The Unfortunate-Flow Problem

Orna Kupferman
School of Computer Science and Engineering, The Hebrew University, Israel

Gal Vardi
School of Computer Science and Engineering, The Hebrew University, Israel

Abstract

In the traditional maximum-flow problem, the goal is to transfer maximum flow in a network by directing, in each vertex in the network, incoming flow into outgoing edges. The problem is one of the most fundamental problems in TCS, with application in numerous domains. The fact a maximal-flow algorithm directs the flow in all the vertices of the network corresponds to a setting in which the authority has control in all vertices. Many applications in which the maximal-flow problem is applied involve an adversarial setting, where the authority does not have such a control.

We introduce and study the *unfortunate flow problem*, which studies the flow that is guaranteed to reach the target when the edges that leave the source are saturated, yet the most unfortunate decisions are taken in the vertices. When the incoming flow to a vertex is greater than the outgoing capacity, flow is lost. The problem models evacuation scenarios where traffic is stuck due to jams in junctions and communication networks where packets are dropped in overloaded routers.

We study the theoretical properties of unfortunate flows, show that the unfortunate-flow problem is co-NP-complete and point to polynomial fragments. We introduce and study interesting variants of the problem: *integral unfortunate flow*, where the flow along edges must be integral, *controlled unfortunate flow*, where the edges from the source need not be saturated and may be controlled, and *no-loss controlled unfortunate flow*, where the controlled flow must not be lost.

2012 ACM Subject Classification Mathematics of computing → Graph theory → Network flows

Keywords and phrases Flow Network, Graph Algorithms, Games

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.125

1 Introduction

A *flow network* is a directed graph in which each edge has a capacity, bounding the amount of flow that can travel through it. The amount of flow that enters a vertex equals the amount of flow that leaves it, unless the vertex is a *source*, which has only outgoing flow, or a *target*, which has only incoming flow. The fundamental *maximum-flow problem* gets as input a flow network and searches for a maximal flow from the source to the target [4, 10]. The problem was first formulated and solved in the 1950's [8, 9]. It has attracted much research on improved algorithms, variants, and applications [6, 5, 11, 15].

The maximum-flow problem can be applied in many settings in which something travels along a network. This covers numerous application domains, including traffic in road or rail systems, fluids in pipes, packets in a communication network, and many more [1]. Less obvious applications involve flow networks that are constructed in order to model settings with an abstract network, as in the case of scheduling with constraints [1]. In addition, several classical graph-theory problems can be reduced to the maximum-flow problem. This includes the problem of finding a maximum bipartite matching, minimum path cover, maximum edge-disjoint or vertex-disjoint path, and many more [4, 1]. Variants of the maximum-flow problem can accommodate further settings, like circulation problems [18], multiple source and target vertices, costs for unit flows, multiple commodities, and more [7].

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Don Sannella; Article No. 125; pp. 125:1–125:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Studies of flow networks so far assume that the vertices in the network are controlled by a central authority. Indeed, maximum-flow algorithms directs the flow in all vertices of the network. In many applications of flow networks, however, vertices of the network may not be controlled. Thus, every vertex may make autonomous and independent decisions regarding how to direct incoming flow to outgoing edges.

Consider, for example, a road network of a city, where the source s models the center of the city and the target t models the area outside the city. In order to *evacuate* the center of the city, drivers navigate from s to t. In each vertex, every incoming driver chooses an arbitrary outgoing edge with free capacity. If the outgoing capacity from a vertex is less than the incoming flow, then a traffic jam occurs, and flow is lost. As another example, consider a communication network in which packets are sent from a source router s and should reach a target router t. Whenever an internal router receives a packet it forwards it to an arbitrary neighbor router. If the outgoing capacity from a vertex is less than the incoming flow, then packets are dropped, and flow is lost.

In both examples, we want to find the flow that is guaranteed to reach the target in the worst scenario. We now formalize this intuition. Let $\mathcal{G} = \langle V, E, c, s, t\rangle$ be a flow network, where $\langle V, E\rangle$ is a directed graph, $c : E \to \mathbb{N}$ assigns a capacity for each edge, and s, t are the source and target vertices. A *preflow* is a function $f : E \to \mathbb{R}$ that assigns to each edge $e \in E$, a flow in $[0, c(e)]$ such that the incoming flow to each vertex is greater or equal to its outgoing flow. A *saturating preflow* is a preflow in which all outgoing edges from s are saturated, and for every vertex $v \in V \setminus \{s, t\}$, the outgoing flow from v is the minimum between the incoming flow to v and the outgoing capacity from v. That is, in a saturating preflow, flow loss occurs in a vertex v if and only if the incoming flow to v is greater than the capacity of the edges outgoing from v. The *unfortunate flow value* of $\mathcal{G}$ is the minimal flow that reaches t in a saturating preflow. Thus, it is the flow that is guaranteed to reach t when the edges that leave s are saturated, yet the most unfortunate routing decisions are taken in junctions. In the *unfortunate-flow problem*, we want to find the unfortunate flow value of $\mathcal{G}$.

▶ **Example 1.** Consider the flow network $\mathcal{G}$ appearing in Figure 1 (a). A maximum flow in $\mathcal{G}$ has value 8, attained, for example, with the preflow in (b). A saturating preflow in $\mathcal{G}$ appears in (c), and has value 5. While the edges leaving s are saturated, the routing of 7 flow units to the vertex at the bottom leads to a loss of 4 flow units in this vertex. ◀

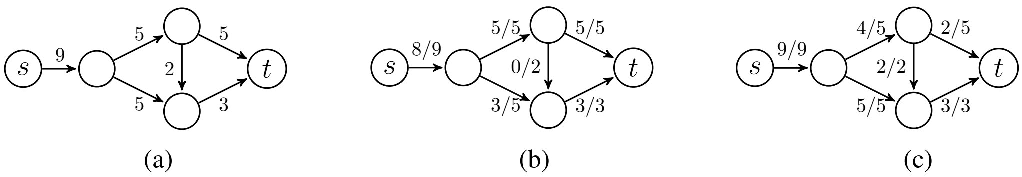

Figure 1 A flow network $\mathcal{G}$, and preflows that attain its maximum-flow and unfortunate-flow values.

We introduce the unfortunate-flow problem, study the theoretical properties of saturating preflows, and study the complexity of the problem. We also introduce and study interesting variants of the problem: *integral unfortunate flow*, where the flow along edges must be integral, *controlled unfortunate flow*, where the edges from the source need not be saturated and may be controlled, and *no-loss controlled unfortunate flow*, where the controlled flow must avoid loss.

Before we describe our contribution, let us review *flow games* and their connection to our contribution here. In flow games [14], the vertices of a flow network are partitioned between two players. Each player controls how incoming flow is partitioned among edges outgoing from her vertices. Then, one player aims at maximizing the flow that reaches t and the other player aims at minimizing it. It is shown in [14] that when the players are restricted to *integral strategies*, thus when the flows along the edges are integers, then the problem of finding the maximal flow that the maximizer player can guarantee is Σ_2^P-complete. The restriction to integral strategies is crucial. Indeed, unlike the case of the traditional maximum-flow problem, non-integral strategies may be better than integral ones. In fact, the problem of finding a maximal flow for the maximizer in a setting with non-integral strategies was left open in [14]. The unfortunate-flow problem can be viewed as a special case of flow games, in which the maximizer player controls no vertex.

We start with the complexity of the unfortunate-flow problem. We consider the decision-problem variant, where we are given a threshold $\gamma > 0$ and decide whether the unfortunate flow value is at least γ. In the case of maximal flow, the problem can be solved in polynomial time [9], and so are many variants of it. We first show that, quite surprisingly, the unfortunate-flow problem is co-NP-hard and that it is NP-hard to approximate within any multiplicative factor. We then point to a polynomial fragment. Intuitively, the fragment restricts the number of vertices in which flow may be lost, which we pinpoint as the computational bottleneck. Formally, we say that a vertex is a *funnel* if its incoming capacity is greater than its outgoing capacity. We show that the unfortunate-flow problem can be solved in time $O(2^{|H|} \cdot (|E|^2 log|V| + |E||V|log^2|V|))$, where $H \subseteq V$ is the set of funnels in $\mathcal{G}$. In particular, the problem can be solved in strongly-polynomial time if the network has a logarithmic number of funnels. Our solution reduces the problem to a sequence of *min-cost max-flow* problems [1], implying the desirable *integral flow property*: the unfortunate-flow value can always be attained by an integral flow. The integral flow property implies a matching co-NP upper bound, thus the unfortunate-flow problem is co-NP-complete.

In some scenarios, we have some initial control on the flow. For example, in evacuation scenarios, as in the example of traffic leaving the city, police may direct cars at the center of the city, but has no control on them once they leave the center. Likewise, when entering or evacuating stadiums, police may direct the crowd to different gates, but has no control on how people proceed once they pass the gates [13]. We study the *controlled unfortunate-flow problem*, where the outgoing flow from s is bounded and controlled. Formally, there is an integer $\alpha \geq 0$ such that the total outgoing flow from s is bounded by α, and it is possible to control how this outgoing flow is partitioned among the edges that leave s. Our goal is to control this flow so that the flow that reaches t in the most unfortunate case is maximized. [1] We show that the integral-flow property no longer holds in this setting. Thus, there are networks in which an optimal strategy is to partition the α units of flow that leave s into non-integers. A troublesome implication of this is that an algorithm that guesses the strategy has to go over unboundedly many possibilities. This challenge is what has left flow games undecidable [14]. We show that we can still reduce the controlled unfortunate-flow problem into the second alternation level of the theory of real numbers under addition and order [17]. The reduction implies membership in Σ_2^P, and we show a matching lower bound. Thus, the controlled unfortunate-flow problem is Σ_2^P-complete. We also study a generalization of the problem, where control can be placed in a subset of the vertices.[2]

Finally, in some scenarios it is crucial for flow not to get lost. For example, in evacuation scenarios, we may prefer to give up an evacuation attempt under a loss risk, and in communication networks, we may tolerate low traffic and not tolerate dropping of packets. We say that a flow network $\mathcal{G}$ is *safe* if all saturating preflows have no loss. For example, networks with no funnels are clearly safe. It is easy to see that $\mathcal{G}$ is safe if its unfortunate flow value is equal to the maximal flow the source can generate, thus to the capacity of the edges outgoing from the source. This gives a co-NP algorithm for deciding the safety of a network. We show one can do better and reduce the

[1] We note that this is different from work done in *evacuation planning*, where the goal is to find routes and schedules of evacuees (for a survey, see [16]).

[2] Not to confuse with the problem of finding critical nodes for firefighters [2, 3]. While there the firefighters block the fire, in our setting they direct the evacuation. Thus, there, the goal is to block undesired vulnerabilities in the network, and here the goal is maximize desired traffic.

safety problem to a maximum weighted flow problem, which can be solved in polynomial time. We then turn to study the *no-loss controlled unfortunate-flow problem*, where we control the flow in edges from s, and we want to maximize the flow to t but in a way that flow loss is impossible. We show that the problem is NP-complete.

Due to space limitations, some examples and proofs are omitted and can be found in the full version, in the authors' URLs.

2 Preliminaries

A *flow network* is $\mathcal{G} = \langle V, E, c, s, t \rangle$, where V is a set of vertices, $E \subseteq V \times V$ is a set of directed edges, $c : E \to \mathbb{N}$ is a capacity function, and $s, t \in V$ are source and target vertices. The capacity function assigns to each edge $e \in E$ a nonnegative capacity $c(e) \geq 0$. We define the *size* of $\mathcal{G}$, denoted $|\mathcal{G}|$ by $|V| + |E| + |c|$, where $|c|$ is the size required for encoding the capacity function c, thus assuming the capacities are given in binary. For a vertex $v \in V$, let $E^{\to v}$ and $E^{v\to}$ be the sets of incoming and outgoing edges to and from v, respectively. That is, $E^{\to v} = (V \times \{v\}) \cap E$ and $E^{v\to} = (\{v\} \times V) \cap E$. A *sink* is a vertex v with no outgoing edges, thus $E^{v\to} = \emptyset$. We assume that t is a sink, it is reachable from s, and $E^{\to s} = \emptyset$. We also assume that $\langle V, E \rangle$ does not contain parallel edges and self loops. For a vertex $v \in V$, let $c(\to v) = \sum_{e \in E^{\to v}} c(e)$ and $c(v\to) = \sum_{e \in E^{v\to}} c(e)$ be the sums of capacities of edges that enter and leave v, respectively. We say that a vertex $v \in V$ is a *funnel* if $c(v\to) < c(\to v)$. We use C_s to denote the total capacity of edges outgoing from the source, thus $C_s = c(s\to)$.

A *preflow* in $\mathcal{G}$ is a function $f : E \to \mathbb{R}$ that satisfies the following two properties:

- For every $e \in E$, we have that $0 \leq f(e) \leq c(e)$.
- For every vertex $v \in V \setminus \{s\}$, the incoming flow to v is greater or equal to its outgoing flow. Formally, $\sum_{e \in E^{\to v}} f(e) \geq \sum_{e \in E^{v\to}} f(e)$.

For a preflow f and an edge $e \in E$, we say that e is *saturated* if $f(e) = c(e)$. We extend f to vertices: for every vertex $v \in V$, let $f(\to v) = \sum_{e \in E^{\to v}} f(e)$ and $f(v\to) = \sum_{e \in E^{v\to}} f(e)$. For a vertex $v \in V \setminus \{s, t\}$, the *flow loss of f in v*, denoted $l_f(v)$, is the quantity that enters v and does not leave v. Formally, $l_f(v) = f(\to v) - f(v\to)$. Then, $L_f = \sum_{v \in V \setminus \{s,t\}} l_f(v)$ is the *flow loss of f*. The value of a preflow f, denoted $val(f)$, is $f(\to t)$; that is, the incoming flow to t. Note that $val(f) = f(s\to) - L_f$. A *flow* is a preflow f with $L_f = 0$. A *maximum flow* is a flow with a maximal value.

A *saturating preflow* is a preflow in which all edge in $E^{s\to}$ are saturated, and for every $v \in V \setminus \{s, t\}$, we have $f(v\to) = \min\{f(\to v), c(v\to)\}$. That is, in a saturating preflow, flow loss may occur in a vertex v only if the incoming flow to v is bigger than the capacities of the edges outgoing from v.

The *unfortunate value* of a flow network $\mathcal{G}$, denoted $uval(\mathcal{G})$, is the minimal value of a saturating preflow in $\mathcal{G}$. That is, it is the value that is guaranteed to reach t when the edges that leave s are saturated, yet the most unfortunate routing decisions are taken in junctions. An *unfortunate saturating preflow* is a saturating preflow that attains the network's unfortunate value. The *unfortunate flow* problem (UF problem, in short) is to decide, given a flow network $\mathcal{G}$ and a threshold $\gamma > 0$, whether $uval(\mathcal{G}) \geq \gamma$.

3 The Complexity of the Unfortunate-Flow Problem

In this section we study the complexity of the unfortunate-flow problem. We start with bad news and show that the problem is co-NP-hard, and in fact is NP-hard to approximate within any multiplicative factor. A more precise analysis of the complexity then enables us to point to a polynomial fragment and to prove an integral-flow property, which implies a matching co-NP upper bound.

▶ **Theorem 2.** *The UF problem is co-NP-hard.*

Proof. We show a reduction from CNF-SAT to the complement problem, namely deciding whether $uval(\mathcal{G}) < \gamma$ for some $\gamma \in \mathbb{N}$. Let $\psi = C_1 \wedge \ldots \wedge C_m$ be a CNF formula over the variables $x_1 \ldots x_n$. We assume that every literal in $x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n$ appears in exactly k clauses in ψ. Indeed, every CNF formula can be converted to such a formula in polynomial time and with a polynomial blowup. We construct a flow network $\mathcal{G} = \langle V, E, c, s, t \rangle$ as demonstrated in Figure 2. Let $Z = \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$. For a literal $z \in Z$ and a clause C_i, the network $\mathcal{G}$ contains an edge $\langle z, C_i \rangle$ iff C_i contains z. Thus, each vertex in Z has exactly k outgoing edges. The capacities of these edges are 1. Each vertex C_i has two outgoing edges – to t and to the sink u. In Appendix A.1, we prove that ψ is satisfiable iff $uval(\mathcal{G}) < kn - m + 1$. ◀

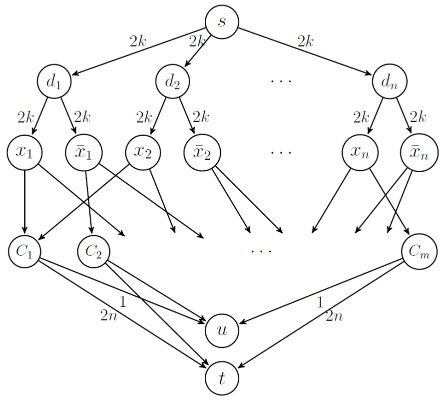

Figure 2: The flow network $\mathcal{G}$. The capacities of the edges entering $C_1, \ldots, C_m$ are 1.

By a simple manipulation of the network $\mathcal{G}$ constructed in the reduction in the proof of Theorem 2, we can obtain, given a CNF-SAT formula ψ, a network $\mathcal{G}'$ such that if ψ is satisfiable, then $uval(\mathcal{G}') = 0$, and otherwise, $uval(\mathcal{G}') \geq 1$. Hence the following (a detailed proof can be found in Appendix A.2).

▶ **Theorem 3.** *It is NP-hard to approximate the UF problem within any multiplicative factor.*

Following the hardness of the problem, we turn to analyze its complexity in terms of the different parameters of the flow network. Our analysis points to a class of networks for which the UF problem can be solved in polynomial time.

Consider a flow network $\mathcal{G} = \langle V, E, c, s, t \rangle$. Let $H \subseteq V \setminus \{s, t\}$ be the set of funnels in $\mathcal{G}$. Thus, $H = \{v : c(\rightarrow v) > c(v \rightarrow)\}$. For $L \subseteq V$, let $\mathcal{F}_L$ be a set of saturating preflows in which edges outgoing from vertices in L are saturated, and flow loss may occur only in vertices in L. Thus, $f \in \mathcal{F}_L$ iff f is a saturating preflow in $\mathcal{G}$ such that for every $u \in L$, we have $f(u\rightarrow) = c(u\rightarrow)$, and for every $u \in V \setminus L$, we have $l_f(u) = 0$. By the definition of a saturating flow, flow loss in $\mathcal{G}$ may occur only in vertices in H. Accordingly, we have the following.

▶ **Lemma 4.** $\bigcup_{L \subseteq H} \mathcal{F}_L$ *contains all the saturating preflows in* $\mathcal{G}$.

By Lemma 4, a search for the unfortunate value of $\mathcal{G}$ can be restricted to preflows in $\mathcal{F}_L$, for $L \subseteq H$. Accordingly, the UF problem can be solved by solving $2^{|H|}$ optimization problems, solvable by either linear programming (Theorem 5) or a reduction to the min-cost max-flow problem (Theorem 6).

▶ **Theorem 5.** *Consider a flow network* $\mathcal{G}$ *and let* H *be the set of funnels in* $\mathcal{G}$*. The UF problem for* $\mathcal{G}$ *can be solved in time* $2^{|H|} \cdot poly(|\mathcal{G}|)$.

Proof. The algorithm goes over all the subsets of H and for each subset $L \subseteq H$, finds a minimum-value preflow in $\mathcal{F}_L$. The latter is done by linear programming. Given L, the linear program for $\mathcal{F}_L$

is described below. The variable x_e, for every $e \in E$, stands for $f(e)$. The program is of size linear in $|\mathcal{G}|$, thus the overall complexity is $2^{|H|} \cdot \mathit{poly}(|\mathcal{G}|)$.

$$\begin{array}{lll} \text{minimize} & \sum_{e \in E^{\to t}} x_e & \\ \text{subject to} & 0 \le x_e \le c(e) & \text{for each } e \in E \\ & x_e = c(e) & \text{for each } u \in L \cup \{s\}, e \in E^{u\to} \\ & \sum_{e \in E^{u\to}} x_e \le \sum_{e \in E^{\to u}} x_e & \text{for each } u \in L \\ & \sum_{e \in E^{u\to}} x_e = \sum_{e \in E^{\to u}} x_e & \text{for each } u \notin L \cup \{s,t\} \end{array}$$

◀

The complexity of solving each linear program in the algorithm described in the proof of Theorem 5 is polynomial in $|\mathcal{G}|$, but not *strongly polynomial*. Thus its running time depends (polynomially) on the number of bits required for representing the capacities in $\mathcal{G}$. We now describe an alternative algorithm whose complexity depends only on the number of vertices and edges in the network.

Our algorithm reduces the problem of finding a minimal-value preflow in $\mathcal{F}_L$ to the *min-cost max-flow* problem in flow networks with costs [1]. A flow network with costs is $\mathcal{G} = \langle V, E, a, c, s, t\rangle$, where $\langle V, E, c, s, t\rangle$ is a flow network and $a : E \to \mathbb{R}$ is a cost function. The cost of a flow f in $\mathcal{G}$, denoted $\mathit{cost}(f)$, is $\sum_{e \in E} a(e) \cdot f(e)$. In the min-cost max-flow problem we are given a flow network with costs, and find a maximum flow with a minimum cost. By [1], this problem can be solved in time $O(|E|^2 log|V| + |E||V|log^2|V|)$.

▶ **Theorem 6.** *Consider a flow network $\mathcal{G} = \langle V, E, c, s, t\rangle$ and let H be the set of funnels in $\mathcal{G}$. The UF problem for $\mathcal{G}$ can be solved in time $O(2^{|H|} \cdot (|E|^2 log|V| + |E||V|log^2|V|))$.*

Proof. The algorithm finds, for each subset $L \subseteq H$, a minimum-value preflow in $\mathcal{F}_L$ by a reduction to the min-cost max-flow problem. By Lemma 4, the minimum value found for some $L \subseteq H$ is $\mathit{uval}(\mathcal{G})$.

Consider the flow network with costs $\mathcal{G}' = \langle V', E', a, c', s, t'\rangle$ that is obtained from $\mathcal{G}$ as follows. We add a new vertex t' and edges $\langle u, t'\rangle$ for every $u \in L \cup \{t\}$, thus $V' = V \cup \{t'\}$ and $E' = E \cup (L \cup \{t\}) \times \{t'\}$. The capacity $c'(e)$ for every new edge $e \in E' \setminus E$ is large (for example, it may be C_s), and for every $e \in E$, we have $c'(e) = c(e)$. Let $C = \max\{c(e) : e \in E\}$ denote the maximal capacity in $\mathcal{G}$. For edges $e \in L \times \{t'\}$, we define $a(e) = -1$; for edges $e \in L \times V$, we define $a(e) = -C \cdot |V|^2$; and for all the other edges, we define $a(e) = 0$. Intuitively, the costs of the edges in $L \times V$ are negative and small enough, so that a min-cost max-flow in $\mathcal{G}'$ would have to saturate them first, and only then try to direct flow to edges in $L \times \{t'\}$.

In Appendix A.3, we prove the correctness of the following algorithm: First, find a min-cost max-flow f' in $\mathcal{G}'$. If $\mathit{val}(f') < C_s$ or $\mathit{cost}(f') > -C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e)$, then $\mathcal{F}_L = \emptyset$. Otherwise, the minimal value of a preflow in $\mathcal{F}_L$ is $C_s + \mathit{cost}(f') + C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e)$. Since the min-cost max-flow problem can be solved in time $O(|E|^2 log|V| + |E||V|log^2|V|)$ [1] and there are $2^{|H|}$ subsets of funnels to check, the required complexity follows. ◀

▶ **Corollary 7.** *The UF problems for networks with a logarithmic number of funnels can be solved in strongly-polynomial time.*

We say that a preflow $f : E \to \mathbb{R}$ is *integral* if $f(e) \in \mathbb{N}$ for all $e \in E$. It is sometimes desirable to restrict the flow to an integral one, for example in settings in which the objects we transfer along the network cannot be partitioned into fractions. We now show that the UF problem always has an integral-flow solution, and that such a solution can be obtained by the algorithm shown in the proof of Theorem 6. Essentially (see proof in Appendix A.4), it follows from the fact that the min-cost max-flow problem has an integral solution. As we show in Section 4, this *integral flow property* is not maintained in variants of the UF problem.

▶ **Theorem 8.** *The UF problem has an integral-flow solution: for every flow network, there exists an integral unfortunate saturating preflow. Moreover, such integral preflow can be found by the algorithm described in the proof of Theorem 6.*

The integral-flow property suggests an optimal algorithm for solving the UF problem:

▶ **Theorem 9.** *The UF problem is co-NP-complete.*

Proof. Hardness in co-NP is proven in Theorem 2. We prove membership in NP for the complementary problem: given $\gamma > 0$ and a flow network $\mathcal{G}$, we need to decide whether $uval(\mathcal{G}) < \gamma$. According to Theorem 8, it is enough to decide whether there is a saturating preflow f in which for every $e \in E$, the value $f(e)$ is an integer, and $val(f) < \gamma$. Given a function $f : E \to \mathbb{N}$, checking whether f satisfies these requirements can be done in polynomial time, implying membership in NP. ◀

4 The Controlled Unfortunate-Flow Problem

In this section we study the controlled unfortunate-flow problem, where the outgoing flow from s is bounded and controlled. That is, there is $0 \leq \alpha \leq C_s$ such that the total outgoing flow from s is bounded by α, and it is possible to control how this outgoing flow is partitioned among the edges that leave s. Our goal is to control this flow so that the flow that reaches t in the worst case is maximized. As discussed in Section 1, this problem is motivated by scenarios where we have an initial control on the flow, say by positioning police at the entrance to a stadium or at the center of a city we need to evacuate, or by transmitting messages we want to send from a router we own.

For $\alpha \geq 0$, a *regulator with bound* α is a function $g : E^{s\to} \to \mathbb{R}$ that directs α flow units from s. Formally, for every $e \in E^{s\to}$, we have $0 \leq g(e) \leq c(e)$, and $\sum_{e \in E^{s\to}} g(e) \leq \alpha$. A *controlled saturating preflow that respects a regulator* g is a preflow $f : E \to \mathbb{R}$ such that for every $e \in E^{s\to}$, we have $f(e) = g(e)$, and for every $v \in V \setminus \{s, t\}$, we have $f(v\to) = \min\{f(\to v), c(v\to)\}$. Thus, unlike saturating preflow, here the edges in $E^{s\to}$ need not be saturated and the flow in them is induced by g. The *unfortunate* g*-controlled value* of $\mathcal{G}$, denoted $cuval(\mathcal{G}, g)$, is the minimal value of a controlled saturating preflow that respects g. Then, the *unfortunate* α*-controlled value* of $\mathcal{G}$, denoted $cuval(\mathcal{G}, \alpha)$ is the maximal unfortunate g-controlled value of $\mathcal{G}$ for some regulator g with bound α. In the *controlled unfortunate flow* problem (CUF problem, for short), we are given a flow network $\mathcal{G}$, a bound $\alpha \geq 0$, and a threshold $\gamma > 0$, and we need to decide whether $cuval(\mathcal{G}, \alpha) \geq \gamma$. Thus, in the CUF problem we need to decide whether there is a regulator g with bound α that ensures a value of at least γ.

For two regulators g and g', we denote $g \geq g'$ if for every $e \in E^{s\to}$, we have $g(e) \geq g'(e)$. In the following theorem we show that the g-controlled unfortunate value is monotonic with respect to g. Thus, increasing g can only increase the value. In particular, it follows that a maximal $cuval(\mathcal{G}, \alpha)$ is obtained with $\alpha = C_s$ and a regulator g in which $g(e) = c(e)$ for every $e \in E^{s\to}$. Thus, if the outgoing flow from s is not bounded, then the optimal behavior is to saturate the edges in $E^{s\to}$. Essentially (see full proof in Appendix A.5), it follows from the fact that given g and g' such that $g \geq g'$, and a minimum-value controlled saturating preflow f that respects g, we can construct a controlled saturating preflow f' that respects g' and such that $val(f) \geq val(f')$.

▶ **Theorem 10.** *Consider a flow network $\mathcal{G}$, and let g, g' be two regulators such that $g \geq g'$. Then, $cuval(\mathcal{G}, g) \geq cuval(\mathcal{G}, g')$.*

We now turn to study the complexity of the CUF problem. We first explain why the problem is challenging. One could expect an algorithm in which, given $\mathcal{G}$, α, and γ, we guess an *integral regulator* $g : E^{s\to} \to \mathbb{N}$ with bound α, and then use an NP oracle in order to check whether $cuval(\mathcal{G}, g) \geq \gamma$. The problem with the above idea is that it restricts the regulators to integral ones. In Theorem 11 below we show that in some cases, an optimal regulator must use non-integral values. Accordingly, an algorithm that guesses a regulator, as has been the case with the guessed flows in Theorem 9, has to go over unboundedly many possibilities. In fact, when an arbitrary set of vertices (rather than the source only) may be controlled, the problem is not known to be decidable [14].

▶ **Theorem 11.** *Integral regulators are not optimal: There is a flow network $\mathcal{G}$ such that for every integral regulator $g : E^{s\to} \to \mathbb{N}$ with bound 2 we have $cuval(\mathcal{G}, g) = 1$, but there is a regulator $g' : E^{s\to} \to \mathbb{R}$ with bound 2 such that $cuval(\mathcal{G}, g') = 2$.*

Proof. Consider the flow network $\mathcal{G}$ appearing in Figure 3. For every pair $\langle u_i, u_j\rangle$ for $1 \leq i < j \leq 4$, the network $\mathcal{G}$ contains a vertex v_{ij} with incoming edges from u_i and u_j. The capacities of the edges in $\mathcal{G}$ are all 1. It is not hard to see that for every integral regulator g with bound 2 we have $cuval(\mathcal{G}, g) = 1$. Indeed, for such g there is a controlled saturating preflow that respects g, which directs a flow of 2 to some vertex v_{ij}, causing a loss of 1 in v_{ij}. Consider now the regulator g' that assigns a flow of 0.5 to every edge in $E^{s\to}$. In this case, a flow of more than 1 cannot be directed to any vertex v_{ij} and therefore $cuval(\mathcal{G}, g') = 2$. ◀

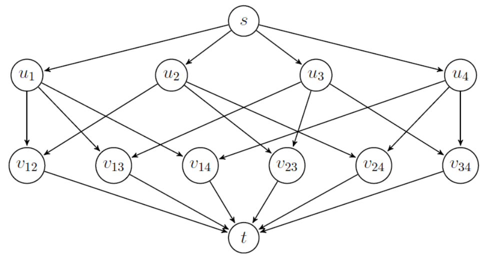

Figure 3: The flow network $\mathcal{G}$. All capacities are 1.

We turn to solve the CUF problem. Theorem 11 forces us to consider non-integral regulators. We do this by a reduction to a problem with a similar challenge, namely *the second alternation level of the theory of real numbers under addition and order*. There, we are given a formula of the form $\exists x_1 \dots x_n \forall y_1 \dots y_m F(x_1, \dots, x_n, y_1, \dots, y_m)$, where F is a propositional combination of linear inequalities of the form $a_1x_1 + \dots a_nx_n + b_1y_1 + \dots + b_my_m \leq d$, for constant integers $a_1, \dots, a_n$, $b_1, \dots, b_m$, and d, and we have to decide whether there is an assignment of $x_1, \dots, x_n$ to real numbers so that F is satisfied for every assignment of $y_1, \dots, y_m$ to real numbers. Even though the domain of possible solutions is infinite, It is shown in [17] that the problem can be solved in Σ_2^P, namely the class of problems that can be solved by a nondeterministic polynomial Turing machine that has an oracle to some NP-complete problem. In [14], a Σ_2^P lower bound is proven for the problem of finding the value of a flow game, where the outgoing flow of a subset of the vertices can be controlled. Recall that in the CUF problem, only the flow from the source vertex can be controlled. While this corresponds to the "exists-forall" nesting of quantifiers that characterizes reasoning in Σ_2^P, it not clear how to reduce Boolean formulas to unfortunate flows. Indeed, in the reduction in [14], control in intermediate vertices is used in order to model disjunctions in the formulas. In the CUF problem, such a control is impossible, as all vertices in the network except for the source are treated in a conjunctive manner.

▶ **Theorem 12.** *The CUF problem is Σ_2^P-complete.*

Proof. We first prove membership in Σ_2^P by a reduction to the second alternation level of the theory of real numbers under addition and order. Given G, α, and γ, we construct a propositional combination F of linear inequalities over the variables x_e for every $e \in E^{s\to}$, and variables y_e, for every $e \in E$. The formula F states that the values of the variables x_e corresponds to a regulator g with bound α, and that if the values of the variables y_e correspond to a controlled saturating preflow f

that respects g then $val(f) \geq \gamma$. Then, our problem amounts to deciding whether there are real values x_e such that for every real values y_e the formula F holds.

For the lower bound, we describe a reduction from QBF$_2$, namely satisfiability for quantified Boolean formulas with one alternation of quantifiers, where the external quantifier is "exists". Let ψ be a propositional formula over the variables $x_1, \ldots, x_n, y_1, \ldots, y_m$, and let $\theta = \exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_m \psi$. Also, let $X = \{x_1, \ldots, x_n\}$, $\bar{X} = \{\bar{x}_1, \ldots, \bar{x}_n\}$, $Y = \{y_1, \ldots, y_m\}$, $\bar{Y} = \{\bar{y}_1, \ldots, \bar{y}_m\}$, $Z = X \cup Y$, and $\bar{Z} = \bar{X} \cup \bar{Y}$. We construct a flow network $\mathcal{G}_\theta$ and define α and γ, such that θ holds iff there is a regulator g with bound α such that $cuval(\mathcal{G}_\theta, g) \geq \gamma$.

We assume that ψ is given in a positive normal form; that is, ψ is constructed from the literals in $Z \cup \bar{Z}$ using the Boolean operators $\vee$ and $\wedge$, and that there is $k \geq 1$ such that every literal in $Z \cup \bar{Z}$ appears in ψ exactly k times. Clearly, every Boolean propositional formula can be converted with only a quadratic blow-up to an equivalent one that satisfies these conditions.

We first translate ψ into a Boolean circuit $\mathcal{C}_\psi$ with $k(2n + 2m)$ inputs – one for each occurrence of a literal in ψ. For example, in Figure 4, on the left, we describe C_ψ for $\psi = x \vee (\bar{x} \wedge y) \wedge ((x \wedge \bar{y}) \vee (y \vee \bar{y} \vee \bar{x}))$. Each gate in $\mathcal{C}_\psi$ has fan-in 2 and fan-out 1. We say that an input assignment to $\mathcal{C}_\psi$ is consistent if it corresponds to an assignment to the variables in Z. That is, for each variable $z \in Z$, there is a value $b \in \{0, 1\}$ such that all the k inputs that correspond to the literal z have value b and all the k inputs that correspond to the literal $\bar{z}$ have value $1 - b$. If the input to $\mathcal{C}_\psi$ is consistent then $\mathcal{C}_\psi$ computes the value of ψ for the corresponding assignment.

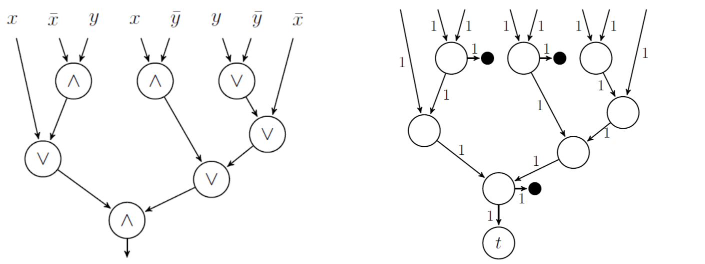

Figure 4: The Boolean circuit $\mathcal{C}_\psi$ and the external-source flow network G_ψ.

Now, we translate $\mathcal{C}_\psi$ to an *external-source flow network* $\mathcal{G}_\psi = \langle V, E, c, t \rangle$: a flow network in which there is no source vertex, and an input flow is given externally. Formally, some of the edges in E have an unspecified source, to be later connected to edges with an unspecified target. The idea behind the translation is as follows: The capacities in $\mathcal{G}_\psi$ are all 1. Each OR gate in $\mathcal{C}_\psi$ induces a vertex v that has in-degree 2 and out-degree 1. Thus, if the incoming flow in each incoming edge to v is 0 or 1, then its outgoing flow is 1 iff at least one of its incoming edges has flow 1. Then, each AND gate in $\mathcal{C}_\psi$ induces a vertex v that has in-degree 2 and out-degree 2, yet, one of the two edges that leaves v leads to a sink. Accordingly, if the incoming flow in each incoming edge to v is 0 or 1, then the outgoing flow in the edge that does not lead to the sink must be 1 iff both incoming edges have flow 1. For example, the Boolean circuit $\mathcal{C}_\psi$ from Figure 4 is translated to the external-source flow network $\mathcal{G}_\psi$ to its right.

Given a flow from the external source, we define the unfortunate value of $\mathcal{G}_\psi$ as the minimal value of a controlled saturating preflow that respects the external flow. The following lemma can be easily proved by induction on the structure of ψ.

▶ **Lemma 13.** *Consider a Boolean formula ψ and its corresponding external-source flow network $\mathcal{G}_\psi$.*

1. *Given input flows to $\mathcal{G}_\psi$, if we increase some input flow, then the new unfortunate value of $\mathcal{G}_\psi$ is greater than or equal to the original unfortunate value.*

2. *Given input flows in $\{0,1\}$ to $\mathcal{G}_\psi$, the unfortunate value of $\mathcal{G}_\psi$ is equal to the output of $\mathcal{C}_\psi$ with the same input. Thus, if the input flow to $\mathcal{G}_\psi$ corresponds to a consistent input to $\mathcal{C}_\psi$, then the unfortunate value of $\mathcal{G}_\psi$ is the value of ψ for the corresponding assignment.*

We complete the reduction by constructing the flow network $\mathcal{G}_\theta$ that uses $\mathcal{G}_\psi$ as a sub-network as shown in Figure 5. The vertices $d_{y_1}, \ldots, d_{y_m}$ are associated with the variables in Y. The vertices $x_i, \bar{x}_i, y_i, \bar{y}_i$ for every i are associated with the literals in $Z \cup \bar{Z}$. Each outgoing edge from a literal vertex that enters $\mathcal{G}_\psi$ is connected to an input of $\mathcal{G}_\psi$ that corresponds to this literal. The outgoing edge from the subnetwork $\mathcal{G}_\psi$ corresponds to an edge from the target vertex of $\mathcal{G}_\psi$. In Appendix A.6 we describe the network $\mathcal{G}_\theta$ for the case $\psi = (x \vee y) \wedge (\bar{x} \vee \bar{y})$.

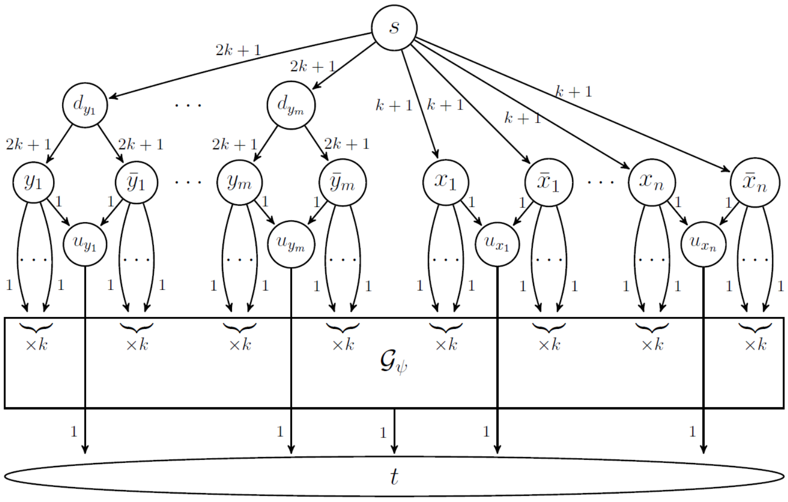

Figure 5: The flow network $\mathcal{G}_\theta$.

In Appendix A.7, we prove that θ holds iff there is a regulator g with bound $(2k+1)m+(k+1)n$ such that for every controlled saturating preflow f that respects g we have $val(f) \geq m+n+1$. ◀

In Theorem 11 we showed that in some cases an optimal regulator must use non-integral values. Sometimes, however, it is desirable to restrict attention to integral regulators. In the following theorem (see proof in Appendix A.8) we show that the Σ_2^P-completeness stays valid also for integral regulators.

▶ **Theorem 14.** *Let $\mathcal{G}$ be a flow network and let α, γ be integral constants. Deciding whether there exists an integral regulator $g : E^{s\rightarrow} \rightarrow \mathbb{N}$ with bound α such that $cuval(\mathcal{G}, g) \geq \gamma$, is Σ_2^P-complete.*

▶ Remark. **[Bounded Global Control]** In the CUF problem, it is possible to control the flow leaving the source. This could be generalized by letting an authority control also internal vertices in the network. In the *bounded global control* problem, we get as input a flow network $\mathcal{G}$, a number $k \geq 0$, and a threshold $\gamma > 0$, and we need to decide whether we can guarantee an unfortunate flow of at least γ by controlling the outgoing flow in at most k vertices. Note that while in the problem of finding critical nodes for firefighters [2, 3], a firefighter blocks the fire, in our setting the firefighters direct the evacuation. Thus, there, the goal is to block undesired vulnerabilities in the network, and here the goal is maximize desired traffic in the network. The formal definition of the bounded global control problem goes through the flow games of [14], which includes the notion of strategies for controlling flow. The Σ_2^P algorithm for solving flow games with integral flows can be extended to solve the bounded global control problem. By making the control on the source vertex essential (say, by adding a transition with a large capacity to a sink), the CUF problem can be reduced to the global control problem with $k = 1$, implying Σ_2^P completeness.

5 Safe Networks and No-Loss Unfortunate Flow

In this section we consider settings in which loss must be avoided. We say that a flow network $\mathcal{G}$ is *safe* if $L_f = 0$ for every saturating preflow f. For example, networks with no funnels are clearly safe. It is easy to see that $\mathcal{G}$ is safe iff $uval(\mathcal{G}) = C_s$. Together with Theorem 9, this gives a co-NP algorithm for deciding the safety of a network. We first show that by reducing the safety problem to the maximum weighted flow problem, we can decide safety in polynomial time. Essentially, the reduction checks, for every vertex $v \in V$, whether it is possible to direct to v flow that is greater than its outgoing capacity, and the weights are used in order to filter flow incoming to v. For details, see Appendix A.9.

▶ **Theorem 15.** *Deciding whether a flow network is safe can be done in polynomial time.*

We now consider the case where the total outgoing flow from s is controlled, and we need to find an optimal regulator that guarantees no flow loss. Formally, in the *no-loss controlled unfortunate-flow problem* (NLCUF problem, for short), we are given a flow network $\mathcal{G}$ and an integer $\gamma > 0$, and we need to decide whether there exists a regulator g such that $\sum_{e \in E^{s\to}} g(e) \geq \gamma$, and for every controlled saturating preflow that respects g the flow loss is 0 (equivalently, $cuval(\mathcal{G}, g) = \gamma$). That is, decide whether there is a regulator that ensures no loss and a value of at least γ. We show that the NLCUF problem is NP-complete. For the upper bound one could expect an algorithm in which we guess an integral regulator $g : E^{s\to} \to \mathbb{N}$ in which the total flow is at least γ, and then use Theorem 15 in order to check in polynomial time whether flow loss is possible. However, Theorem 11 shows that in some cases a regulator must use non-integral values in order to ensure that flow loss is impossible. Consequently, our algorithm is more complicated and uses a result from the theory of real numbers with addition.

▶ **Theorem 16.** *The NLCUF problem is NP-complete.*

Proof: We start with the upper bound. For a rational number q we denote by $\#(q)$ the *length* of q, namely, if $q = a/b$ with a, b relatively prime, then $\#(q)$ is the sum of the number of bits in the binary representations of a and b. Consider a formula $\varphi = \exists x_1, ..., x_n \forall y_1, ..., y_m F(x_1, ..., x_n, y_1, ..., y_m)$, where F is a propositional combination of linear inequalities of the form $a_1x_1 + ... + a_nx_n + b_1y_1 + ... + b_my_m \leq d$ for integral constants $a_1, ..., a_n, b_1, ..., b_m$, and d. The variables $x_1, ..., x_n, y_1, ..., y_m$ are real. In [17] (in the proof of Theorem 3.1 there) it is shown that φ holds iff there exists rational values $x_1, ..., x_n$ such that for every i the length $\#(x_i)$ is polynomial in the size of φ and for every real values $y_1, ..., y_m$ the formula F holds.

We construct a propositional combination F of linear inequalities over the variables x_e, for every $e \in E^{s\to}$, and y_e, for every $e \in E$. The formula F states that the values of the variables x_e correspond to a regulator g with bound γ, and that if the values of the variables y_e correspond to a controlled saturating preflow f that respects g, then $L_f = 0$. Then, our problem amounts to deciding whether there are real values x_e such that for every real values y_e, the formula F holds. By [17], it is enough to check whether there are rational values x_e for $e \in E^{s\to}$ with polynomial lengths such that for every real values y_e for $e \in E$, the formula F holds. Given values for the variables x_e, checking whether for every real values y_e the formula F holds can be done in polynomial time with the algorithm shown in the proof of Theorem 15. Hence the membership in NP.

We proceed to the lower bound. We show a reduction from CNF-SAT. Let $\psi = C_1 \wedge \ldots \wedge C_m$ be a CNF formula over the variables $x_1 \ldots x_n$. We denote $Z = \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$. We assume that for every literal $z \in Z$ there is at least one clause in ψ that does not contain z. We construct a flow network $\mathcal{G} = \langle V, E, c, s, t \rangle$ as demonstrated in Figure 6. For a literal $z \in Z$ and a clause C_i, the network $\mathcal{G}$ contains an edge $\langle z, C_i \rangle$ iff the clause C_i does not contain the literal z. Let $\gamma = 2n$. In Appendix A.10, we show that ψ is satisfiable iff there is a regulator that ensures no loss and a value of at least γ. ◀

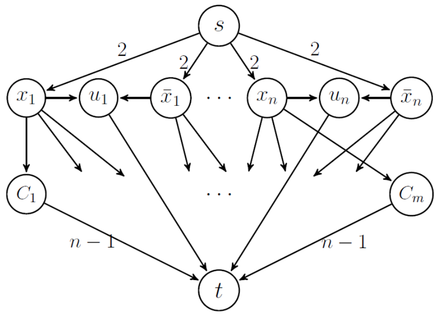

Figure 6: The flow network $\mathcal{G}$. Unless stated otherwise, the capacities are 1.

Sometimes it is desirable to restrict attention to integral regulators. As we show below, NP-completeness applies for them too (see Appendix A.11 for proof).

▶ **Theorem 17.** *Let $\mathcal{G}$ be a flow network and let $\gamma > 0$ be an integer. Deciding whether there exists an integral regulator $g : E^{s\rightarrow} \rightarrow \mathbb{N}$ in $\mathcal{G}$ such that $\sum_{e \in E^{s\rightarrow}} g(e) \geq \gamma$, and for every controlled saturating preflow that respects g the flow loss is 0, is NP-complete.*

6 Discussion

The unfortunate-flow problem captures settings in which the authority has no control on how flow is directed in the vertices of a flow network. For many problems, a transition from a cooperative setting to an adversarial one dualizes the complexity class to which the problem belongs, as in NP for satisfiability vs. co-NP for validity. In the case of flow, the polynomial complexity of the maximum-flow problem is not preserved when we move to the dual unfortunate-flow problem, and we prove that the problem is co-NP-complete.

On the positive side, the integral-flow property of maximal flow is preserved in unfortunate flows. This property, however, is lost once we move to controlled unfortunate flows, where non-integral regulators may be more optimal than integral ones. The need to consider real-valued flows questions the decidability of the controlled unfortunate-flow problem. As we show, the problem is decidable, by a reduction to the second alternation level of the theory of real numbers under addition and order [17]. There, the infinite domain of the real numbers is reduced to a finite one, namely rational numbers of length polynomial in the input. A direct algorithm for the controlled unfortunate-flow problem, thus one that does not rely on [17], is still open. Such a direct algorithm would reduce the real-number domain to a finite one in a tighter manner – one that depends on the network. We see several interesting problems in this direction, in particular finding a *sufficient granularity* that a regulator may need, and *bounding the non-optimality* caused by integral regulators. Similar problems are open in the settings of flow games with two or more players [14, 12].

Finally, the unfortunate-flow problem sets the stage to problems around network design, where the goal is to design networks with maximal unfortunate flows. In particular, in *network repair*, we are given a network and we are asked to modify it in order to increase its unfortunate flow value. Different algorithms correspond to different types of allowed modifications. For example, we may be allowed to change the capacity of a fixed number of edges. Note that unlike the case of maximal flow, here a repair may reduce the capacity of edges. Also, unlike the case of maximal flow, there is no clear theory of minimal cuts that may assist us in such a repair.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: Theory, algorithms, and applications*. Prentice Hall Englewood Cliffs, 1993.
2. C. Bazgan, M. Chopin, M. Cygan, M.R. Fellows, F. V. Fomin, and E. Leeuwen. Parameterized complexity of firefighting. *Journal of Computer and Systems Science*, 80(7):1285–1297, 2014.
3. J. Choudhari, A. Dasgupta, N. Misra, and M.S. Ramanujan. Saving critical nodes with firefighters is FPT. In *Proc. 44th Int. Colloq. on Automata, Languages, and Programming*, volume 80 of *LIPIcs*, pages 135:1–135:13, 2017.
4. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.
5. E.A. Dinic. Algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math. Doll*, 11(5):1277–1280, 1970. English translation by RF. Rinehart.
6. J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.
7. S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.
8. L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.
9. L.R. Ford and D.R. Fulkerson. *Flows in networks*. Princeton Univ. Press, Princeton, 1962.
10. A.V. Goldberg, É. Tardos, and R.E. Tarjan. Network flow algorithms. Technical report, DTIC Document, 1989.
11. A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.
12. S. Guha, O. Kupferman, and G. Vardi. Multi-player flow games. In *Proc. 17th International Conference on Autonomous Agents and Multiagent Systems*, 2018.
13. S. Keren, A. Gal, and E. Karpas. Goal recognition design for non optimal agents. In *Proc. 29th AAAI conference*, pages 3298–3304, 2015.
14. O. Kupferman, G. Vardi, and M.Y. Vardi. Flow games. In *Proc. 37th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:38–38:16, 2017.
15. A. Madry. Computing maximum flow with augmenting electrical flows. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 593–602. IEEE, 2016.
16. L. Qingsong, G. Betsy, and S. Shashi. Capacity constrained routing algorithms for evacuation planning: A summary of results. In *International Symposium on Spatial and Temporal Databases*, pages 291–307. Springer, 2005.
17. E.D. Sontag. Real addition and the polynomial hierarchy. *Information Processing Letters*, 20(3):115–120, 1985.
18. É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

A Examples and Proofs

A.1 A proof of the reduction in the proof of Theorem 2

Assume that ψ is satisfiable and let τ be a satisfying assignment for ψ. Consider the following saturating preflow f: In each vertex d_i, the incoming flow of $2k$ is directed to the vertex x_i if x_i holds in τ and otherwise it is directed to the vertex $\bar{x}_i$. Then, in these vertices the incoming flow is greater than the outgoing capacity and therefore the outgoing edges are saturated. In every vertex C_i with a positive incoming flow, f directs a flow of 1 to the sink u and the rest is directed to t. Since

τ satisfies ψ, every vertex C_i has a positive incoming flow and therefore $val(f) = kn - m$. Hence $uval(\mathcal{G}) < kn - m + 1$.

Assume now that ψ is not satisfiable and let f be a saturating preflow. For each vertex d_i, the outgoing flow in at least one outgoing edge is greater than or equal to k. We denote by l_i a literal in $\{x_i, \bar{x}_i\}$ that has an incoming flow of at least k. These literals induce an assignment to $x_1, \ldots, x_n$. By our assumption, this assignment does not satisfy ψ. Thus, at least one vertex in $C_1, \ldots, C_m$ has an incoming flow of 0 from the vertices $l_1, \ldots, l_n$. Hence, the total outgoing flow from the vertices $l_1, \ldots, l_n$ is kn and from this flow at most $m - 1$ is lost. Therefore, the incoming flow to t is at least $kn - (m - 1)$ and thus $uval(\mathcal{G}) \geq kn - m + 1$.

A.2 A proof of Theorem 3

The network $\mathcal{G}$ constructed in the reduction in the proof of Theorem 2 is such that $uval(\mathcal{G}) \leq kn-m$ if the given CNF-SAT formula ψ is satisfiable and $uval(\mathcal{G}) \geq kn - m + 1$ otherwise. Let $\mathcal{G}'$ be the network obtained from $\mathcal{G}$ by adding a new vertex v, making v the target of $\mathcal{G}'$, adding an edge with capacity $kn - m$ from t to a sink, and an edge with capacity 1 from t to v. Thus, if the flow that reaches t is at most $kn - m$, then all of it can be directed to the sink. Consequently, if ψ is satisfiable, then $uval(\mathcal{G}') = 0$, and otherwise, $uval(\mathcal{G}') \geq 1$. Since every approximation algorithm (within any factor) can determine whether the value is positive or 0, then approximation within any factor is NP-hard.

A.3 Correctness proof of the algorithm in the proof of Theorem 6

We first show that there exists a preflow in $\mathcal{F}_L$ with flow loss greater than or equal to β for some $\beta \geq 0$ iff there exists a flow in $\mathcal{G}'$ with value C_s and cost less than or equal to $-C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \beta$. Assume that there exists a preflow f in $\mathcal{F}_L$ with $L_f \geq \beta$. This preflow induces a flow f' in $\mathcal{G}'$ where for every $v \in L$ we have $f'(\langle v, t' \rangle) = l_f(v)$. Since f is a saturating preflow, then $val(f') = C_s$. Also, since $\sum_{u \in L} l_f(v) \geq \beta$ and for every $e \in L \times V$ we have $f(e) = c(e)$, then $cost(f') \leq -C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \beta$. Conversely, assume that there exists a flow f' in $\mathcal{G}'$ with $val(f') = C_s$ and $cost(f') \leq -C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \beta$. It is known that for every network flow with costs there exists an integral min-cost max-flow [1]. Let f'' be an integral min-cost max-flow in $\mathcal{G}'$. Thus, $val(f'') = C_s$ and $cost(f'') \leq -C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \beta$. Since $cost(f'') \leq -C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e)$ and f'' is integral, then for every $e \in L \times V$ we must have $f''(e) = c(e)$. Indeed, otherwise $cost(f'') = \sum_{e \in E'} f''(e)a(e) = (-C \cdot |V|^2) \cdot \sum_{e \in L \times V} f''(e) + (-1) \cdot \sum_{e \in L \times \{t'\}} f''(e) \geq (-C \cdot |V|^2) \cdot [(\sum_{e \in L \times V} c(e)) - 1] - |V| \cdot (|V| - 1) \cdot C > -C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e) + C \cdot |V|^2 - C \cdot |V|^2 = -C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e)$. The flow f'' induces a preflow f in $\mathcal{F}_L$ with $L_f \geq \beta$.

Since there exists a preflow in $\mathcal{F}_L$ with flow loss greater than or equal to β iff there exists a flow in $\mathcal{G}'$ with value C_s and cost less than or equal to $-C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \beta$, then we have that $\mathcal{F}_L \neq \emptyset$ iff there exists a flow in $\mathcal{G}'$ with value C_s and cost less than or equal to $-C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e)$. Note that a flow in $\mathcal{G}'$ with value C_s is a maximum flow. Assume that $\mathcal{F}_L \neq \emptyset$ and let $\gamma \geq 0$ be the maximal flow loss in a preflow in $\mathcal{F}_L$, namely $\gamma = \max\{L_f : f \in \mathcal{F}_L\}$. Note that according to Theorem 5 there exists a maximum for this set. Let $-C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \beta$ be the cost of a min-cost max-flow in $\mathcal{G}'$. Since there exists a preflow in $\mathcal{F}_L$ with flow loss γ then there exists a maximum flow f' in $\mathcal{G}'$ with $cost(f') \leq -C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \gamma$. Therefore, $\beta \geq \gamma$. Conversely, since there exists a flow in $\mathcal{G}'$ with value C_s and cost less than or equal to $-C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \beta$, then there exists a preflow in $\mathcal{F}_L$ with flow loss greater than or equal to β, and thus $\gamma \geq \beta$. Hence $\beta = \gamma$. Therefore, if the min-cost max-flow in $\mathcal{G}'$ has value C_s and cost $\alpha = -C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \beta$ for some $\beta \geq 0$, then the maximal flow loss of a preflow in $\mathcal{F}_L$ is $\beta = -\alpha - C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e)$, and thus the minimal value of a preflow in $\mathcal{F}_L$ is $C_s - \beta = C_s + \alpha + C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e)$.

Accordingly, the algorithm finds a min-cost max-flow f' in $\mathcal{G}'$. If $val(f') < C_s$ or $cost(f') > -C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e)$, then $\mathcal{F}_L = \emptyset$. Otherwise, the minimal value of a preflow in $\mathcal{F}_L$ is $C_s + cost(f') + C \cdot |V|^2 \cdot \sum_{e \in L \times V} c(e)$. Since the min-cost max-flow problem can be solved in time $O(|E|^2 log|V| + |E||V|log^2|V|)$ [1] and there are $2^{|H|}$ subsets of funnels to check, the required complexity follows.

A.4 A proof of Theorem 8

The algorithm shown in the proof of Theorem 6 runs over all the subsets of H and for each subset $L \subseteq H$ finds a minimum-value preflow in $\mathcal{F}_L$. In the proof of Theorem 6 it is shown that if $\mathcal{F}_L \neq \emptyset$ and $\gamma = \max\{L_f : f \in \mathcal{F}_L\}$, then a min-cost max-flow in $\mathcal{G}'$ has value C_s and cost $-C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \gamma$. Since the min-cost max-flow problem always has a solution in which all flows are integral [1], let f' be an integral min-cost max-flow in $\mathcal{G}'$. Thus $val(f') = C_s$ and $cost(f') = -C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \gamma$. In the proof of Theorem 6 it is also shown that an integral flow in $\mathcal{G}'$ with value C_s and cost $-C \cdot |V|^2 \cdot (\sum_{e \in L \times V} c(e)) - \gamma$ induces a preflow in $\mathcal{F}_L$ with flow loss γ. Hence, the preflow that f' induces in $\mathcal{F}_L$ is integral and has a maximal flow loss. Since this is the case for every $L \subseteq H$, then the UF problem has an integral-flow solution.

A.5 A proof of Theorem 10

Let $\mathcal{G} = \langle V, E, c, s, t \rangle$, and let f be a minimum-value controlled saturating preflow that respects g. We construct a controlled saturating preflow f' that respects g' and such that $val(f') \leq val(f)$. The preflow f' is obtained from f by reducing flows in edges according to the following process: We start with $f' = f$. For every $e \in E^{s\rightarrow}$, if $g'(e) < g(e)$ then we change f' such that $f'(e) = g'(e)$. Now, for some vertices the incoming flow in f' might be less than the outgoing flow. We fix this with the following steps. Let $V = \{v_1, \ldots, v_n\}$. We choose a vertex $v \in V \setminus \{s, t\}$ such that $f'(v\rightarrow) - f'(\rightarrow v)$ is maximal. Then, we find the minimal i such that $\langle v, v_i \rangle \in E$ and $f'(\langle v, v_i \rangle) > 0$ and reduce the flow in $\langle v, v_i \rangle$ such that $f'(v\rightarrow) = f'(\rightarrow v)$. If there is not enough flow in $\langle v, v_i \rangle$, then we reduce the flow there to 0 and then find the next i such that $\langle v, v_i \rangle \in E$ and $f'(\langle v, v_i \rangle) > 0$ and reduce the flow there too. We repeat this until $f'(v\rightarrow) = f'(\rightarrow v)$. Then we find the next vertex $v \in V \setminus \{s, t\}$ such that $f'(v\rightarrow) - f'(\rightarrow v)$ is maximal and repeat the above steps. We finish this process when for every $v \in V \setminus \{s, t\}$, we have $f'(v\rightarrow) = \min\{f'(\rightarrow v), c(v\rightarrow)\}$. Since f' is obtained from f by reducing flow in some edges, then $val(f') \leq val(f)$.

We show that the above process terminates after a finite number of iterations. Assume that the process does not terminate. Since we only reduce flows, there is a finite number of iterations in which a flow in an edge is reduced to 0. Thus, from some point, for every $v \in V$, there is an edge $e_v \in E^{v\rightarrow}$ such that a flow reduction in $E^{v\rightarrow}$ occurs only in e_v. Let $U \subseteq V$ be the vertices from which we reduce flow infinitely many times. Thus, from some point we reduce flow only for edges e_v for $v \in U$. Every vertex $v \in U$ must have a vertex $u \in U$ such that $e_u = \langle u, v \rangle$. Therefore, the subgraph induced by the vertices U and the edges $\{e_v : v \in U\}$ consists of disjoint cycles. Hence, if for some $v \in U$ we have $e_v = \langle v, w \rangle$ then $w \in U$. Note that from some point, for every vertex $v \in U$ the incoming flow is less than its outgoing capacity. Thus, from some point, if we reduce a flow of ϵ from an edge $e_u = \langle u, v \rangle$ then $u, v \in U$ and a flow of at least ϵ should be reduced later from the edge e_v. Since in every iteration we choose a vertex from which the flow reduction that is needed is maximal, then in the next iteration a flow of at least ϵ will be reduced. Hence, a flow of at least ϵ will be reduced in every iteration from this point and on. Since the flows are bounded below by 0 then this process cannot continue for infinitely many iterations.

A.6 An example of the reduction in Theorem 12

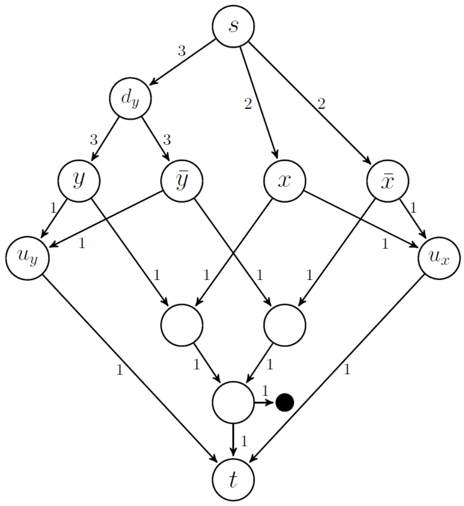

Figure 7: The flow network $\mathcal{G}_\theta$ for $\theta = \exists x \forall y (x \vee y) \wedge (\bar{x} \vee \bar{y})$.

A.7 A proof of the reduction in the proof of Theorem 12

we prove that θ holds iff there is a regulator g with bound $(2k+1)m + (k+1)n$ such that for every controlled saturating preflow f that respects g we have $val(f) \geq m+n+1$. Assume first that θ holds. Let π be an assignment for X such that for every assignment for Y, the formula ψ holds. Consider the regulator g where for every vertex $u \in d_{y_1}, \ldots, d_{y_m}$ we have $g(\langle s, u\rangle) = 2k+1$, for every vertex x in X such that $\pi(x) = 1$ we have $g(\langle s, x\rangle) = k+1$ and $g(\langle s, \bar{x}\rangle) = 0$, and for every vertex $x \in X$ such that $\pi(x) = 0$ we have $g(\langle s, x\rangle) = 0$ and $g(\langle s, \bar{x}\rangle) = k+1$. Note that $\sum_{e \in E^{s\rightarrow}} g(e) = (2k+1)m + (k+1)n$. Thus, the input flows to $\mathcal{G}_\psi$ for the variables X must be consistent with the assignment π and for every i, the vertex u_{x_i} must have an incoming flow of 1. Since for each i, the vertex d_{y_i} has an incoming flow of $2k+1$, then either y_i or $\bar{y}_i$ must have an incoming flow of at least $k+1$. Therefore, for every i, the outgoing edges of either y_i or $\bar{y}_i$ are saturated. Hence, according to Lemma 13 (1), a minimum-value controlled saturating preflow should direct a flow of $2k+1$ to either y_i or $\bar{y}_i$. Thus, the input flows to $\mathcal{G}_\psi$ for the variables Y are consistent with some assignment τ to these variables. According to Lemma 13 (2), the unfortunate value of $\mathcal{G}_\psi$ for these input flows is the value of ψ for the assignments π and τ, which is 1. Therefore, for every controlled saturating preflow f that respects g we have $val(f) = m+n+1$.

Now, assume that θ does not hold. We show that for every regulator g in $\mathcal{G}_\theta$ such that $\sum_{e \in E^{s\rightarrow}} g(e) \leq (2k+1)m + (k+1)n$, there is a controlled saturating preflow f that respects g such that $val(f) < m+n+1$. First, in order to ensure an incoming flow of at least 1 to every vertex u_{y_i}, the regulator g must have $g(\langle s, d_{y_i}\rangle) = 2k+1$. Since $\sum_{e \in E^{s\rightarrow}} g(e) \leq (2k+1)m + (k+1)n$, then in order to ensure also an incoming flow of 1 to every vertex u_{x_i}, the regulator g must have $g(\langle s, x_i\rangle) = k+1$ or $g(\langle s, \bar{x}_i\rangle) = k+1$. Therefore, the input flows to $\mathcal{G}_\psi$ for the variables X are consistent with some assignment π. According to the assumption, for every assignment to X there is an assignment to Y such that ψ does not hold. Let τ be such an assignment to Y. A preflow that assigns to the vertices $Y \cup \bar{Y}$ flows that respect τ, namely a flow of $2k+1$ to the literals that hold in τ, will result in input flows to $\mathcal{G}_\psi$ for the variables Y that are consistent with τ. According to Lemma 13 (2), in this case the unfortunate value of $\mathcal{G}_\psi$ is 0 and therefore the incoming flow to t will be less than $m+n+1$.

A.8 A proof of Theorem 14

According to Theorem 2, given an integral regulator g, deciding whether the minimal value of a controlled saturating preflow that respects g is at least γ is in co-NP. Thus we can guess an integral regulator g such that $\sum_{e \in E^{s\to}} g(e) \leq \alpha$ and then use an NP oracle in order to check whether the minimal value of a controlled saturating preflow that respects g is at least γ. Hence the Σ_2^P upper bound.

For the lower bound, note that the reduction described in the proof of Theorem 12 holds also for integral regulators.

A.9 A proof of Theorem 15

Consider a flow network $\mathcal{G} = \langle V, E, c, s, t\rangle$. For every vertex $v \in V \setminus \{s,t\}$, we construct a flow network $\mathcal{G}_v$ in which every edge e is associated with a weight $a_v(e)$. The weighted flow network $\mathcal{G}_v = \langle V_v, E_v, a_v, c_v, s, t_v\rangle$ is obtained by extending $\mathcal{G}$ as follows. We add a new vertex t_v and new edges from every vertex in V to t_v, thus, $V_v = V \cup \{t_v\}$ and $E_v = E \cup V \times \{t_v\}$. The capacity c_v for the new edges is large and for every $e \in E$ we have $c_v(e) = c(e)$. For every edge $e \in E^{\to v}$ we have $a_v(e) = 1$ and for every $e \notin E^{\to v}$ we have $a_v(e) = 0$. We show that flow loss is possible in $\mathcal{G}$ iff there is a vertex $v \in V$ such that the maximum weighted flow in $\mathcal{G}_v$ has weight greater than $\sum_{v' \in V} c(\langle v, v'\rangle)$, that is, there is a flow f in $\mathcal{G}_v$ such that $\sum_{e \in E_v} a_v(e)f(e) > \sum_{v' \in V} c(\langle v, v'\rangle)$. Intuitively, the maximum weighted flow in $\mathcal{G}_v$ has weight greater than $\sum_{v' \in V} c(\langle v, v'\rangle)$ iff an incoming flow of more than $\sum_{v' \in V} c(\langle v, v'\rangle)$ to v is possible in $\mathcal{G}$ iff flow loss is possible in v. Checking for every $v \in V$ whether the maximum weighted flow in $\mathcal{G}_v$ has weight greater than $\sum_{v' \in V} c(\langle v, v'\rangle)$ can be done in polynomial time by solving a linear program.

Assume that flow loss is possible. Let f be a saturating preflow in $\mathcal{G}$ such that $l_f(v) > 0$ for some vertex $v \in V$, that is, the incoming flow to v is greater than its outgoing capacity. The preflow f induces a flow f' in $\mathcal{G}_v$, where the flow losses in f and the flow in t are directed to the vertex t_v. Thus, for every $u \in V \setminus \{s,t\}$ we have $f'(\langle u, t_v\rangle) = l_f(u)$ and $f'(\langle t, t_v\rangle) = f(\to t)$. In the flow f' the incoming flow to v is greater than the outgoing capacity from v in $\mathcal{G}$ and therefore $\sum_{e \in E_v} a_v(e)f'(e) > \sum_{v' \in V} c(\langle v, v'\rangle)$.

Now, assume that there is a vertex $v \in V \setminus \{s,t\}$ such that there is a flow in $\mathcal{G}_v$ with weight greater than $\sum_{v' \in V} c(\langle v, v'\rangle)$. Hence, there is a flow f in $\mathcal{G}_v$ such that the incoming flow to the vertex v is greater than $\sum_{v' \in V} c(\langle v, v'\rangle)$. The flow f induces a preflow f' in $\mathcal{G}$, where for every $e \in E$ we have $f'(e) = f(e)$. Note that f' is a preflow in $\mathcal{G}$ but it may not be a saturating preflow. We change f' according to the following steps in order to obtain a saturating preflow in $\mathcal{G}$ such that the flow in each edge $e \in E$ is greater or equal to $f(e)$. We denote $V = \{v_1, \ldots, v_n\}$. First, for every $e \in E^{s\to}$ we change f' such that $f'(e) = c(e)$. Then, we choose a vertex $u \in V \setminus \{s,t\}$ such that $\min\{f'(\to u), c(u\to)\} - f'(u\to)$ is maximal. We find the minimal i such that $\langle u, v_i\rangle \in E$ and $f'(\langle u, v_i\rangle) < c(\langle u, v_i\rangle)$ and increase the flow in $\langle u, v_i\rangle$ such that $f'(u\to) = \min\{f'(\to u), c(u\to)\}$. If there is not enough free capacity in $\langle u, v_i\rangle$ then we increase the flow there to $c(\langle u, v_i\rangle)$ and find the next i such that $\langle u, v_i\rangle \in E$ and $f'(\langle u, v_i\rangle) < c(\langle u, v_i\rangle)$ and increase the flow there also. We repeat this until $f'(u\to) = \min\{f'(\to u), c(u\to)\}$. Then, we find the next $u \in V \setminus \{s,t\}$ such that $\min\{f'(\to u), c(u\to)\} - f'(u\to)$ is maximal and repeat the above steps. We finish this process when for every $u \in V \setminus \{s,t\}$ we have $f'(u\to) = \min\{f'(\to u), c(u\to)\}$. Since in this process we only increase flows, then we still have that the incoming flow to the vertex v is greater than $\sum_{v' \in V} c(\langle v, v'\rangle)$ and therefore there is a flow loss in v.

We now show that the above process terminates after a finite number of iterations. Assume that the process does not terminate. Since we only increase flows, there is a finite number of iterations in which a flow in an edge $e \in E$ is increased to $c(e)$. Thus, from some point, for every $u \in V$ there is an edge $e_u \in E^{u\rightarrow}$ such that a flow increase in $E^{u\rightarrow}$ occurs only in e_u. Let $U \subseteq V$ be the vertices in which we commit a flow increase infinitely many times. Thus, from some point we add flow only to edges e_u for $u \in U$. Every vertex $w \in U$ must have a vertex $u \in U$ such that $e_u = \langle u, w\rangle$. Therefore, the subgraph induced by the vertices U and the edges $\{e_u : u \in U\}$ consists of disjoint cycles. Hence, if for some $u \in U$ we have $e_u = \langle u, w\rangle$ then $w \in U$. Note that for every vertex $u \in U$ the flow in e_u is always less than $c(e_u)$. Thus, from some point, if we add a flow of ϵ to an edge $e_u = \langle u, w\rangle$ then $u, w \in U$ and a flow of at least ϵ should be added later to the edge e_w. Since in every iteration we choose a vertex to which the flow increase that is needed is maximal, then in the next iteration a flow of at least ϵ will be added. Hence, a flow of at least ϵ will be added in every iteration from this point and on. Since the sum of the flows in the edges of $\mathcal{G}$ is bounded above by $\sum_{e\in E} c(e)$, then this process cannot continue for infinitely many iterations.

A.10 A proof of the reduction in the proof of Theorem 16

Assume that ψ is satisfiable and let π be a satisfying assignment. Consider the regulator g in which for each literal $z \in Z$ that holds in π we have $g(\langle s, z\rangle) = 2$ and for every other edge $e \in E^{s\rightarrow}$ we have $g(e) = 0$. Thus, the total outgoing flow from s is $2n$. We show that every saturating preflow in $\mathcal{G}$ that respects g has no flow losses. First, a flow loss cannot occur in a vertex u_i because for every i either x_i or $\bar{x}_i$ has an incoming flow of 0. A flow loss also cannot occur in a vertex $z \in Z$ since its outgoing capacity is at least 2. Finally, since π satisfies ψ, each clause C_i contains at least one literal that holds in π and therefore there is at least one literal $z \in Z$ with $g(\langle s, z\rangle) > 0$ such that $\langle z, C_i\rangle \notin E$. Thus, the incoming flow to C_i is at most $n - 1$ and hence a flow loss cannot occur in the vertex C_i.

Now assume that ψ is not satisfiable and let g be a regulator with $\sum_{e\in E^{s\rightarrow}} g(e) \geq 2n$. We show that there is a saturating preflow in $\mathcal{G}$ that respects g and has flow losses. First, if there is some i such that $g(\langle s, x_i\rangle) + g(\langle s, \bar{x}_i\rangle) > 2$ then a flow greater than 1 can be directed to u_i and thus a flow loss occurs. Since $\sum_{e\in E^{s\rightarrow}} g(e) \geq 2n$ we assume that for every i we have $g(\langle s, x_i\rangle) + g(\langle s, \bar{x}_i\rangle) = 2$. If for some i we have $g(\langle s, x_i\rangle) > 0$ and $g(\langle s, \bar{x}_i\rangle) > 0$ then a flow greater than 1 can be directed to u_i and thus a flow loss occurs. Otherwise, for every i we have $g(\langle s, x_i\rangle) = 2$ or $g(\langle s, \bar{x}_i\rangle) = 2$ and therefore g induces an assignment to the variables $x_1, \ldots, x_n$. Since ψ is not satisfiable, there is a clause C_i that does not contain literals $z \in Z$ with $g(\langle s, z\rangle) = 2$ and therefore a flow of n can be directed to the vertex C_i, resulting in a flow loss.

A.11 A proof of Theorem 17

Given an integral regulator, deciding whether a flow loss is possible can be done in polynomial time according to Theorem 15. Hence the NP upper bound.

For the lower bound, note that the reduction shown in the proof of Theorem 16 holds also for integral regulators.