EPiC Series in Computing

Volume 57, 2018, Pages 18–25

LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Playing with the Maximum-Flow Problem *

Orna Kupferman

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 has been extensively used in order to optimize the performance of networks in numerous application areas. The definition of the problem 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 survey recent studies on *flow games*, which capture settings in which the vertices of the network are owned by different and selfish entities. We start with the case of two players, MAX (the authority), which aims at maximizing the flow, and MIN (the hostile environment), which aims at minimizing the flow. We argue that such flow games capture many modern settings, such as partially-controlled pipe or road systems or hybrid software-defined communication networks. We then continue to the special case where all vertices are owned by MIN. This case captures *evacuation* scenarios, where the goal is to maximize the flow that is guaranteed to travel in the most unfortunate routing decisions. Finally, we study the general case, of *multiple players*, each with her own target vertex. In all settings, we study the problems of finding the maximal flows, optimal strategies for the players, as well as stability and equilibrium inefficiency in the case of multi-player games. We discuss additional variants and their applications, and point to several interesting open problems.

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 [7, 14]. The problem was first formulated and solved in the 1950's [12, 13]. It has attracted much research on improved algorithms [9, 8, 15] 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

*Based on joint work with Shibashis Guha, Gal Vardi, and Moshe Y. Vardi [20, 16, 19].

to model settings with an abstract network, as in the case of elimination in partially completed tournaments [27] 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, minimum path cover, maximum edge-disjoint or vertex-disjoint path, and many more [7, 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 [29], networks with multiple source and target vertices [10], 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, 30].

The above applications suggest that the maximal-flow problem should be revisited, taking into account the *game-theoretic* nature of the setting. We survey recent studies on *flow games*,[1] which capture settings in which the vertices of the network are owned by different entities. We start with two-players flow games, introduced in [20]. There, the vertices in the network are partitioned between two players, MAX and MIN. The player MAX corresponds to the network authority, whose goal is to maximize the flow that reaches the target, while MIN corresponds to the hostile environment. A strategy for a player advices her 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*, depending on the story behind the application) 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 MAX is to maximize the flow that enters the target t, no matter how MIN plays. Thus, it is a *Stackelberg game* where MAX is the leader [24]. 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 MAX, and in which the "flow conservation" property is respected in all vertices. Indeed, in the game setting, MIN may cause flow to get stuck by directing the flow to vertices whose outgoing capacity is smaller than the incoming flow.

Example 1. *Consider the flow game in Figure 1 below. We represent vertices of* MAX *by circles and vertices of* MIN *by squares. Sinks, namely vertices other than the target that do not have*

[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., [17]).

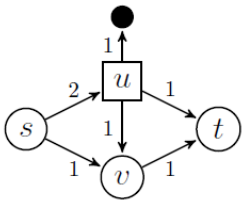

Figure 1: A flow game.

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. We can view the network as a road system, where the vertex* u*, which belongs to* MIN*, 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* MAX *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* MIN *can direct it to the sink and to vertex* v*,* MAX *does not have a strategy that ensures a flow of more than* 1*. Hence, the maximum flow that* MAX *can ensure is* 1*.* □

In essence, [20] lifts 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*) [5]. In complexity theory, this is the transition from non-deterministic computation to alternating computation [4]. In logic, this is the transition from Boolean satisfiability [6] to quantified Boolean satisfiability [28]. In temporal reasoning, this is the transition from satisfiability [21] to temporal synthesis [26]. 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. [23], 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.

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, yet 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

[2]Different aspects of networks have already been extensively studied from the perspectives of algorithmic game theory. This includes, for example, network formation games [3] or incentive issues in interdomain routing and the BGP protocol [11]. We are the first, however, to consider the maximal-flow problem from this perspective.

major theme of this work.

The study in [20] is of *acyclic* game networks. There, given strategies for the players, it is possible to calculate the flow by following a topological ordering of the vertices. It is shown in [20] 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. The study in [20] includes 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. Other important variants studied in [20] refer to *no-loss flow* and *non-integral flow*. For the first, 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 MAX such that for every strategy of MIN, 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, giving rise to the problem of finding the maximal flow that the authority can transfer while ensuring no loss. As good news, it is shown in [20] 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.

For the second variant, recall that 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 [12]. It is shown in [20] that, interestingly, the game setting makes strategies that use non-integral flow stronger: partitioning outgoing flows into non-integers may increase the flow that MAX can ensure. Moreover, the gain cannot be bounded by a constant. The problem of finding the flow that MAX can ensure when using non-integral strategies is left open in [20]. Hardness in Σ_2^P holds also for this setting. Yet, as real numbers are second-order creatures, the problem may be undecidable.

We continue to a study of the special case of networks in which the authority has no control [19]. 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 amount of flow that is guaranteed to reach the target in the worst scenario. It is not hard to see that this corresponds to a flow game in which all vertices are owned by MIN. Formally, we define the *unfortunate flow value* of a network as the amount of flow that is guaranteed to reaches the target t no matter what the strategy of MIN is, assuming all the outgoing edges from the source s are saturated. Thus, MIN cannot restrict flow form the source, yet the most unfortunate routing decisions are taken in junctions. In the *unfortunate-flow problem*, we want to find the unfortunate flow value of a given network.

Example 2. *Consider the flow network $\mathcal{G}$ appearing in Figure 2 (a). A maximum flow in $\mathcal{G}$ has value 8, attained, for example, with the flow in (b). Since the minimal cut in $\mathcal{G}$ has capacity*

8*, this is indeed a maximum flow. An unfortunate flow for* $\mathcal{G}$ *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.* □

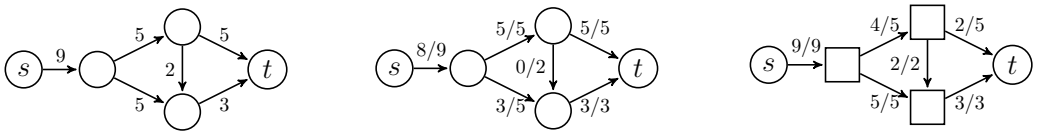

Figure 2: A flow network $\mathcal{G}$, and flows that attain its maximum-flow and unfortunate-flow values.

In [19], we introduce and study the unfortunate-flow problem. We start with the complexity of the 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 [13], 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 in a network with n vertices, m edges, and h funnels can be solved in time $O(2^h \cdot (m^2 \log n + mn \log^2 n))$. 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 [2], 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. 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.

In [16], we introduce and study *multi-player flow games* (MFGs, for short).[3] Essentially, the vertices of an MFG are partitioned among the players, and a player that owns a vertex directs the flow that reaches it. Each player has a different target vertex, and the objective of the players is to maximize the flow that reaches their target vertices. As in flow games, a strategy for a player advises her how to direct flow that enters vertices under her control. As there, we assume that the network is *acyclic*. Then, given strategies for the players, it is possible to calculate the flow by following a topological ordering of the vertices.

Example 3. *Consider the MFG* $\mathcal{G}$ *appearing in Figure 3. The game is played between two players. The vertices of Player* 1 *are circles, and those of Player* 2 *are squares. An initial flow of* 2 *arrives to vertex* s*, and the targets of the players are* t_1 *and* t_2*. We can view* $\mathcal{G}$ *as a communication network with routers operated by companies with different targets. Unless outgoing channels are filled, a router does not drop packets that reach it, and it can direct the packets however it chooses.*

[3] 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., [17]).

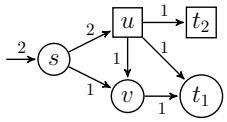

Figure 3: The MFG $\mathcal{G}$

No matter how Player 1 *directs the flow in vertex* s*, a flow of at least* 1 *reaches* t_1*. Indeed, if Player* 1 *directs* 1 *to* v*, then it continues from there to* t_1*. Also, if Player* 1 *directs* 2 *to* u*, then Player* 2 *directs at most* 1 *to* t_2*, and directs the rest, namely at least* 1*, to* v *(from where it is directed to* t_1*) or to* t_1*. Note that Player* 2 *may direct no flow to* t_2*, in which case a flow of* 2 *reaches* t_1*, yet Player* 2 *has no incentive to do so. Moreover, if Player* 1 *directs* 1 *to* v *and* 1 *to* u*, and Player* 2 *directs* 1 *from* u *to* v*, then flow gets lost in* v*, as the capacity of edges outgoing from* v *is only* 1*.* □

Tthe questions on MFGs that are studied in [16] originate from its game-theoretic nature, and are very different from those studied in [20].

In order to describe the contribution, we first need some notations. A *profile* in an MFG is a tuple of strategies, one for each player. Primary questions about games in traditional game-theory applications concern their stability. The most common criterion for stability is the existence of a *Nash equilibrium* (NE, for short) [22]: a profile in which no (single) player can benefit from unilaterally changing her strategy. It is well known that decentralized decision-making may lead to stable profiles that are sub-optimal from the point of view of society as a whole. Formally, a profile is a *social optimum* (SO, for short) if it maximizes the flow to all target vertices together. An SO thus corresponds to a maximum flow in a network obtained from the MFG by adding a source vertex in which the initial flow is generated, and a target vertex to which all target vertices are connected. The inefficiency incurred due to selfish behavior of the players is measured by the *price of anarchy* (PoA) [18, 25] and *price of stability* (PoS) [3] measures. The PoA is the worst-case inefficiency of an NE (that is, the ratio between the flow in an SO and in a worst NE, namely one in which minimum flow reaches all targets). The PoS is the best-case inefficiency of a Nash equilibrium (that is, the ratio between the flow in an SO and a best NE). Another important question in game-theory applications is that of finding a *best-response* move, namely a strategy that maximizes the utility of a given player (that is, the flow to her target, in the case of MFGs), given the strategies of the other players. The absence of regulation by some central authority is a driving theme of *algorithmic game theory*, cf. [23], inspired by the open nature of today's computing environments.

We start with some bad news about the stability of MFGs. We show that there are simple (in fact, two-player) MFGs in which no NE exists. Moreover, the PoA and even the PoS of MFGs are unbounded. That is, for every threshold $x \geq 1$, there is an MFG $\mathcal{G}_x$ such that the SO in $\mathcal{G}_x$ is x (that is, when cooperating, the players can direct x units of flow to their targets), whereas a best NE in $\mathcal{G}_x$ is 1 (that is, in all stable profiles, only 1 flow unit reaches a target vertex). Also, the problem of deciding whether a given MFG has an NE is Σ_2^P-complete, which essentially suggests that we have to go over all possible profiles and deviations from them. We continue with the best-response problem and show that it is NP-complete. The high complexity is not surprising, and corresponds to the known computational price when moving from a nondeterministic setting to a game-based one, for example the increase from PSPACE to 2EXPTIME when moving from temporal satisfiability [21] to temporal realizability [26].

We continue with some good news and consider a variant of MFGs in which flow may be dropped (MFGD, for short). Thus, an owner of a vertex may choose not to direct some of the

incoming flow. In particular, when Player i owns a vertex from which her target cannot be reached, then she has no incentive not to drop the flow. We show that, surprisingly, while this model seems to incentivize the above selfish behavior, it is actually stable, and with no stability inefficiency. Thus, MFGDs always have an NE, and their PoS is 1. Moreover, such an NE that is also an SO can be found in polynomial time. Our algorithm is based on a careful choice of *augmenting paths* in the Ford-Fulkenson method [13], chosen in a way that guarantees that no player has an incentive to deviate from the profile that induces the maximum flow found by the algorithm. We show that this careful choice is acute, as the PoA of MFGs with drops is unbounded.

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] 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.

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

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

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

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

[8] 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.

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

[10] 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.

[11] 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.

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

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

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

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

[16] S. Guha, O. Kupferman, and G. Vardi. Multi-player flow games. In *Proc. 17th International Conference on Autonomous Agents and Multiagent Systems*, pages 104–112, 2018.

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

[18] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.

[19] O. Kupferman and G. Vardi. The unfortunate-flow problem. In *Proc. 45th Int. Colloq. on Automata, Languages, and Programming*, volume 107 of *LIPIcs*, pages 157:1–157:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[20] 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.

[21] 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.

[22] J.F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences of the United States of America*, 1950.

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

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

[25] C. H. Papadimitriou. Algorithms, games, and the internet. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 749–753, 2001.

[26] 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.

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

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

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

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