polylog}(n)})$*, it is impossible to approximate BFR within* $O(2^{\log^{1-\delta}|E|})$*, for any* $\delta > 0$*.*

2. Unless P = NP, it is impossible to approximate BFR within $O(2^{\log^{1-\delta}|E|})$*, where* $\delta = 1/\log\log^c|E|$*, for any constant* $c < 0.5$*.*

Proof. The existing hardness results state the impossibility to approximate within $O(2^{\log^{1-\delta}|\mathcal{S}|})$, for certain values of δ. If BFR could be approximated within $O(2^{\log^{1-\delta}|E|})$, reducing RBSC to BFR would allow approximating RBSC within $O(2^{\log^{1-\delta}2|\mathcal{S}|})$, contrary to the impossibility. ■

On the approximation side, Theorems 3.5 and 3.6 from [15] imply that.

Corollary 2. *BFR can be approximated within* $2\sqrt{|E|\log|BF|}$*.*

Proof. The approximation follows by reducing the given instance of BFR to an RBSC, employing Algorithm Low_Deg2 from Sect. 3 of [15] on the obtained instance of RBSC, and then translating the obtained cover to a set of the corresponding edges. The transformations from BFR to RBSC and back preserve the approximation ratio, and the approximation of the algorithm, $2\sqrt{|\mathcal{S}|\log|B|}$, is equal to $2\sqrt{|E|\log|BF|}$, since each edge defines a set in the reduction of BFR to RBSC. ■

3.2 BBFR

For the sake of handling the BBFR problem, we define a problem we will reduce to and from, namely the Positive-Negative Partial Set Cover problem, which generalizes the unweighted definition from Sect. 1.1 from [12].

Definition 4. *The Positive-Negative Partial Set Cover (±PSC) problem receives the input* $(R, B, \mathcal{S}, w : R \cup B \to \mathbb{R}_+)$*, where* R *and* B *are two disjoint*

sets of red and blue elements and every set S in the collection $\mathcal{S}$ is a subset of $R \cup B$. Finally, we denote by w the weight of the elements.

A solution *is a subcollection $\mathcal{C}$ of $\mathcal{S}$.*

Any solution is feasible *here. The aim is to find a feasible solution with the minimum total weight of the uncovered blue elements plus the covered red elements,* $w(\mathcal{C}) \triangleq \sum_{b \in B \setminus \cup_{S \in \mathcal{C}} S} w(b) + \sum_{r \in R \cap \{\cup_{S \in \mathcal{C}} S\}} w(r)$.

We now reduce BBFR to and from ±PSC, omitting the proofs to save space.

Theorem 2. *BBFR is approximation-equivalent to ±PSC. Reducing from ±PSC to BBFR, we can ensure that* $|E| = 2\,|\mathcal{S}|$, $|GF| = |R| + 1$ *and* $|BF| = |B|$. *Reducing from BBFR to ±PSC,* $|\mathcal{S}| = |E|\,, |R| = |GF|$ *and* $|B| = |BF|$.

This theorem implies that the hardness results and the approximation results transfer from ±PSC to BBFR. In particular, the following holds:

Corollary 3. *1. Unless NP* $\subseteq DTIME(n^{\mathrm{polylog}(n)})$, *it is impossible to approximate BBFR within* $O(2^{\log^{1-\delta}|E|})$, *for any* $\delta > 0$.

2. *Unless P = NP, it is impossible to approximate BBFR within* $O(2^{\log^{1-\delta}|E|})$, *where* $\delta = 1/\log\log^c |E|$, *for any constant* $c < 0.5$.
3. *Unless P = NP, there is no approximation to BBFR within* $O(2^{\log^{1-\delta}|BF|})$, *for any* $\delta > 0$.

As for approximation, Corollary 3 from [12] implies.

Corollary 4. *BBFR is approximable within* $2\sqrt{(|E| + |BF|)\log(|BF|)}$.

4 Approximation

First, consider the approximation for submodular cost set cover within the maximal frequency of an element from [9, Sect. 4.4]. Since the total weight of the removed good flows is a submodular function of the chosen edge subset, that approximation applies to BFR. The maximal frequency of an element becomes the maximal number of edges a bad flow contains, i.e. l', which is the approximation ratio for BFR. We can also approximate BBFR using the following theorem.

Theorem 3. *BBFR can be reduced to BFR while preserving approximation by extending the path of each bad flow at its end by an edge with a new second node and defining a new good flow with the path that consists solely of this new edge and with the weight* $w(b_i)$.

Proof. Miettinen [12, Sect. 2.2] reduces ±PSC to RBSC, inspiring us to the following reduction of BBFR to BFR. Given a BBFR instance $(G = (N, E), F, GF, BF, w\colon F \to \mathbb{R})$, construct the following BFR instance $(G = (N', E'), F', GF', BF', w'\colon GF \to \mathbb{R})$. We extend the path of each bad flow

b_i at its end by an edge with a new second node and we also define a new good flow with the path that consists solely of this new edge. The weight of the new good flow is defined to be $w(b_i)$, and the function w is restricted to GF, to obtain a BFR. We transform a solution S to this BFR to the solution $S \cap E$ for the original BBFR.

Similarly to [12, Sect. 2.2], the approximation $S \cap E$ provides for the original BBFR is at least as good as the approximation S provides for BFR. ■

This reduction increases the maximum length of a bad flow by 1, and thus, allows approximating BBFR withing $(l' + 1)$.

We also propose the following greedy algorithm, inspired by the famous greedy algorithm for set cover [16, Algorithm 2.2] and the greedy algorithm from [15, Sect. 3.1].

ALGORITHM 1. GreedyBFR$(G = (N, E), F, GF, BF, w)$

1. Given a BFR instance, define the following weighted set cover instance.
   (a) the elements are the bad flows with all edges intersecting good flows;
   (b) the sets are the good flows, a good flow covering all the bad flows it intersects.
2. Approximately solve this set cover instance, obtaining the output $S \subseteq GF$.
3. Return the edge set of S, i.e. $\cup_{g \in S} P(g)$, augmented with edges of bad flows intersecting no good flows.

Proposition 1. *Algorithm 1 approximates BFR within factor* $k \cdot (\ln(|BF|) + 1)$.

Proof. Algorithm 1 returns a feasible solution, since all the bad flows intersecting at least one good flow can be covered by the edges of the good flows.

As for the approximation ratio, call the original problem I and let I' denote the problem we construct in line 1. Problem I' models directly removing the bad flows by removing the good ones that intersect them, ignoring the fact that removing occurs through deleting edges, which can remove several intersecting good flows simultaneously. Therefore, $\text{OPT}(I') \leq \text{OPT}(I)$, and so the $\ln(|BF|)+1$-approximation to I' costs at most $\ln(|BF|) + 1$ times $\text{OPT}(I)$. We do have to add the intersecting good flows to ensure feasibility. This action can require the k factor, implying the proposition. ■

Theorem 3 implies we can approximate BBFR within the same factor. Theorems 1 and 2 imply that this algorithm allows approximating the Red-Blue Set Cover and the Positive-Negative Partial Set Cover within $k \cdot (\ln(|B|) + 1)$, k being the maximum number of the red elements that a red elements can have a common set with.

Remark 1. Using the 2-approximation for set cover on trees [8, Sect. 7] instead of the general approximation for set cover, we can adopt Algorithm 1 to approximate BFR on trees within $2k$. As before, Theorem 3 implies the same approximation factor for BBFR as well.

Similarly to Peleg [15, Sect. 3.2], we could continue with dividing the good flows to those intersecting many other good flows and not, approximating the problem with the good flows intersecting not too many other good flows and bounding the number of good flows that intersect many others. We omit the details, because this approach would yield the same approximation ratios as Peleg receives for Red-Blue Set Cover.

5 Trees

We showed that the problems BFR and BBFR are extremely hard to approximate, though we still provided approximations. Given the proven hardness of the general case, we now concentrate on trees, often found in communication networks. Trees subsume star networks, for instance. For trees, we can do more.

5.1 Hardness

We will show that BFR and BBFR problems are MAX SNP-hard even for trees. First, we prove an important result, connecting the two problems.

Theorem 4. *BFR is reducible to BBFR with an approximation-preserving reduction that preserves the problem instance, besides assigning weights to the bad flows.*

Proof. Consider the following reduction. Given an instance x of BFR, construct an instance x' of BBFR by defining the weight of each bad flow to be $2\sum_{g\in GF} w(g)$. Then, a solution y' of x' is directly transformed to a solution y of x, unless y' does not cover all the bad flows. In the latter case, y' is transformed to the trivial solution for x that simply contains all the edges. Now, the weights of the solutions y and y' are equal, unless y' does not cover all the bad flows. In the latter case, however, the weight of y' is at least twice larger than that of y, by the definition of the weights of the bad flows. ■

We are now set to prove the following hardness result.

Proposition 2. *Even on trees of height* 1 *and good flows of unit weights, the problems BFR and BBFR are MAX SNP-hard and not approximable within* 1.166*.*

The proof reduces BFR to the tree set cover and is omitted for lack of space.

5.2 Approximation

Having shown the hardness, we design two approximation algorithms for BFR. Later, we show that similar results hold for BBFR as well.

We now suggest other algorithms for BFR on trees that approximate it within $2l$ and $2\sqrt{2\,|E|}$. These algorithms refine the **Greedy_RB** and **Low_Deg2** algorithms, respectively, for RBSC from [15]. Recall that l is defined as

$\max\{|P(g) : g \in GF|\}$, the maximum length of a good flow. We first assume a non-weighted BFR; the extension for the weighted case is straight-forward.

First, in a tree Set Cover can be approximated within the factor of 2 [8, Sect. 7]. This allows, extending [15, Sect. 3.1], to reduce BFR to Set Cover on trees and obtain the approximation ratio of $2l$. Call this algorithm **SCTreeGreedy**. This is a useful algorithm on its own right, and we continue now to obtain a $2\sqrt{2\,|E|}$- approximation.

In order to present the final approximation, we first present Algorithm 2, inspired by **Low_Deg** from [15].

ALGORITHM 2. Low_Deg_TreeBFR$(G = (N, E), F, GF, BF, w, x)$

1. Remove the edges in E that belong to more than x good flows, creating E' and defining a new problem instance $(G' = G[E'], F', GF', BF', w, x)$, where $G[E']$ is the subgraph induced by E', and so are the flows F', GF' and BF'.
2. **If** the new instance is infeasible, **return** E (the trivially feasible solution).
3. Let the long good flows be $GF'_l \triangleq \left\{g \in GF' : |P(g)| > \sqrt{|E|/2}\right\}$.
4. Leave only the not long good flows, i.e. $GF'_s \triangleq GF' \setminus GF'_l$.
5. **Return SCTreeGreedy**$(G' = G[E'], F', GF'_s, BF', w, x)$.

Similarly to Lemma 3.3 from [15], we prove Lemma 1.

Lemma 1. *Let x be the last input to Algorithm 2 and let the long good flows of GF' be $GF'_l \triangleq \left\{g \in GF' : |P(g)| > \sqrt{|E|/2}\right\}$. Then, $|GF'_l| < \sqrt{2\,|E|}x$.*

Proof. Since every edge in E' belongs to at most x good flows, we have

$$|GF'_l|\sqrt{|E|/2} < \sum_{g\in GF'_l} |P(g)| \le \sum_{g\in GF'} |P(g)| = \sum_{e\in E'} |\{g \in G | e \in P(g)\}| \le |E'|\,x \le |E|\,x,$$

the first inequality stemming from the definition of GF'_l, and the equality being a reversal of the summation order. Therefore, $|GF'_l| < |E|\,x/\sqrt{|E|/2} = \sqrt{2\,|E|}x$. ■

And we can now prove.

Lemma 2. *Let S^* be an optimal solution for the BFR instance at hand (if we knew it). If we activate Algorithm 2 with $\hat{x} \triangleq \max|\{g : GF | e \in P(g) \cap S^*\}|$, then it returns a $2\sqrt{2\,|E|}$-approximation.*

Proof. By definition of $\hat{x}$, the algorithm returns a feasible solution in line 5.

Since **SCTreeGreedy** yields an $2l$-approximation and its input has $l \le \sqrt{|E|/2}$, then its solution, say S, fulfills $|GF'_s(S)| \le \sqrt{2\,|E|}\,|GF'_s(S^*)|$. Lemma 1 applies that $|GF'_l| < \sqrt{2\,|E|}\hat{x}$. Therefore, the total number of removed good flows is $|GF(S)| < \sqrt{2\,|E|}\,|GF'_s(S^*)| + \sqrt{2\,|E|}\hat{x}$. Since $\hat{x} \le |GF(S^*)|$, we can bound $|GF(S)|$ by $2\sqrt{2\,|E|}\,|GF(S^*)|$. ■

We are now ready to present the full algorithm for approximating the actual problem, where we do not know the $\hat{x}$ in advance.

ALGORITHM 3. Low_Deg_TreeBFR2$(G = (N, E), F, GF, BF, w)$

1. min_sol $\leftarrow E$
2. **for** $x = 1 \ldots |GF|$ **do:**
   (a) $S \leftarrow$ **Low_Deg_TreeBFR** $(G = (N, E), F, GF, BF, w, x)$
   (b) **if** $w(GF(S)) < w(GF(\text{min\_sol}))$:
   min_sol $\leftarrow S$
3. **return** min_sol

Lemma 2 implies the following.

Theorem 5. *Algorithm 3 approximates the solution to BFR within* $2\sqrt{2\,|E|}$.

This has been proven for the non-weighted case, but is straight-forward to extend the result to the weighted case.

Having found approximations for BFR within $2l$ and $2\sqrt{2\,|E|}$, we approximate BBFR:

Theorem 6. *BBFR can be approximated within* $2l$ *and within* $2\sqrt{2(|E| + |BF|)}$.

Proof. We employ the approximation preserving reduction from Theorem 3. The new graph is still a tree, because we have added connected edges and created no cycles. We can, therefore, solve it using an algorithm for BFR on trees.

If we employ the $2l$ approximation, we obtain the same approximation ratio for BBFR, since only the length of the bad flows has increases when reducing from BBFR to BFR. Alternatively, if we employ the $2\sqrt{2\,|E|}$-approximation, we obtain a $2\sqrt{2(|E| + |BF|)}$-approximation, since the reduction introduces $|BF|$ new edges. ■

5.3 Trees with Root-to-Leaf Flows

Since communication often goes from the clients to the Internet Service Providers (ISPs) and then to the backbone, we assume that the network is a *tree*, and there exists a fixed node r called root, such that every flow is on a path from the root to a leaf.

This assumption allows us to solve both BFR and BBFR exactly using Dynamic Programming (DP). We define a subproblem of our DP to be a subtree and the flows that strictly flow through its root after possible edge deletions outside this subtree. Let $v \in N$ be a node and let $P(v)$ be the set of the edges on the (only) path from v to the root of the tree. Let $F(v) \triangleq \{f \in F : v \text{ is on } P(f)\}$ be the set of the originally given flows that pass through v. Denote by $T(v)$ the

subtree rooted on v. The possible subsets of flows that enter $T(v)$ after deleting some edges outside of $T(v)$ are $\mathcal{F}(v) = \{F(v) \backslash F(v)(\{e\}) : e \in P(v)\}$[1]. We do not consider deleting a subset of edges on $P(v)$, because for $\mathcal{F}(v)$ it would be equivalent to deleting the edge of this subset that is closest to v.

The DP Algorithm 4 receives a root r such that every flow is on a path from r to a leaf and solves BFR and BBFR exactly.

ALGORITHM 4. DPAlg_TreeRoottoLeaf$(G = (N, E), F, GF, BF, w, r)$

1. The algorithm maintains the DP-table indexed by $\{v, \mathcal{F}(v) : v \in N \setminus \{r\}\}$.
2. **For each** node $v \in N \setminus \{r\}$ in a **post-order** traversal (i.e. its subtree has been handled):
   (a) **For each** $S \in \mathcal{F}(v)$:
      i. Delete the edge from v to its parent $\iff$ it maximizes the total objective function in $T(v)$. This uses the optimal solutions that we have memoized for the children of v.
      ii. Memoize the resulting edge deletion and the resulting objective function for the current entry $(v, S) \in \{v, \mathcal{F}(v) : v \in N \setminus \{r\}\}$.
3. The completed DP-table contains an optimal set of edge deletions.

Theorem 7. *Algorithm 4 optimally solves the problem on trees when all the flows are from the root to leaves in* $O(|N|^3 |F|)$.

Proof. The algorithm is correct, since all the flows go from the root to a leaf, thereby making such a traversal consider all the relevant edge deletions.

For each entry in the DP-table, the algorithm looks at all the children of the current node. There are at most $|N|^2$ entries, because each $\mathcal{F}(v)$ contains $|P(v)|$ elements, as defined and explained above. Each entry requires looking at all the flows that pass through the node, providing the factor of $|F|$. In addition, a node has at most $|N| - 1$ children, implying the theorem. ■

In the rest of the section, when a flow from a to z does not flow from the root r to leaves, we may split it to two parts that do by looking at the first node on the (only) path from a to r that is also on the (only) path from r to z.

Approximation when All the Bad Flows are Root-to-Leaf. If we find a root such that the bad flows can be guaranteed to be from that root to leaves, we can 2-approximate the problem as follows. Given such an instance I and a root r we define another instance I' by splitting each good flow g that does not flow from r to a leaf to two good flows that do. Define the weight of each one of the obtained good flows to be $w(g)$. Denote the weight of the optimal solution to an instance by OPT(instance). Then, $\text{OPT}(I') \leq 2\text{OPT}(I)$. Therefore, we can solve I' using Algorithm 4 and this will constitute a 2-approximation for I.

[1] We remind that for a set of flows D and a set of edges S, we denote by $D(S)$ the flows from D removed by deleting S.

FPT when All the Good Flows are Root-to-Leaf. If we actually find a root such that the good flows are from the root to leaves, we can split each bad flow that does not go from the given root to a leaf by splitting each bad flow b that does not flow from r to a leaf to two bad flows that do. Remove one of the parts and leave the other part. If we are given a BBFR instance, assign the remaining part of the bad flow the weight of the original bad flow. We solve each of the obtained problems for each such a split of the bad flows and output the best solution. This constitutes an optimal algorithm that runs in $O(2^{|BF|})$ times a polynomial time.

5.4 FPT for Trees

Section 5.3 assumes some of the flows are from the root to leaves. We now advance and prove that when the flows follow any simple paths, meaning that a flow does not intersect itself, then the problem is fixed-parameter tractable, parametrized by the number of the bad and the number of the good flows.

We first present the definition of a parametrized optimization problem and fixed-parameter tractability with several parameters, adapted from [5,6].

Definition 5. *A* parametrized optimization problem with $t \in \mathbb{N}$ parameters *is a set of instances $\Sigma^* \times \mathbb{N}^t$, where Σ is a finite alphabet, encoding the object at hand, and the t natural numbers are called the* parameters.

For example, set cover parametrized by the total number of the elements is a parametrized optimization problem. Tractability is defined as follows.

Definition 6. *A parametrized optimization problem with t parameters, consisting of the instances $\Sigma^* \times \mathbb{N}^t$, is called* fixed-parameter tractable (FPT) *if there exists an algorithm A, a computable function $f\colon \mathbb{N}^t \to \mathbb{N}$, and a constant c such that if A receives instance $(x, k_1, \ldots, k_t) \in \Sigma^* \times \mathbb{N}^t$, it computes an optimal solution to it within at most $f(k_1, \ldots, k_t)(|x| + k_1 + \ldots + k_t)^c$ time.*

Intuitively speaking, only the parameters may contribute more than a polynomial to the run time. We are now ready to prove that BFR and BBFR, when the network is a tree and the flows have simple paths, is FPT. Consider Algorithm 5.

Notice that step 2 follows the approach of Sect. 5.3, only that now there is no assumption about flowing from the root to the leaves.

We summarize Algorithm 5 in the following theorem.

Theorem 8. *Algorithm 5 runs in time $O(2^{|BF|}4^{|GF|}\,|BF|\,|GF|\,|N|^3\,|F|)$.*

Proof. The algorithm is correct, because it goes over all the possibilities to remove bad flows and good flows. A bad flow can be removed be deleting either edge on its path, so we just choose the best option. A good flow has to be paid for only once, if it is removed at all, and so we nullify the weight of a good flow if the other split part has been removed.

As for the time complexity, maintaining the splits of the bad and of the good flows takes the factor of $|BF|\,|GF|$. The algorithm first goes over $O(2^{|BF|})$

ALGORITHM 5. DPAlg_Tree$(G = (N, E), F, GF, BF, w)$

1. Arbitrarily pick a node to be the root. Call it r.
2. Split each bad flow that does not flow from the root to a leaf to two parts that do.
3. Delete one part, and if this is a BBFR instance, define the weight of the remaining part to be the weight of the original flows. For each bad flow that does not flow from r to a leaf, there are 2 options as to which part to delete. For each set of options, do:
   (a) For each good flow that has a path not from r to a leaf, split it to two parts that do, and assign each part the weight of the original good flow.
   (b) Solve the obtained instance using Algorithm 4 with the following adjustment. If the dynamic programming decides to delete an edge from a split part of a good flow, then it has to assign the second part of that flow zero weight (in its subtree). Accounting for 2 options per a split good flow each time requires 2 attempts per each split good flow that enters the subtree.
4. Return the best solution from all the solutions in the above tried combinations.

splitting options for bad flows (i.e. which part of each split bad flow to delete). For each such an option, it splits the good flows and runs Algorithm 4, while trying all the options for the weights of each split good flow, i.e. $O(2^{2|GF|})$. Employing Theorem 7, the total resulting time is

$$O(|BF|\,|GF| \cdot 2^{|BF|} \cdot 2^{2|GF|}\,|N|^3\,|F|) = O(2^{|BF|}4^{|GF|}\,|BF|\,|GF|\,|N|^3\,|F|).$$

■

This immediately implies the following.

Corollary 5. *BFR and BBFR parametrized by $|BF|$ and $|GF|$ are FPT.*

6 Conclusion

We study two problems that model fighting DoS and malicious communication: BFR and BBFR. We need to delete edges so that the bad (undesirable) flows are disconnected. Unlike the usual network design problems [18], we do not merely minimize the number of the deleted edges, but rather the resulting number of the disconnected good (desirable) flows. We prove that in the general setting, these problems are extremely hard to approximate, being approximation equivalent to hard problems. We reduce our problems to submodular set cover to provide a approximation and provide a greedy approximation as well. In the important case when the network is a tree, the problems are still MAX SNP-hard, and we provide an approximation algorithm. Furthermore, if the tree can be rooted such that every flow is on the path from the root to a leaf, we solve the problems exactly using dynamic programming (DP). This also inspires us to 2-approximate the case where just the bad flows are known to be from the root to leaves and to

provide fixed parameter tractable algorithms for the case of just the good flows being from the root to leaves and for the general case of a tree network.

These results suggest removing all the edges that do not pass through a good flow, being free, and then checking for every connected component of the resulting graph whether it is a tree. If yes, we can employ the designed algorithm for trees. Furthermore, if a root can be chosen such that every flow in such a tree flows on the path from that root to a leaf, then the suggested `DP` solves the problem exactly. In case the number of the bad and the good flows are small, we can also employ the suggested fixed parameter algorithms. We can also postprocess and delete only the edges that uniquely remove a bad flow.

We have a continuous ranking of the bad flows by weight, but the distinction between the bad and the good is binary. In the future, exploring other rankings would allow modeling other domains of congestion problems. Another possibility one can model is rerouting the disconnected flows, when the tree contains cycles. We would then need to consider the edge capacities, which were not needed so far. Another challenge is also avoiding disconnecting the network or at least minimizing the number of the connected components in the resulting network.

To conclude, we have modeled and approximated two important NP-complete problems at various topology-dependent complexity levels, providing the basis for future research.

Acknowledgments. This research is funded by the Dutch Science Foundation project SARNET (grant no: CYBSEC.14.003/618.001.016)

References

1. Bafna, V., Berman, P., Fujito, T.: A 2-approximation algorithm for the undirected feedback vertex set problem. SIAM J. Discret. Math. **12**(3), 289–297 (1999)
2. Bondy, J., Murty, U.: Graph Theory with Applications. Elsevier Science Publishing Co., Inc., New York (1976)
3. Carr, R.D., Doddi, S., Konjevod, G., Marathe, M.: On the red-blue set cover problem. In: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2000, pp. 345–353. Society for Industrial and Applied Mathematics, Philadelphia (2000)
4. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009)
5. Cygan, M., et al.: Parameterized Algorithms, 1st edn. Springer Publishing Company, Incorporated, Cham (2015). https://doi.org/10.1007/978-3-319-21275-3
6. Downey, R.G., Fellows, M.R.: Fixed parameter tractability and completeness I: basic results. SIAM J. Comput. **24**(4), 873–921 (1995). https://doi.org/10.1137/S0097539792228228
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
8. Garg, N., Vazirani, V.V., Yannakakis, M.: Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica **18**(1), 3–20 (1997)
9. Iwata, S., Nagano, K.: Submodular function minimization under covering constraints. In: 2009 50th Annual IEEE Symposium on Foundations of Computer Science, pp. 671–680, October 2009

10. Khuller, S., Thurimella, R.: Approximation algorithms for graph augmentation. J. Algorithms **14**(2), 214–225 (1993)
11. Koning, R., de Graaff, B., de Laat, C., Meijer, R., Grosso, P.: Interactive analysis of SDN-driven defence against distributed denial of service attacks. In: 2016 IEEE NetSoft Conference and Workshops (NetSoft), pp. 483–488, June 2016
12. Miettinen, P.: On the positive-negative partial set cover problem. Inf. Process. Lett. **108**(4), 219–221 (2008)
13. Mirkovic, J., Dietrich, S., Dittrich, D., Reiher, P.: Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security). Prentice Hall PTR, Upper Saddle River (2004)
14. Mirkovic, J., Reiher, P.: A taxonomy of DDoS attack and DDoS defense mechanisms. SIGCOMM Comput. Commun. Rev. **34**(2), 39–53 (2004)
15. Peleg, D.: Approximation algorithms for the label-covermax and red-blue set cover problems. J. Discret. Algorithms **5**(1), 55–64 (2007)
16. Vazirani, V.: Approximation Algorithms. Springer, Heidelberg (2001). https://doi.org/10.1007/978-3-662-04565-7
17. Watanabe, T., Ae, T., Nakamura, A.: On the removal of forbidden graphs by edge-deletion or by edge-contraction. Discret. Appl. Math. **3**(2), 151–153 (1981)
18. Yannakakis, M.: Node-and edge-deletion NP-complete problems. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC 1978, pp. 253–264. ACM, New York (1978)
19. Zargar, S.T., Joshi, J., Tipper, D.: A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. IEEE Commun. Surv. Tutor. **15**(4), 2046–2069 (2013, Fourth)