Flow Games

Orna Kupferman[1], Gal Vardi[1], and Moshe Y. Vardi[2]

1 School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel*
2 Department of Computer Science, Rice University, Houston, Texas, USA†

Abstract

In the traditional maximal-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. While the problem has been extensively used in order to optimize the performance of networks in numerous application areas, it corresponds to a setting in which the authority has control on all vertices of the network. Today's computing environment involves parties that should be considered adversarial. We introduce and study *flow games*, which capture settings in which the authority can control only part of the vertices. In these games, the vertices are partitioned between two players: the authority and the environment. While the authority aims at maximizing the flow, the environment need not cooperate. We argue that flow games capture many modern settings, such as partially-controlled pipe or road systems or hybrid software-defined communication networks. We show that the problem of finding the maximal flow as well as an optimal strategy for the authority in an acyclic flow game is Σ_2^P-complete, and is already Σ_2^P-hard to approximate. We study variants of the game: a restriction to strategies that ensure no loss of flow, an extension to strategies that allow non-integral flows, which we prove to be stronger, and a dynamic setting in which a strategy for a vertex is chosen only once flow reaches the vertex. We discuss additional variants and their applications, and point to several interesting open problems.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Flow networks, Two-player Games, Algorithms

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.38

1 Introduction

A *flow network* is a directed graph in which each edge has a capacity, bounding the amount of flow that can go 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 with a source vertex and a target vertex and searches for a maximal flow from the source to the target [9, 19]. The problem was first formulated and solved in the 1950's [16, 17]. It has attracted much research on improved algorithms [11, 10, 20] and applications [2].

The maximum-flow problem can be applied in many settings in which something travels along a network. This covers numerous applications domains, including traffic in road or rail systems, fluids in pipes, currents in an electrical circuit, packets in a communication network, and many more [2]. Less obvious applications involve flow networks that are constructed in order to model settings with an abstract network, as in the case of elimination in partially completed tournaments [32] or scheduling with constraints [2]. 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,

* The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)
† Work supported in part by NSF grants CCF-1319459 and IIS-1527668, and by NSF Expeditions in Computing project "ExCAPE: Expeditions in Computer Augmented Program Engineering"

minimum path cover, maximum edge-disjoint or vertex-disjoint path, and many more [9, 2]. Variants of the maximum-flow problem can accommodate further settings, like circulation problems, where there are no sink and target vertices, yet there is a lower bound on the flow that needs to be traversed along each edge [34], networks with multiple source and target vertices [12], networks with costs for unit flows, and more.

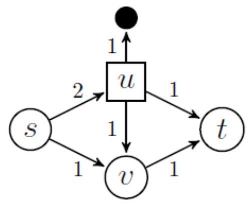
All studies of flow networks so far assume that all the vertices in the network can be controlled by a central authority. That is, the maximum-flow algorithm finds a flow that directs, in all vertices of the network, incoming flow into the outgoing edges. In many applications of flow networks, however, only some of the vertices of the network can be so controlled: In road systems, police can direct traffic in only some of the junctions; in pipe systems, a company may have control only on some of the valves; and in communication networks, the authority may control only in some of the routers. In particular, in the area of *software defined network* (SDN), there is growing interest in hybrid networks, where some vertices are software-defined by a logically-centralized controller and in some others the routers behave as they see fit (e.g., running traditional or proprietary routing protocols, making adversarial decisions, etc.). Moreover, new network elements (e.g., SDN switches) should be added to the network gradually, so that traffic in not suspended, which results in a hybrid network [1, 35].

The above applications suggest that the maximal-flow problem should be revisited, taking into account the *game-theoretic* nature of the setting. Indeed, in more and more applications, the authority can direct the flow only in a subset of the vertices. In the others, it is the environment that directs the flow. The environment need not cooperate, giving rise to a game between the authority and the environment.

We introduce and study *flow games*.[1] In a flow game, the vertices in the network are partitioned between two players, Player 0 and Player 1. Player 0 corresponds to the network authority, whose goal is to maximize the flow, while Player 1 corresponds to the hostile environment. A strategy for a player advices him how to direct flow that enters vertices under his control. Formally, for each vertex u, let E_u denote the set of edges outgoing from u. Also, for each edge e, let $c(e) \in \mathbb{N}$ denote its capacity. Then, for each vertex u controlled by the player, a strategy for the player includes a policy $f_u : \mathbb{N} \to \mathbb{N}^{E_u}$ that maps every incoming flow $x \in \mathbb{N}$ to a function describing how x is partitioned among the edges outgoing from u. For each incoming flow $x \in \mathbb{N}$ and edge $e \in E_u$, we require that $f_u(x)(e) \leq c(e)$ and $\sum_{e \in E_u} f_u(x)(e) = \min\{x, \sum_{e \in E_u} c(e)\}$. Thus, $f_u(x)$ assigns to each edge outgoing from u a flow that is bounded by its capacity. Also, when the incoming flow is larger than the capacity of the outgoing edges (which bounds the outgoing flow), then flow *gets stuck* or *leaks* and the outgoing flow is lower than the incoming flow. In addition, a policy for the source s assigns to each edge outgoing from s a flow that is bounded by its capacity. The goal of Player 0 is to maximize the flow that enters the target t, no matter how Player 1 plays. Thus, it is a *Stackelberg game* where Player 0 is the leader [29]. Note that the definition of flow in a flow game is different from the traditional definition, which corresponds to the case all vertices belong to Player 0, and in which the "flow conservation" property is respected in all vertices. Indeed, in the game setting, Player 1 may cause flow to gets stuck or to leak by directing the flow to vertices whose outgoing capacity is smaller than the incoming flow.

▶ **Example 1.** Consider the flow game in the figure below. We represent vertices of Player 0 by circles and vertices of Player 1 by squares. Sinks, namely vertices other than the target that do not have outgoing edges, are represented by filled circles. In the classical max-flow problem, the maximal flow is 2, which is also the minimal cut in the network.

[1] Not to confuse with games in which players *cooperate* in order to construct a sub-graph that maximizes the flow in the traditional setting, which are also termed flow games (c.f., [24]).



We can view the network as a road system, where the vertex u, which belongs to Player 1, is a junction in which the police does not direct incoming traffic. Note that the traffic is not lost in u, it is only that the police cannot direct it to t, and so it may go to the sink, where it gets lost, or to vertex v. In the context of road systems, flow loss means that the outgoing flow is less than the incoming flow and thus a traffic jam occurs. Likewise, the network may model a communication network in which the vertex u is a router whose software we do not control. Again, unless the outgoing channels are filled, the router does not dismiss packets that reach it, but it can direct the packets however it chooses. A strategy for Player 0 that directs 1 unit of flow from s to v and no flow from s to u ensures a flow of 1. Also, since the incoming flow to u is at most 2, and Player 1 can direct it to the sink and to vertex v, Player 0 does not have a strategy that ensures a flow of more than 1. Hence, the maximum flow that Player 0 can ensure is 1. ◀

In essence, what we propose here is to lift the maximum-flow problem from its classical *one-player* setting to a *two-player* setting. Such a transition has been studied in computer science in many contexts. In graph theory, this transition corresponds to going from graph reachability to alternating graph reachability (*Path Systems*) [7]. In complexity theory, this is the transition from nondeterministic computation to alternating computation [6]. In logic, this is the transition from Boolean satisfiability [8] to quantified Boolean satisfiability [33]. In temporal reasoning, this is the transition from satisfiability [27] to temporal synthesis [30]. Generally, this can be viewed as a transition from *closed systems*, which are completely under our control, to *open systems*, in which we have to contend with adversarial environments. The absence of regulation by some central authority is indeed a driving theme of *algorithmic game theory*, cf. [28], inspired by the open nature of today's computing environments. Thus, this work can be viewed as a study of maximal flow in a network from the perspective of algorithmic game theory.[2] In addition, flow games extend less direct applications of the maximum-flow problem to open settings. In particular, many of the constrained scheduling problems that are reduced to maximum-flow problems [2] actually correspond to cases in which not all involved entities may be controlled. We note that one could also consider flow games among more than two players, and also examine classical algorithmic game-theory questions: existence of a Nash equilibrium, stability inefficiency, and many more (see Section 7). Two-player games are an important and popular first step in studying richer multiplayer settings. Beyond the technical interest (lower bounds apply to the multiplayer setting, and upper bounds are often extendible), the two-player setting captures also an environment composed of different entities. Even when these different entities have no centralized authority, a worst case approach to their joint behavior, which is the one we want to take, amounts to viewing them as a single player.

Other game variants of classical graph-theory problems include the study of *Turán numbers* and *Saturation numbers* for various graph monotone decreasing properties such as triangle freedom, planarity, and 3-colorability. There, two players take turns choosing edges of a given graph as long as the property is preserved in the generated subgraph. One player aims for the process to be long (that is, for the generated subgraph to be big), and the second aims for it to be short [18, 21]. Likewise, the *game chromatic number* of graphs is the smallest number of colors with which a graph can be colored when two players alternate turns in deciding the color of the next vertex [5]. Finally, in *spanning-tree games*, the players alternate turns choosing edges of a weighted graph as long as a

[2] Different aspects of networks have already been extensively studied from the perspectives of algorithmic game theory. This includes, for example, network formation games [4] or incentive issues in interdomain routing and the BGP protocol [13]. We are the first, however, to consider the maximal-flow problem from this perspective.

cycle is not closed. One player aims to maximize the weight of the generated spanning tree and the second aims to minimize it [22].

The transition from closed to open systems does not come without a computational price. Graph reachability is in NLOGSPACE, while alternating reachability is PTIME-complete. Boolean satisfiability is NP-complete, while quantified Boolean satisfiability is PSPACE-complete. Temporal satisfiability is PSPACE-complete, while temporal synthesis is 2EXPTIME-complete. Understanding the computation cost from going from network flow to network-flow games is a major theme of this work.

We study here *acyclic* game networks. There, given strategies for the players, it is possible to calculate the flow by following a topological ordering of the vertices. We first show that the problem of finding the maximal flow as well as an optimal strategy for the authority in a flow game is Σ_2^P-complete, and is already Σ_2^P-hard to approximate. We point to easy fragments and variants: when the network is a *tree*, and the goal is to maximize the flow that reaches the leaves, and when the environment may *swallow* flow.

Once we find the complexity of flow games, we turn to study three important variants of the setting. Recall that when the incoming flow is larger than the capacity of the outgoing edges, then flow is lost: it gets stuck or leaks. Sometimes it is desirable to find a strategy of Player 0 such that for every strategy of Player 1, there is no loss of flow. Indeed, in some applications the authority cannot tolerate loss of traffic and is willing to reduce the flow in order to ensure no loss. We study the problem of finding the maximal flow that the authority can transfer while ensuring no loss. As good news, we show that the problem of deciding whether some positive flow can be transferred is PTIME-complete, as opposed to the Σ_2^P-completeness in the “with loss" setting. Finding, however, the maximal flow that can be guaranteed with no loss stays Σ_2^P-complete.

Our definition of strategies assumes that policies are integral: vertices receive integral incoming flow and partition it to integral flows in the outgoing edges. Integral-flow games arise naturally in settings in which the objects we transfer along the network cannot be partitioned into fractions, as is the case with cars, packets, and more. Sometimes, however, as in the case of liquids, flow can be partitioned arbitrarily. In the traditional maximum-flow problem, it is well known that the maximum flow can be achieved by integral flows [16]. We show that, interestingly, the game setting makes strategies that use non-integral flow stronger: partitioning outgoing flows into non-integers may increase the flow that Player 0 can ensure. Moreover, the gain cannot be bounded by a constant. We leave open the problem of finding the flow that Player 0 can ensure when using non-integral strategies. Hardness in Σ_2^P holds also for this setting. Yet, as real numbers are second-order creatures, the problem may be undecidable.

As a third variant, we define *dynamic flow games*, in which the policy for a vertex u is chosen only when flow reaches it. More formally, the policy in u is chosen once flow in all edges that precede u in the topological order has traveled. We show that the dynamic setting enables a formulation of the problem by means of the first-order theories of integral addition or real addition, making the related decision problems decidable for both integral and non-integral flows [15, 14].

Finally, we discuss additional variants, which we leave for future research: The first is motivated by *evacuation* scenarios [25]: Consider, for example, a road network of a city, where the vertex s denotes the center of the city and the vertex t denotes the area outside of the city. In order to evacuate the center of the city, drivers are asked to navigate from s to t. In each vertex, every incoming driver chooses an arbitrary outgoing edge. If the outgoing capacity from a vertex is smaller than the incoming flow, then a traffic jam occurs, and flow is lost. We want to find the number of cars that are guaranteed to evacuate in the worst scenario. This corresponds to solving a flow-game in which all vertices belong to Player 1.[3] Then, in multiplayer flow games, the vertices of the network are partitioned among several (possibly more than 2) players, each having her target vertex. Finally, in partially-specified flow networks, a strategy is known for some vertices, and we need to find an optimal strategy for the others.

2 Flow Games

A *flow network* is $N = \langle V, E, c, s, t\rangle$, where V is a set of vertices, $E \subseteq V \times V$ is a set of directed edges, and $c : E \to \mathbb{N}$ is a capacity function, assigning to each edge an integral amount of flow that the edge can transfer, and $s, t \in V$ are source and target vertices. We assume that t is reachable from s and the capacities are given in unary. A *flow game* between two players, denoted Player 0 and Player 1, is $\mathcal{G} = \langle V_0, V_1, E, c, s, t\rangle$, where V_0 and V_1 are sets of vertices and $\langle V_0 \cup V_1, E, c, s, t\rangle$ is a flow network in which the set of vertices is partitioned between Player 0 and Player 1. Intuitively, Player 0 directs flow that arrives to vertices in V_0 and his goal is to maximize the flow from s to t. Then, Player 1 directs flow that arrives to vertices in V_1 and his goal is to minimize the flow from s to t. A *sink* is a vertex u with no outgoing edges. That is, there is no vertex v with $E(u, v)$. We assume that t is a sink, $s \in V_0$, and that no edge enters s. That is, there is no vertex v with $E(v, s)$.

Let $V = V_0 \cup V_1$. For a vertex $u \in V$, let E^u and E_u be the sets of incoming and outgoing edges to and from u, respectively. That is, $E^u = (V \times \{u\}) \cap E$ and $E_u = (\{u\} \times V) \cap E$. A *policy* for a vertex $u \in V$, for $u \neq s$, is a function that partitions an incoming flow between the outgoing edges. Formally, a policy for u is a function $f_u : \mathbb{N} \to \mathbb{N}^{E_u}$ such that for every flow $x \in \mathbb{N}$ and edge $e \in E_u$, we have $f_u(x)(e) \leq c(e)$ and $\sum_{e\in E_u} f_u(x)(e) = \min\{x, \sum_{e\in E_u} c(e)\}$. Thus, $f_u(x)$ assigns to each edge outgoing from u a flow that is bounded by its capacity. Also, when the incoming flow is larger than the capacity of the outgoing edges (which bounds the outgoing flow), then flow is *lost* and the outgoing flow is lower than the incoming flow. In practice, loss of flow may correspond to leaks – fluid in a pipe system that is lost when the system is overflowed, to traffic that gets stuck – in jammed road systems, or to packets that are thrown by routers all whose outgoing channels are filled. Note that this is different from the traditional definition of flow in a network, which corresponds to the case all vertices belong to Player 0, and in which the “flow conservation" property is respected. Note also that $f_u(0)(e) = 0$ for every e. For the source vertex s, a policy is a function $f_s \in \mathbb{N}^{E_s}$ such that for every edge $e \in E_s$, we have $f_s(e) \leq c(e)$.

A *flow* in a flow game is a function $f \in \mathbb{N}^E$ that assigns to each edge the flow that travels in it. We require that for every edge $e \in E_u$, we have $f(e) \leq c(e)$, and for every vertex $u \in V$, except for s and t, we have $\sum_{e\in E_u} f(e) = \min\{\sum_{e\in E^u} f(e), \sum_{e\in E_u} c(e)\}$. That is, the flow in each edge is bounded by its capacity, and the flow that leaves each vertex is the minimum between the flow that enters the vertex and the sum of the capacities of edges outgoing from it. We focus on the case where the graph $\langle V, E\rangle$ is acyclic. Then, given policies f_u for all vertices in $u \in V$, we can calculate the flow in the game as follows. First, we order the vertices in a topological ordering. Then, we start from the vertex s (we ignore every vertex preceding s), and use f_s to assign a flow to each edge in E_s. Now, we continue to the next vertex in the topological ordering. Whenever we reach a vertex u, the incoming flow to u, denoted x, has already been calculated. We then use $f_u(x)$ to assign a flow for each edge in E_u, and continue along the topological ordering until we reach t. Since the flow that enters a vertex u depends only on the sub-game that reaches u, it is easy to see that the calculation above is independent of the topological ordering. Indeed, if u_1 and u_2 are not ordered, then flow that leaves u_1 does not reach u_2, and vice versa.

[3] 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 [31]).

A strategy for Player 0 is a collection of policies, one for each vertex in V_0. Likewise, a strategy for Player 1 is a collection of policies, one for each vertex in V_1. Let F_0 and F_1 be the sets of all possible strategies of Player 0 and Player 1 respectively. Given strategies $\alpha \in F_0$ and $\beta \in F_1$, the flow in the game, denoted $f^{\alpha,\beta}$, can be calculated as described above. The *outcome* of the game with strategies α and β is then the amount of flow that reaches the target t. Thus, $outcome(\alpha, \beta) = \sum_{e \in E^t} f^{\alpha,\beta}(e)$. The *value* of the game, denoted $value(\mathcal{G})$, is defined by $\max_{\alpha \in F_0} \min_{\beta \in F_1} outcome(\alpha, \beta)$. That is, against every strategy of Player 0, we take the most hostile strategy of Player 1, and the value is obtained by considering the strategy of Player 0 that achieves the maximal flow no matter how Player 1 responds. Note that since we quantify on the strategies of Player 1 universally, this corresponds to a setting in which Player 0 proceeds first and determines the policies in all his vertices, and then Player 1 responds by determining the policies in all his vertices.

3 Problem Complexity

In this section we study the complexity of the problem of finding the value of a flow game. We consider the corresponding decision problem, which is parameterized by a threshold. Formally, the input to the *flow-game problem* (FG problem, for short) is a flow game $\mathcal{G}$ and a threshold $\gamma \in \mathbb{N}$, and the goal is to decide whether $value(\mathcal{G}) \geq \gamma$.

We show that the problem is complete 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.

▶ **Theorem 2.** *The FG problem is Σ_2^P-complete.*

Proof. We start with the upper bound. Consider a flow game $\mathcal{G}$ and a threshold $\gamma \in \mathbb{N}$. Strategies for Player 0 and Player 1 can be represented by polynomial-length strings. Given a strategy α for Player 0, the problem of checking whether there is a strategy β for Player 1 such that $outcome(\alpha, \beta) < \gamma$ is in NP. Consequently, deciding whether there is a strategy α for Player 0 such that for every strategy β for Player 1 we have $outcome(\alpha, \beta) \geq \gamma$, can be done by a nondeterministic polynomial-time Turing machine with an NP oracle.

We continue to the lower bound and describe a reduction from QBF$_2$: satisfiability for quantified Boolean formulas with 2 alternations of quantifiers, where the most external quantifier is "exists". Let ψ be a Boolean 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 game $\mathcal{G}_\theta$ such that the value of G_θ is 1 if θ holds and is 0 otherwise.

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 1, 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 $\bar{b}$. If the input to $\mathcal{C}_\psi$ is consistent then $\mathcal{C}_\psi$ computes the value of ψ for the corresponding assignment.

Now, we translate $\mathcal{C}_\psi$ to an *external-source flow game* $\mathcal{G}_\psi = \langle V_0, V_1, E, c, t \rangle$: a flow game 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

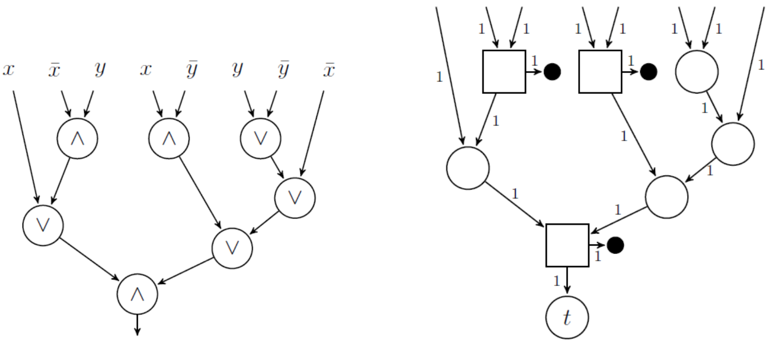


Figure 1 The circuit $\mathcal{C}_\psi$ and the external-source flow game G_ψ, for $\psi = x \vee (\bar{x} \wedge y) \wedge ((x \wedge \bar{y}) \vee (y \vee \bar{y} \vee \bar{x}))$.

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_0 in V_0 that has in-degree 2 and out-degree 1. Thus, if the incoming flow in each incoming edge to v_0 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_1 in V_1 that has in-degree 2 and out-degree 2, yet, one of the two edges that leaves v_1 leads to a sink. Accordingly, if the incoming flow in each incoming edge to v_1 is 0 or 1, then the outgoing flow in the edges that does not lead to the sink is 1 in all the policies for v_1 iff both incoming edges have flow 1. For example, the Boolean circuit $\mathcal{C}_\psi$ from Figure 1 is translated to the external-source flow game $\mathcal{G}_\psi$ to its right.

The following lemma can be proved by an induction on the structure of ψ.

► **Lemma 3.** *Consider a Boolean formula ψ and its corresponding external-source flow game $\mathcal{G}_\psi$.*

1. *Given input flows to $\mathcal{G}_\psi$, if we increase some input flow, then the new value of the game is greater than or equal to the original value.*
2. *Given input flows in $\{0, 1\}$ to $\mathcal{G}_\psi$, the value of the game is equal to the output of $\mathcal{C}_\psi$ with the same input. Thus, if the input flows to $\mathcal{G}_\psi$ corresponds to a consistent input to $\mathcal{C}_\psi$, then the value of $\mathcal{G}_\psi$ is the value of ψ for the corresponding assignment.*

We complete the reduction by constructing the flow game $\mathcal{G}_\theta$, which uses $\mathcal{G}_\psi$ as a sub-game. As can be seen in Figure 2, the game $\mathcal{G}_\theta$ has a source s that can direct flow to a layer of *variable vertices* – vertices associated with the variables in Z. The existentially quantified variables, namely these in X, are associated with V_0 vertices, and the universally quantified ones, namely these in Y, are associated with V_1 vertices. The source s can direct $2k$ units of flow to each variable vertex. The third layer of the game consists of *literal vertices*: each variable vertex d_z has two successors, associated with the literals z and $\bar{z}$. Vertices associated with literals in $X \cup \bar{X}$ are in V_1, whereas these associated with literals in $Y \cup \bar{Y}$ in V_0. The edges from a variable vertex to its literal vertices have capacity $2k$. Intuitively, the structure of the game, together with Lemma 3, imply that optimal strategies for both players directs all the $2k$ units that enter a variable vertex d_z into one of its literal vertices, which corresponds to a choice between z and $\bar{z}$. Consequently, there is a correspondence between the outgoing flow from the variable vertices and assignments to the variables, which induces an input to $\mathcal{C}_\psi$. Each outgoing edge from a literal vertex enters an input of $\mathcal{C}_\psi$ that corresponds to this literal.

In Appendix A.1 we describe the game $\mathcal{G}_\theta$ for the case $\psi = (x \vee y) \wedge (\bar{x} \vee \bar{y})$, and prove that the value of $\mathcal{G}_\theta$ is 1 if θ holds and is 0 otherwise. ◄

Note that from the proof of Theorem 2 we can also conclude that the FG problem is Σ_2^P-hard already for flow games in which the capacities of the edges are 1. Indeed, we can change the

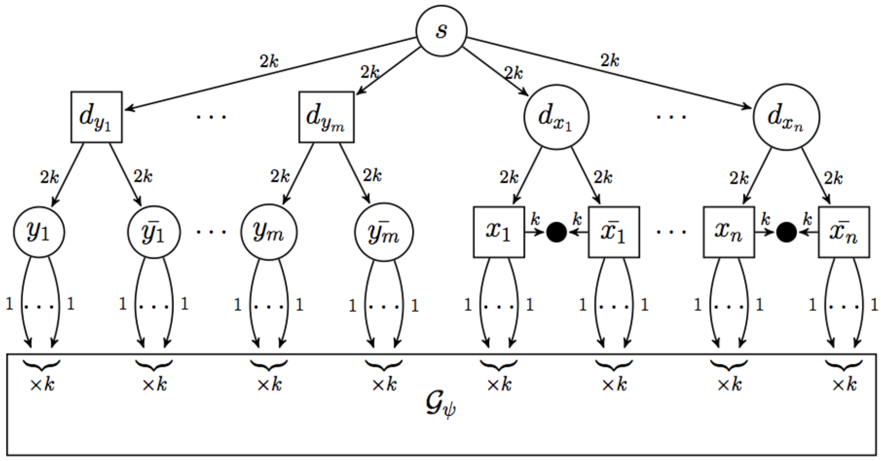


Figure 2 The game $\mathcal{G}_\theta$.

reduction so that in the game $\mathcal{G}_\theta$, an edge with capacity d is replaced by d parallel edges with capacity 1.

A natural question that arises is whether we can efficiently find an approximated solution for the FG problem. In particular, we say that a solution x is a δ-approximation for the value of a flow game $\mathcal{G}$ if $\frac{x}{\delta} \leq value(\mathcal{G}) \leq \delta \cdot x$. As we now show, finding an approximated solution to the FG problem is not easier than finding an exact solution. Formally, we have the following.

▶ **Theorem 4.** *It is Σ_2^P-hard to approximate the FG problem within any multiplicative factor.*

Proof. The reduction used for proving the Σ_2^P lower bound in the proof of Theorem 2 is such that the value of the flow game $\mathcal{G}_\theta$ is 1 if θ holds and is 0 otherwise. Thus, deciding whether the value of a flow game is 0 or 1 is already Σ_2^P-hard. Since every algorithm that approximates the solution of the problems within a multiplicative factor can determine whether the value of the game is positive or is 0, then approximated solution amounts to a precise solution, and is therefore Σ_2^P-hard. ◀

3.1 Feasible Cases

We describe a variant of the problem as well as a structural restriction on games for which the FG problem can be solved efficiently.

Swallowing Flow A most hostile environment is one that simply swallows all flow that reaches vertices it controls. That is, flow that reaches V_1 is lost. It is easy to analyze flow games in this setting, as they coincide with traditional networks in which the vertices in V_1 are eliminated. Indeed, Player 0 should avoid sending flow to vertices in V_1. It follows that maximal flow in this setting can be found in polynomial time. A variant of this setting is one in which there is a bound on the flow that Player 1 can swallow in each vertex. It is not hard to see that this variant can be reduced to our model by adding an edge to a sink from each vertex of Player 1. The capacity of the edge is the amount of flow that Player 1 can swallow in the vertex.

Tree Flow Games Let $\mathcal{G} = \langle V_0, V_1, E, c, s\rangle$ be a flow game without a target vertex. Let $V = V_0 \cup V_1$. Assume that the directed graph $\langle V, E\rangle$ is a tree, namely, the in-degree of every vertex in $V \setminus \{s\}$ is 1. The goal of Player 0 is to maximize the flow that enters the leaves of the tree. In Appendix A.2 we show that if the out-degree of every vertex is bounded by a constant then the FG problem can be solved in polynomial time. Essentially, the value of a game from a vertex u can be expressed as a min-max expression over the possible values of the games that start from the successors of u, which enables the formulation of the problem by means of dynamic programming. When the out-degrees of the vertices are constants, the latter can be solved in polynomial time.

4 No-Loss Flow Games

Recall that when the incoming flow is larger than the capacity of the outgoing edges, then flow is lost and the outgoing flow is lower than the incoming flow. Sometimes it is desirable to find a strategy of Player 0 such that for every strategy of Player 1, there is no loss of flow. We call such a strategy of Player 0 a *no-loss strategy*. Formally, $\alpha \in F_0$ is a no-loss strategy if for every strategy $\beta \in F_1$, there is no vertex $u \in V$ such that the flow that enters u in $f^{\alpha,\beta}$ is larger than the sum of the capacities of the edges that leave u. Thus, for all vertices $u \in V$ except for s and t, we have $\sum_{e \in E_u} f^{\alpha,\beta}(e) = \sum_{e \in E^u} f^{\alpha,\beta}(e)$.

No-loss strategies are required in applications in which the authority cannot tolerate loss of traffic and is willing to reduce the flow in order to ensure no loss. For example, when it involves leaks of hazards or loss of crucial packets in a communication network. The trivial strategy, where the flow is 0, is clearly a no-loss strategy. In this section we study optimal no-loss strategies, namely no-loss strategies that ensure a maximal flow. Given a flow game $\mathcal{G}$, let $nl\_value(\mathcal{G})$ denote the value of the $\mathcal{G}$ when the strategy of Player 0 is restricted to strategies that ensure no loss.

Recall that in Theorem 2 we showed that deciding whether $value(\mathcal{G}) \geq 1$ is Σ_2^P-complete. We start with some good news and show that if Player 0 is restricted to no-loss strategies, the problem can be solved in polynomial time. In order to prove this, we first define *reachability games*.

Consider a graph $\langle V, E\rangle$, source and target vertices $s, t \in V$, and a partition $V_0 \cup V_1$ of V. The reachability game on $\langle V_0, V_1, E, s, t\rangle$ is played between Player 0 and Player 1 as follows. Initially, a token is placed on s. Then, in each step, if the token is placed on a vertex $u \in V_0$ (respectively, $u \in V_1$), then Player 0 (respectively, Player 1) chooses a successor vertex v, namely v such that $\langle u, v\rangle \in E$, and moves the token to v. A strategy of Player 0 (respectively Player 1) maps each vertex $u \in V_0$ (respectively, $u \in V_1$) that is neither t nor a sink to a successor vertex v. A pair of strategies, α for Player 0 and β for Player 1, induces a unique path, denoted $outcome(\alpha, \beta)$, to be traversed by the token when the players follow their strategies. When the graph is acyclic, the path is finite and Player 0 wins if the path reaches t. Otherwise, the path reaches a sink and Player 1 wins. A *winning strategy* for Player 0 is a strategy α such that for every strategy β of Player 1, the path $outcome(\alpha, \beta)$ reaches t. it is well known that the problem of deciding the winner in a reachability game can be solved in linear time and is PTIME-complete [23].

▶ **Theorem 5.** *Consider a flow game $\mathcal{G}$. Deciding whether $nl\_value(\mathcal{G}) \geq 1$ can be done in linear time and is PTIME-complete.*

Proof. We reduce the problem of deciding whether a flow game $\mathcal{G} = \langle V_0, V_1, E, c, s, t\rangle$ satisfies the requirement $nl\_value(\mathcal{G}) \geq 1$ to the problem of deciding the reachability game $\mathcal{G}' = \langle V_0, V_1, E, s, t\rangle$, and reduce the reachability game for $\mathcal{G}'$ to the problem of deciding whether a flow game $\mathcal{G} = \langle V_0, V_1, E, c, s, t\rangle$ in which all capacities are 1 satisfies $nl\_value(\mathcal{G}) \geq 1$. The upper and lower bounds then follow from known complexity of reachability games. In both reductions, a no-loss strategy for Player 0 in $\mathcal{G}$ is related with a strategy for Player 0 in $\mathcal{G}'$. Essentially, the successor vertex to which the flow of 1 is going to be directed in a no-loss strategy is the successor vertex to which the token is going to be moved, and vice versa. The details of the reductions can be found in Appendix A.3. ◀

While deciding whether a positive outcome can be achieved by a no-loss strategy can be done in polynomial time, calculating the no-loss value of the game is not easier than in the general case. Formally, in the *no-loss flow game problem* (NLFG problem, for short) the input is a flow game $\mathcal{G}$ and a threshold $\gamma \in \mathbb{N}$, and the goal is to decide whether $nl\_value(\mathcal{G}) \geq \gamma$.

▶ **Theorem 6.** *The NLFG problem is Σ_2^P-complete.*

Proof. The upper bound is similar to the case where loss is allowed. Indeed, given a strategy α for Player 0, the problem of checking whether there is a strategy β for Player 1 such that there is loss or that $outcome(\alpha, \beta) < \gamma$ is in NP, which implies membership in Σ_2^P for the NLFG problem.

For the lower bound, we show that the reduction from QBF_2 described in the proof of Theorem 2 can be manipulated to address no-loss strategies. Essentially, this is done by replacing all sinks by a single vertex in which loss is possible, but may be avoided when the formula is satisfiable. The details of the reduction are described in Appendix A.4. ◀

5 Non-Integral Flow Games

Recall that the capacities in the network are integral and that a policy for a vertex can assign only integral flows. Integral-flow games arise naturally in settings in which the objects we transfer along the network cannot be partitioned into fractions, as is the case with cars, packets, and more. Moreover, sometimes, as in cases of messages or other information packages, objects can be partitions up to a known granularity. It is easy to see that by multiplying all capacities by factor γ and solving an integer-flow game in the obtained game, we get a solution that involves strategies with fractions of $\frac{1}{\gamma}$ in the original game. Sometimes, however, as in the case of liquids, flow can be partitioned arbitrarily. In the traditional one-player setting, it is known that when the capacities are integral, then there exists an integral maximum flows. In this section we study an extension of the setting and allow strategies to use flows in $\mathbb{R}$. We show that, interestingly, the game setting makes these strategies stronger, in the sense that partitioning outgoing flows into reals may enlarge the value of the game. Moreover, the gain cannot be bounded by a constant.

Consider a flow game $\mathcal{G} = \langle V_0, V_1, E, c, s, t \rangle$. Let $\mathbb{R}_+$ denote the set of non-negative real numbers. A *non-integral policy* for a vertex $u \neq s$ is a function $f_u : \mathbb{R}_+ \to \mathbb{R}_+^{E_u}$ such that for every flow $x \in \mathbb{R}_+$ and edge $e \in E_u$, we have $f_u(x)(e) \leq c(e)$ and $\sum_{e \in E_u} f_u(x)(e) = \min\{x, \sum_{e \in E_u} c(e)\}$. Likewise, a non-integral policy f_s for the source s is a function in $\mathbb{R}_+^{E_s}$. Thus, non-integral policies allow non-integral flows. We say that a strategy is a *non-integral strategy* if it contains non-integral policies. We use $ni\_value(\mathcal{G})$ to denote the non-integral value of $\mathcal{G}$, namely the value when the players are allowed to use non-integral strategies. We first show that Player 0 can benefit from using non-integral strategies:

▶ **Theorem 7.** *There is a flow game $\mathcal{G}$ such that an optimal strategy for Player 0 in $\mathcal{G}$ must be non-integral even when Player 1 is restricted to integral strategies. Thus, $ni\_value(\mathcal{G}) > value(\mathcal{G})$.*

Proof. Consider the flow game appearing in Figure 3 on the left. Consider the non-integral strategy α that assigns an outgoing flow of 2 to $\langle s, u \rangle$ and partitions the incoming flow to v_0 equally between $\langle v_0, v_1 \rangle$ and $\langle v_0, v_2 \rangle$. It is not hard to see that for every strategy β for Player 1, we have that $outcome(\alpha, \beta) \geq 1.5$. Indeed, the sum of the incoming flows to v_1 and v_2 is 2, implying that there is loss of flow in at most one of these vertices. Since the flow from v_0 to both v_1 and v_2 is at most 0.5, then the loss is at most 0.5. On the other hand, if Player 0 is restricted to integral strategies, then an outcome of more than 1 cannot be ensured, as once Player 0 has chosen his strategy and directs 2 units to u, Player 1 can cause a loss of 1 by choosing a policy in u that directs 1 to v_0 and directs 1 to the vertex (v_1 or v_2) to which Player 0 directs the incoming flow of 1 to v_0. ◀

In the *non-integral flow game problem* (NIFG problem, for short) the input is a flow game $\mathcal{G}$ and a threshold $\gamma \in \mathbb{R}_+$, and the goal is to decide whether $ni\_value(\mathcal{G}) \geq \gamma$. We consider also the *no-loss non-integral flow game problem* (NLNIFG, somewhat shorter), where we consider non-integral strategies that also ensure no loss of flow.

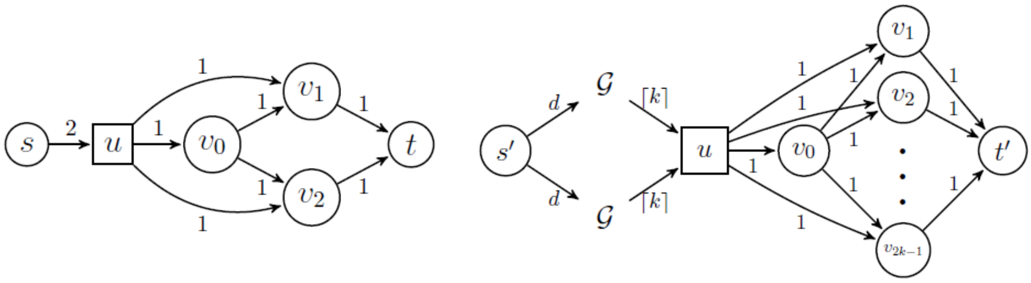


Figure 3 A FG in which Player 0 benefits from non-integral strategies (left) and its amplification (right).

The reduction from QBF_2 described in Theorem 2 holds also for these new settings. Indeed, by Lemma 3, the optimal strategies of the players would result in integral input to $\mathcal{G}_\psi$. Hence, the following theorem.

▶ **Theorem 8.** *The NIFG and NLNIFG problems are Σ_2^P-hard. The NIFG problem is also Σ_2^P-hard to approximate within any multiplicative factor.*

Theorem 8 only gives a lower bound for the problem. The upper bound for the FG problem relied on the fact that integer strategies have a polynomial representation, which cannot be applied in the NIFG problem. Moreover, since reals are second-order creatures, the problem may be undecidable. We leave the decidability problem open and describe, in Section 6, a decidable variant. A natural question is whether an algorithm that solves the FG problem (for integral strategies) approximates the solution for the case of non-integral strategies. As we show in the following theorem, such an approximation cannot be bounded by a constant factor.

▶ **Theorem 9.** *For all $n > 1$, there is a flow game $\mathcal{G}_n$ s.t. $value(\mathcal{G}_n) = 1$ and $ni\_value(\mathcal{G}_n) \geq n$.*

Proof. We amplify the construction used in the proof of Theorem 7. Recall that there, we showed that the flow game in Figure 3 on the left has value 1 and has non-integral value 1.5. The amplification involves two ideas: we parameterize the branching degree of v_0 and we use a recursion that involves sub-games for which a smaller ratio between the value and the non-integral value is known.

Let $\mathcal{G}$ be a flow game such that $value(\mathcal{G}) = 1$ and $ni\_value(\mathcal{G}) = k$, for $k > 1$, and assume that $2k$ is an integer. For example, in the game presented in Figure 3 on the left, we have $k = 1.5$. We construct a flow game $\mathcal{G}'$ such that $value(\mathcal{G}') = 1$ and $ni\_value(\mathcal{G}') = 2k - 1$. By repeating the construction logarithmically many times, we obtain the required flow game $\mathcal{G}_n$.

Consider the game $\mathcal{G}$. Let d_1 and d_2 be the outgoing flow from the source in an optimal strategy and a non-integral strategy of Player 0 in $\mathcal{G}$, respectively. Let d be an integer greater than d_1 and d_2. We define the new game $\mathcal{G}'$ as shown in Figure 3 on the right. The game $\mathcal{G}'$ contains $\mathcal{G}$ twice as a sub-game: the incoming edge to $\mathcal{G}$ enters its source and the outgoing edge from $\mathcal{G}$ leaves its target. In a flow game on $\mathcal{G}'$, the maximum flow to the vertex u that Player 0 can ensure with an integral strategy is 2. Hence, $value(\mathcal{G}') = 1$. Allowing non-integral strategies, the maximum flow to the vertex u that Player 0 can ensure is $2k$, making $ni\_value(\mathcal{G}') = 2k - 1$. ◀

6 Dynamic Flow Games

In our definition of flow games, the players choose their strategies before the game starts. The universal quantification on the strategies of Player 1 implies that this is equivalent to a scenario in which Player 0 chooses a strategy and then Player 1 chooses a strategy. In dynamic flow games we refer to the way information flows in the graph and let the players choose policies in a vertex only after flow travels in the subgraph from which the vertex is reachable. Since the graph is acyclic, this is well defined.

Formally, we first order the graph in a topological ordering such that V_0 vertices appear, whenever possible, before V_1 vertices. That is, we fix a topological ordering $\leq$ such that if there is neither a path from $u \in V_0$ to $v \in V_1$ nor from v to u then $u \leq v$. Now, Player 0 chooses a flow for each outgoing edge of s. Then, we follow the order $\leq$ and for each vertex u, the player that controls u directs the incoming flow to the outgoing edges. Thus, instead of choosing a strategy that describes the behavior for each possible incoming flow for each vertex u, the player just chooses how to direct the incoming flow, and he has information on the flow in edges whose source precedes u in $\leq$. The value of the game is the maximal flow to t that Player 0 can ensure.

Note that the dynamic setting is not equivalent to the non-dynamic (original) one. For example, the value of the flow game in Figure 3 is 1 (and is 1.5 when we allow non-integral strategies), whereas the dynamic setting leads to a value of 2. The flow game in Figure 3 suggests that the information about "earlier" flow helps Player 0 to use integral flows. Indeed, coming to decide the flow from vertex v_0, Player 0 already known which of v_1 and v_2 have already received 1 flow unit. As we show in Appendix A.5, however, the information may not be sufficient in general, and Player 0 can benefit from using non-integral strategies.

The Σ_2^P lower bounds described in Sections 3 and 4 apply also for the dynamic setting. On the positive side, the dynamic setting enables a formulation of the problem by means of the first-order theories of integral addition or real addition, making the related decision problems decidable for both integral and real flows [15, 14].

7 Discussion and Future Work

Today's computing environment involves parties that should be considered adversarial. This calls for a re-examination of classical algorithmic problems. We introduced and studied flow games, which capture settings in which the authority can control the flow only in part of the vertices in a flow network. Below we discuss possible extensions of our work as well as open problems we have left for future research.

7.1 Extensions

Evacuation An evacuation flow game is $\mathcal{G} = \langle \{s\}, V_1, E, c, s, t \rangle$. Thus, all vertices except for the source belong to Player 1.[4] As explained in Section 1, an evacuation flow game corresponds to scenarios in which the authority has no control on how flow travels on the network [25]. Solving the FG problem in an evacuation flow game amounts to finding a strategy for Player 1 that minimizes the flow.

Beyond the clear practical applications, it is interesting to study the computational aspects of the evacuation problem. As it refers only to one player, and thus involves no alternation of quantification, there is hope the problem can be solved in polynomial time, possibly by a reduction to a prioritized variant of the maximal-flow problem. The latter is strongly related to several weighted variants of the max-flow problem. In particular, polynomial solutions are known for the min-cost max-flow problem [34] and to max-flow with lexicographically ordered edges problem [26].

A Classical Algorithmic Game-Theory Examination Beyond the questions we studied for FGs, the game setting calls for a classical algorithmic game-theory examination: existence of a Nash equilibrium, stability inefficiency, and many more. We plan to study them for *multiplayer flow*

[4] Recall that we define a flow game with $s \in V_0$. Alternatively, we could have defined an evacuation flow game as one in which all vertices are in V_1 and parameterize the game with an initial flow that enters s. Indeed, saturating the edges from s is a dominating strategy for Player 0, so Player 0 does not really take decisions even when $s \in V_0$.

games, where the vertices of the network are partitioned between some $k > 1$ players, there are k target vertices, and the objective of each player is to direct as much flow as possible to her target vertex. For example, in a communication network controlled by several companies (that is, vertices correspond to routers owned by the companies), each company wants to maximize the number of messages from its source to target vertices, and we want to find the best Nash equilibria or design networks for which this equilibria is close to the social optimum.

Partially Specified Networks In some cases, a strategy is known for some vertices and we need to find an optimal strategy for the others. For example, in hybrid networks, where some vertices are new SDN routers whose behaviour can be controlled, and the other vertices run known traditional routing protocols (e.g., route via the shortest path) [3]. As another example, consider the case where each time that a driver reaches an uncontrolled vertex, it randomly chooses an outgoing edge with free capacity. In this case, we need to find a strategy that maximizes the expected flow to the target.

7.2 Open Problems

Recall that a maximum flow in a flow game may involve non-integrals even when all capacities are integrals. While we settle the complexity of flow games for strategies that only use integrals, decidability for the setting in which strategies may use non-integrals is open. One approach to obtain decidability and analyze flow games with non-integral strategies is to approximate their value with integral strategies. While such an approximation cannot be bounded by a constant multiplicative factor (Theorem 9), we conjecture that it can be bounded by a linear additive factor. In particular, we conjecture that for every flow game $\mathcal{G}$, we have that $ni\_value(\mathcal{G}_n) \leq value(\mathcal{G}_n) + |E|$, where the intuition is that a non-integral flow in $\mathcal{G}$ should not exceed flow in the game obtained from $\mathcal{G}$ by increasing all capacities by 1. Another approach would be to find a granularity to which it would be sufficient to break an integral flow. Our efforts in this direction so far reveal some counter-intuitive facts. For example, flow games are not *local monotonic*: increasing the flow that enters a vertex may cause a decrease in the flow directed by an optimal strategy to one of its outgoing edges. Also, the required granularity is not continuous, in the following sense: for every $k \geq 1$, we can generate a FG such that an optimal strategy that can break the flow into the fractions $\frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{k-1}$ achieves the same flow achieved by an optimal strategy that only uses integrals, yet breaking the flow into the fraction $\frac{1}{k}$ improves this flow. Also, while we know that non-integral strategies may be better than integral ones, we do not know whether strategies that use rational numbers are as good as strategies that use real ones.

Acknowledgment We thank David Hay for helpful discussions on software-defined networks.

References

1 S. Agarwal, M. S. Kodialam, and T. V. Lakshman. Traffic engineering in software defined networks. In *Proc. 32nd IEEE International Conference on Computer Communications*, pages 2211–2219, 2013.

2 R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network flows: Theory, algorithms, and applications*. Prentice Hall Englewood Cliffs, 1993.

3 B. Aminof, O. Kupferman, and R. Lampert. Rigorous approximated determinization of weighted automata. *Theoretical Computer Science*, 480:104–117, 2013.

4 E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. *SIAM J. Comput.*, 38(4):1602–1623, 2008.

5 T. Bartnicki, J. A. Grytczuk, H. A. Kierstead, and X. Zhu. The map coloring game. *American-Mathematical Monthly*, 114:793–803, 2007.

6 A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, 1981.

7 S.A. Cook. Path systems and language recognition. In *Proc. 2nd ACM Symp. on Theory of Computing*, pages 70–72, 1970.

8 S.A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symp. on Theory of Computing*, pages 151–158, 1971.

9 T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1990.

10 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.

11 J. Edmonds and R.M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

12 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.

13 J. Feigenbaum, C.H. Papadimitriou, R. Sami, and S. Shenker. A BGP-based mechanism for lowest-cost routing. *Distributed Computing*, 18(1):61–72, 2005.

14 J. Ferrante and C. Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.

15 M.J. Fischer and M.O. Rabin. Super-exponential complexity of presburger arithmetic. pages 27–41, 1974.

16 L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

17 L.R. Ford and D.R. Fulkerson. *Flows in networks*. Princeton Univ. Press, Princeton, 1962.

18 Z. Füredi, D. Reimer, and A. Seress. Triangle-free game and extremal graph problems. *Congressus Numerantium*, 82:123–128, 1991.

19 A.V. Goldberg, É. Tardos, and R.E. Tarjan. Network flow algorithms. Technical report, DTIC Document, 1989.

20 A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, 1988.

21 D. Hefetz, M. Krivelevich, A. Naor, and M. Stojaković. On saturation games. *European Journal of Combinatorics*, 41:315–335, 2016.

22 D. Hefetz, O. Kupferman, A. Lellouche, and G. Vardi. Spanning-tree games. To appear, 2018.

23 N. Immerman. Number of quantifiers is better than number of tape cells. *Journal of Computer and Systems Science*, 22(3):384–406, 1981.

24 E. Kalai and E. Zemel. Totally balanced games and games of flow. *Mathematics of Operations Research*, 7(3):476–478, 1982.

25 S. Keren, A. Gal, and E. Karpas. Goal recognition design for non optimal agents. In *Proc. 29th AAAI conference*, pages 3298–3304, 2015.

26 D. Kozen. Lexicographic flow. Technical Report http://hdl.handle.net/1813/13018, Computing and Information Science, Cornell University, June 2009.

27 O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, 1985.

28 N. Nisan, T. Roughgarden, E. Tardos, and V.V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

29 M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

30 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

31 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.

32 B.L. Schwartz. Possible winners in partially completed tournaments. *SIAM Review*, 8(3):302–308, 1966.

33 L.J. Stockmeyer. On the combinational complexity of certain symmetric boolean functions. *Mathematical Systems Theory*, 10:323–336, 1977.

34 É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–255, 1985.

35 S. Vissicchio, L. Vanbever, and O. Bonaventure. Opportunities and research challenges of hybrid software defined networks. *Computer Communication Review*, 44(2):70–75, 2014.

A Examples and Proofs

A.1 Details about the reduction in Theorem 2

We start with an example of the game $\mathcal{G}_\theta$ for the case $\psi = (x \vee y) \wedge (\bar{x} \vee \bar{y})$.

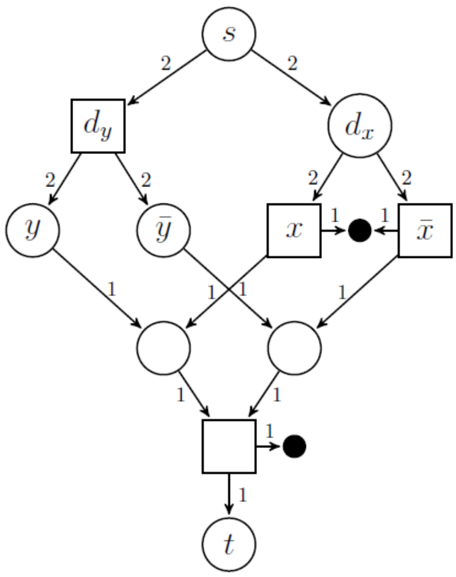


Figure 4 The game $\mathcal{G}_\theta$ for $\psi = (x \vee y) \wedge (\bar{x} \vee \bar{y})$

We proceed to prove that the value of $\mathcal{G}_\theta$ is 1 if θ holds and is 0 otherwise. Assume first that θ holds. Let π be an assignment for X such that for every assignment for Y, the formula ψ holds. Player 0 can use the following strategy: For every x_i such that $\pi(x_i) = 1$, Player 0 assigns to the vertex x_i an incoming flow of $2k$ (and an incoming flow of 0 to $\bar{x}_i$); for every x_i such that $\pi(x_i) = 0$, Player 0 assigns to the vertex $\bar{x}_i$ an incoming flow of $2k$ (and an incoming flow of 0 to x_i). Thus, the input flows to $\mathcal{G}_\psi$ for the variables X is consistent with the assignment π. Also, Player 0 assigns a flow of $2k$ to the vertices $d_{y_1}, \ldots, d_{y_m}$. Therefore, for each i, either y_i or $\bar{y}_i$ must have an incoming flow of at least k. According to Lemma 3, an optimal strategy for Player 1 is to choose to assign a flow of $2k$ either to y_i or to $\bar{y}_i$. In this case, the input flows to $\mathcal{G}_\psi$ for the variables in Y is consistent with some assignment τ to these variables. The outcome of $\mathcal{G}_\theta$ for these strategies is the value of ψ for the assignments π and τ, which is 1. Therefore, Player 0 has a strategy such that for an optimal strategy of Player 1 the outcome of $\mathcal{G}_\theta$ is 1, and hence the value of $\mathcal{G}_\theta$ is 1.

Now assume that θ does not hold, we show that the value of $\mathcal{G}$ is 0. According to the assumption, for every assignment to X there is an assignment to Y such that ψ does not hold. According to Lemma 3, an optimal strategy for Player 1 in the vertices $X \cup \bar{X}$ is to assign as much flow as possible toward the sinks, and therefore for every $1 \leq i \leq n$, either x_i or $\bar{x}_i$ would have outgoing flow 0 to $\mathcal{G}_\psi$. Thus, by Lemma 3 (1) an optimal strategy for Player 0 is to assign, for every $1 \leq i \leq n$, an incoming flow of $2k$ either to x_i or to $\bar{x}_i$. Indeed, this would guarantee an outgoing flow of k from the chosen literal. Hence, the input flow to $\mathcal{G}_\psi$ for the variables X is consistent with some assignment π. Likewise, according to Lemma 3 (1), an optimal strategy for Player 0 is to assign a flow of $2k$ to each vertex $d_{y_i}, \ldots, d_{y_m}$. Then, for each vertex d_{y_i}, an optimal strategy for Player 1 assigns the $2k$ flow to either y_i or $\bar{y}_i$. In this case, the input flow to $\mathcal{G}_\psi$ for the variables Y is consistent with some assignment τ that Player 1 chooses. Hence, by Lemma 3 (2), the value of $\mathcal{G}_\psi$ is the value of ψ for the corresponding assignment. By our assumption, Player 1 can choose τ such that ψ does not hold for the assignments π and τ. Hence, the value of the game is 0.

A.2 A polynomial-time algorithm for bounded-degree trees

For a vertex $u \in V$, let G_u be the sub-graph of $\langle V, E\rangle$ that contains only the vertices that are reachable from u. For every $\gamma > 0$, we compute the value of the flow game induced by G_u, where the incoming flow to u is γ. We denote this value by $value_\gamma(G_u)$. This computation can be done as follows. Let $\{v_1, \ldots, v_k\}$ be the vertices such that $\langle u, v_i\rangle \in E$. If $u \in V_0$, then $value_\gamma(G_u) = \max\{\Sigma_{1\leq i\leq k} value_{\gamma_i}(G_{v_i}) :$ for every i we have $\gamma_i \leq c(u, v_i)$ and $\Sigma_{1\leq i\leq k}\gamma_i = \min\{\gamma, \Sigma_{1\leq i\leq k} c(u, v_i)\}\}$. That is, the vertex u partitions the incoming flow of γ between the graphs G_{v_i}, for $1 \leq i \leq k$, in a way that maximizes the sum of the values of the flow games on these graphs. If $u \in V_1$, then $value_\gamma(G_u)$ can be computed similarly by finding the minimum. The value of the flow game $\mathcal{G}$ is then $max_{\gamma>0}\{value_\gamma(G_s)\}$. Since the degrees of the vertices are bounded by a constant, the computation can be done with dynamic programming in polynomial time.

A.3 A proof of Theorem 5

We start with the upper bound. Consider a flow game $\mathcal{G} = \langle V_0, V_1, E, c, s, t\rangle$. Let $\mathcal{G}' = \langle V_0, V_1, E, s, t\rangle$ be a reachability game played on the network underlying $\mathcal{G}$. We claim that Player 0 has a no-loss strategy that ensures a positive outcome in $\mathcal{G}$ iff there is a winning strategy for Player 0 in a reachability game on $\mathcal{G}'$. Since reachability games can be solved in linear time, the upper bound follows.

Assume that there is a winning strategy for Player 0 in the reachability game. Let $\alpha' : V_0 \to V$ be the winning strategy of Player 0. That is, for every vertex $u \in V_0$, we have that $\alpha'(u)$ is the vertex v to which Player 0 moves the token when it reaches u. Consider the strategy α for Player 0, where for every vertex $u \in V_0 \setminus \{s\}$, the policy f_u is such that $f_u(1)(\langle u, v\rangle) = 1$ iff $\alpha'(u) = v$. That is, if the incoming flow is 1 then it is forwarded to the successor to which α' suggests to move the token in the reachability game. Also, the policy in the vertex s generates flow 1 in the outgoing edge that leads to $\alpha'(s)$. All other edges from s are assigned flow 0. Defining α as above ensures that the incoming flow in each vertex is 0 or 1. Hence, a strategy β for Player 1 in $\mathcal{G}$ defines for each vertex a single successor to which it forwards the incoming flow. Since α' is winning in $\mathcal{G}'$, we have that

$outcome(\alpha, \beta) = 1$ for all strategies β for Player 1. Note that since the capacities are greater than or equal to 1, then loss is impossible.

Assume now that Player 0 has a no-loss strategy α that ensures an outcome of 1 in $\mathcal{G}$. Consider the strategy α' for Player 0 in the reachability game $\mathcal{G}'$ that is induced by α in the expected way. That is, $\alpha'(u) = v$ for the single vertex v for which $f_u(1)(\langle u, v\rangle) = 1$ for the policy f_u of u in α, and $\alpha'(s) = v$ for the single vertex v for which $f_s(\langle s, v\rangle) = 1$. Since α is a no-loss strategy that ensures an outcome of 1, the above policies exist. Also, it is guaranteed that for every strategy β of Player 1 in $\mathcal{G}$, we have that $outcome(\alpha, \beta) = 1$, implying that α' is winning for Player 0 in $\mathcal{G}'$.

For the lower bound, we reduce reachability games, known to be PTIME-hard [23], into our problem. Given a reachability game $\mathcal{G}' = \langle V_0, V_1, E, s, t\rangle$, consider the flow game $\mathcal{G} = \langle V_0, V_1, E, c, s, t\rangle$ in which all capacities are 1. As in the other direction, it is not hard to prove that Player 0 has a no-loss strategy that ensures a positive outcome in $\mathcal{G}$ iff there is a winning strategy for Player 0 in the reachability game on $\mathcal{G}'$.

A.4 A proof of the lower bound in Theorem 6

We show that the reduction from QBF_2 described in the proof of Theorem 2 can be manipulated to address no-loss strategies. As there, let $\theta = \exists x_1 \ldots \exists x_n \forall y_1 \ldots \forall y_m \psi$. 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 $\bar{Z} = \bar{X} \cup \bar{Y}$. Recall that in the proof of Theorem 2 we constructed a flow game $\mathcal{G}_\theta = \langle V_0, V_1, E, c, s, t\rangle$ that contains an external-source flow game $\mathcal{G}_\psi$ as a sub-game. We construct a new flow game $\mathcal{G}'_\theta = \langle V'_0, V_1, E', c', s, t'\rangle$ by changing $\mathcal{G}_\theta$ as follows. First, we add new vertices t_1 and t'. The vertex t' has two incoming edges – from t with capacity 1, and from t_1 with capacity $2k(n+m) - 1$. Now, for the vertices $X \cup \bar{X}$ we remove the outgoing edges to the sinks. Then, for the vertices $Z \cup \bar{Z}$ we add outgoing edges with capacity k to the vertex t_1. That is, each of these vertices has an outgoing capacity of k toward $\mathcal{G}_\psi$ and an outgoing capacity of k toward t_1. Also, for the V_1 vertices of $\mathcal{G}_\psi$ (the vertices that represent AND gates) we remove the outgoing edges to the sinks. Then, for each vertex in $\mathcal{G}_\psi$ (both the V_0 and the V_1 vertices) we add an outgoing edge with capacity 1 to the vertex t_1. Thus, each vertex in $\mathcal{G}_\psi$ now has both incoming and outgoing capacities of 2. In the obtained flow game $\mathcal{G}'_\theta$ we have $V'_0 = V_0 \cup \{t_1, t'\}$.

Note that in $\mathcal{G}'_\theta$ for every vertex $v \neq t_1$ the outgoing capacity is greater than or equal to the incoming capacity. Hence, the only vertex where loss is possible is t_1. Let O_s be an outgoing flow from s, let $I_{t'}$ be an incoming flow to t', and let l_{t_1} be the loss in t_1. Then, $I_{t'} = O_s - l_{t_1}$. It is easy to see that if the outgoing flow from s is $2k(n+m) - 1$, then the outcome must be $2k(n+m) - 1$, since loss is possible only in t_1 and its outgoing capacity is $2k(n+m) - 1$. When the outgoing flow from s is $2k(n+m)$, note that if a flow of 1 reaches t then the flow that reaches t_1 is at most $2k(n+m) - 1$ and then loss is impossible; conversely, if a flow of less than 1 reaches t then the flow that reaches t_1 is greater than $2k(n+m) - 1$ and then there is loss. Intuitively, the goal of Player 0 is to direct a flow of 1 to t and the goal of Player 1 is to prevent it. We show that θ holds iff $nl\_value(\mathcal{G}_\theta) = 2k(n+m)$.

Assume that θ holds. In the proof of Theorem 2, we showed that in this case Player 0 has a strategy α in $\mathcal{G}_\theta$ that ensures an incoming flow of 1 to the vertex t. In α, the outgoing flow from s is $2k(n+m)$. Note that α induces a strategy α' in $\mathcal{G}'_\theta$ that ensures an incoming flow of 1 to t. Thus, α' ensures that a flow of $2k(n+m)$ reaches t' and that there is no loss.

Now, assume that θ does not hold. Also, assume that α' is a strategy of Player 0 in $\mathcal{G}'_\theta$ that ensures that a flow of $2k(n+m)$ reaches t'. Thus, in α' there is an outgoing flow of $2k(n+m)$ from s, and it ensures that a flow of 1 reaches t. From Lemma 3 we can assume that in every vertex $v \in V_0$ the policy of v according to α' does not direct flow toward t_1 unless the outgoing capacity from v to vertices in $\mathcal{G}_\psi$ is filled. Therefore, α' induces a strategy α for Player 0 in $\mathcal{G}_\theta$ that ensures a flow of 1 to t. However, in the proof of Theorem 2, we showed that for every strategy of Player 0 in $\mathcal{G}_\theta$ there is a strategy of Player 1 such that the flow that reaches t is 0. Thus, we have reached a contradiction.

A.5 Non-integral strategies in the dynamic setting

In Figure 5 we show a flow game where Player 0 can benefit from using non-integral strategies in the dynamic setting. According to the dynamic setting the order in which the policies for the vertices are chosen is s,v_0 and then u. If Player 0 is restricted to integer strategies then an outcome of more than 1 cannot be ensured, since Player 1 can choose to direct the outgoing flow from u to the vertex to which Player 0 directs the outgoing flow from v_0. However, if Player 0 partitions the outgoing flow of 1 from v_0 equally between v_1 and v_2 then the maximum possible loss is 0.5 and thus the outcome is 1.5.

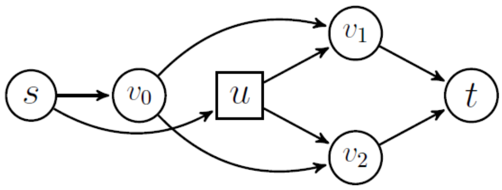


Figure 5 A flow game in which Player 0 benefits from using non-integral strategies in the dynamic setting. The capacities are all 1.