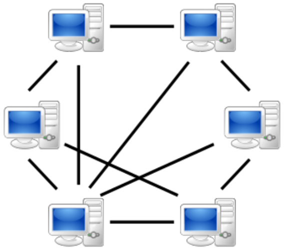
Removing Undesirable Flows by Edge Deletion

Gleb Polevoy Stojan Trajanovski Paola Grosso Cees de Laat

SNE, The University of Amsterdam, The Netherlands



Problems

Consider problems like

- DDoS
- Malicious communication

Needs

While solving by deleting edges, we need to

- Minimize the damage
- Reasonable time

Deleting Edges

Problems of removing "bad" flows by deleting edges on their paths, while trying not to remove too many "good" flows

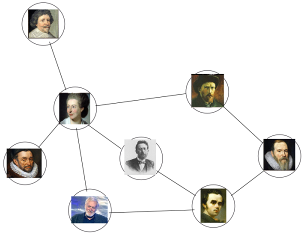
Similar network design problems exist, but not these ones

We

1. formally model
2. hardness and approximation
3. for trees:
   1. Still hard
   2. Two better approximations
   3. FPT
   4. If every flow is on a path from the root to a leaf, exact solution

Model

1. The network is a directed graph $G = (N, E)$.
2. A flow f from node o to d along a path, $f = (\underbrace{P(f)}_{\text{path}})$.
3. Capacities and flow values are irrelevant here.



Model – BFR and BBFR

Definition (Bad Flow Removing (BFR))

1. *Input:* $(G = (N, E), F, GF, BF, w \colon GF \to \mathbb{R}_+)$.
2. *A solution S is a subset of edges to delete.*
3. *A feasible solution is a solution removing all the bad flows.*
4. *Find a feasible solution with the minimum total weight of the removed good flows.*

Definition (Balanced Bad Flow Removing (BBFR))

1. *Input:* $(G = (N, E), F, GF, BF, w \colon F \to \mathbb{R}_+)$.
2. *A solution S is a subset of edges to delete.*
3. *Any solution is feasible.*
4. *Find a feasible solution with the minimum total weight of the remaining bad flows plus the weight of the removed good flows.*

BFR and BBFR are Hard

Hardness of approximation

It is impossible to approximate BFR or BBFR within $O(2^{\log^{1-\delta}|E|})$, for any $\delta > 0$.
For BBFR, it is also impossible within $O(2^{\log^{1-\delta}|BF|})$, for any $\delta > 0$.

BFR can be approximated withing $2\sqrt{|E|\log|BF|}$.
BBFR is approximable within $2\sqrt{(|E|+|BF|)\log(|BF|)}$.

Greedy Approximation

The GreedyBFR Algorithm

1. Given a BFR instance, define the following weighted set cover:
    1. the elements are the bad flows with all edges intersecting good flows;
    2. the sets are the good flows, a good flow covering all the bad flows it intersects.
2. Approximate this set cover instance, obtaining $S \subseteq GF$.
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.

Greedy Approximation - Approximation Ratio

Definition

Let $k \triangleq \max_{g \in GF} |\{g' \in GF : P(g') \cap P(g) \neq \emptyset\}|$.

Proposition

The approximation ratio is $k(\ln(|BF|) + 1)$. For trees, this is even $2k$.

Remark

The same ratios hold for BBFR because of a reduction.

Our Algorithms on Trees – Hardness and Approximation

Both problems are still MAX SNP-hard (inapproximable within 1.166).

Definition

Let l be the maximum length of a good flow, i.e. $\max_{g \in GF} |P(g)|$.

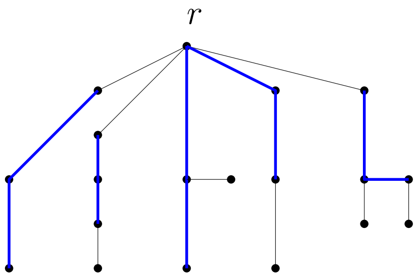
We provide a polynomial $2l$-approximation using a reduction to set cover and this allows for a polynomial $2\sqrt{2|E|}$-approximation for BFR.
This implies a $2l$- and a $2\sqrt{2(|E| + |BF|)}$-approximation for BBFR.

Moreover, there exists an FPT.
If there exists a root $r \in N$ such that every flow flows on a path from r to a leaf, then a `DP` solves it exactly.

These variate results suggest a differentiating approach

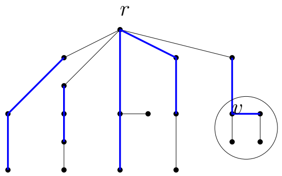
DP if Every Flow Flows on a Path from r to a Leaf

Assume there exists a root $r \in N$ such that every flow flows on a path from r to a leaf (between clients and the ISP and the backbone):



The subprogram is a subtree and the flows that strictly flow through its root after a possible set of outer deletions

DP if Every Flow Flows on a Path from r to a Leaf - 2.

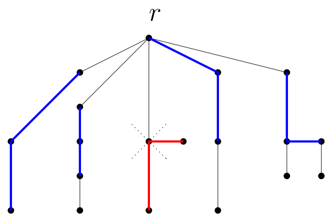


Let $\mathcal{F}(v)$ contain all the possible flows through v after a deletion above v

The DP Algorithm

1. The algorithm maintains the DP-table indexed by $N \times \mathcal{F}(v)$
2. **For each** node $v \in N \setminus \{r\}$ in a **post-order** traversal:
    1. **For each** $S \in \mathcal{F}(v)$:
        1. Delete the edge from v to its parent $\iff$ it maximizes the total objective function in v's subtree
        2. Memoize the resulting edge deletion and the resulting objective function for the current entry $(v, S) \in N \times \mathcal{F}(v)$
3. The completed DP-table contains an optimal set of edge deletions.

Only Bad/Good Flows Flow on a Path from r to a Leaf



Bad flows from r to the leaves - Approximation

The algorithm above implies a 2-approximation when all the bad flows are from a root to the leaves.

Good flows from r to the leaves - FPT

The algorithm above implies an FPT in $O(2^{|BF|})$ times a polynomial time when all the good flows are from a root to the leaves.

FPT for Trees

The DP Algorithm

1. Arbitrarily pick a node to be the root. Call it r.
2. Split each $b \in BF$ to 2 parts that flow from r to a leaf each.
3. Delete one part, and if this is a BBFR instance, define the weight of the remaining part to be $w(b)$. 2 options for each $b \in BF$ that does not flow from r to a leaf. For each set of options, do:
    1. For each $g \in GF$ that has a path not from r to a leaf, split it to 2 parts that do, and assign each part $w(g)$.
    2. Solve the obtained instance using the DP algorithm with the following adjustment. If the DP 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). 2 attempts per each split good flow that enters the subtree.
4. Return the best solution from all the solutions in the above trials.

Conclusions

1. Modeling undesired flow problems (e.g., DDoS, malicious communication)
2. Important, but extremely hard to approximate
3. Greedy approximation
4. Good approximations for trees
5. Optimal DP for trees with all the flows from the root to a leaf
6. FPT in the number of the good and the bad flows for trees
7. $\Rightarrow$ A gradual approach:
    - removing all the edges that don't pass though good flows
    - using the best possible algorithm for each connected component

Future Work

- Other rankings
- Rerouting
- Minimizing the number of connected components in the result

Thank You!