An Abstraction-Refinement Methodology for Reasoning about Network Games *

Guy Avni
IST Austria
guy.avni@ist.ac.at

Shibashis Guha
The Hebrew University
shibashis@cs.huji.ac.il

Orna Kupferman
The Hebrew University
orna@cs.huji.ac.il

Abstract

Network games (NGs) are played on directed graphs and are extensively used in network design and analysis. Search problems for NGs include finding special strategy profiles such as a Nash equilibrium and a globally optimal solution. The networks modeled by NGs may be huge. In formal verification, *abstraction* has proven to be an extremely effective technique for reasoning about systems with big and even infinite state spaces. We describe an abstraction-refinement methodology for reasoning about NGs. Our methodology is based on an abstraction function that maps the state space of an NG to a much smaller state space. We search for a global optimum and a Nash equilibrium by reasoning on an under- and an over-approximation defined on top of this smaller state space. When the approximations are too coarse to find such profiles, we *refine* the abstraction function. Our experimental results demonstrate the efficiency of the methodology.

1 Introduction

Network design is a fundamental and well-studied problem. A game-theoretic approach to the analysis of network design has become especially relevant with the emergence of the Internet, where different users share resources like software or communication channels [Fabrikant *et al.*, 2003; Albers *et al.*, 2014; Anshelevich *et al.*, 2008]. Network games (NGs, for short) [Anshelevich *et al.*, 2008; Rosenthal, 1973; Roughgarden and Tardos, 2002] constitute a well studied model of non-cooperative games. The game is played among selfish players on a network, which is a directed graph. NGs are used to model resources as edges in a network and the cost involved in sharing these resources. Each player has a source and a target vertex, and a strategy is a choice of a path that connects these two vertices. The cost a player pays is the sum of costs of the edges his path traverses, where a cost of an edge depends on the *load* on it, namely the number of players using the edge. We distinguish between two types of costs. In *cost-sharing* games [Anshelevich *et al.*, 2008], each edge has a cost and the players that use it split the cost among them, thus increased load decreases the cost. For example, when the costs correspond to prices, users that share a resource also share its price. Then, in *congestion* games [Rosenthal, 1973], increased load increases the cost: each edge has a non-decreasing *cost function* that maps the load on the edge to its cost given this load. For example, when the network models a road system and costs correspond to the travel time, an increased load on an edge corresponds to a traffic jam, increasing the cost of the players that use it.

Since the players attempt to minimize their own costs, they *selfishly* select a path. The focus in game theory is on the *stable* outcomes of a given setting. The most prominent stability concept is that of a *Nash equilibrium* (NE): a profile (vector of strategies, one for each player) such that no player can decrease his cost by unilaterally deviating from his current strategy; i.e., assuming that the strategies of the other players do not change.[1] By [Rosenthal, 1973], there exists an NE in every NG. A *social optimum* (SO) is a profile that minimizes the sum of the players' costs; thus the one obtained if the players would have obeyed some centralized authority.

Finding SO and NE profiles is a complex and well studied problem. Finding an SO is NP-complete [Tardos and Wexler, 2007; Meyers, 2006], and finding an NE is PLS-complete [Fabrikant *et al.*, 2004; Syrgkanis, 2010] (the complexity class PLS contains local search problems with polynomial time searchable neighborhoods [Johnson *et al.*, 1988]). These high complexities become more acute if we recall that NGs often model huge networks.

The need to cope with very large models is a major research area in *formal verification*. There, we check that a system satisfies its specification by translating the system into some formal model, typically a labeled state-transition graph, and applying model-checking procedures on this model [Clarke *et al.*, 1999]. One of the most successful methodologies for reasoning about the huge state space of systems is *abstraction* [Cousot and Cousot, 1977; Larsen, 1989], where

*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, ERC grant no 278410) and the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award).

[1] Throughout this paper, we consider *pure* strategies, as is the case for the vast literature on NGs.

we hide some of the information about the system. This enables us to reason about systems that are much smaller, yet it gives rise to approximated solutions. Indeed, hiding of information may result in an *under-approximating* system: one that has fewer behaviors than the original system, or in an *over-approximating* system: one that has more behaviors than the original system. Abstraction methodologies use both types of approximations [Dams *et al.*, 1997; Bruns and Godefroid, 1999; Shoham and Grumberg, 2004].

An important step in methodologies that are based on abstraction is *refinement*. Assume that we find an over-approximation of the system that does not satisfy a universal property. That is, the over-approximation has a forbidden behavior. It may be that this forbidden behavior exists also in the concrete system, in which case the verification algorithm terminates and reports a bug in the system. But it may also be that the forbidden behavior exists only in the over-approximation, thus the counterexample is *spurious*. Then, one can use the spurious counterexample to refine the over-approximation in a way that eliminates it, and repeat model-checking until either a real counterexample is found, or the over-approximation satisfies the property. This methodology, of *counterexample guided abstraction-refinement* (CEGAR) has proven to be very effective [Clarke *et al.*, 2003].

Abstraction has been studied in the context of extensive-form games. A strategy for a player in an extensive-form game prescribes which action to perform, given the histories of actions played so far. NGs, on the other hand, are "one-shot" games. The questions studied on NGs focus on stable outcome whereas in extensive-form games one often tries to find an optimal strategy for a player. The abstraction studied in [Gilpin and Sandholm, 2007b; Brown and Sandholm, 2015] tries to merge states in the game tree that are indistinguishable for the players. The idea of merging configurations due to hidden information is the key also in abstractions used in formal methods, yet the technical details are very different. Then, in *action abstraction* [Brown and Sandholm, 2015; Gilpin *et al.*, 2008], some of the actions of the players are disabled, reducing the state space of reachable configurations in the game tree, which is again not similar to the approach taken here. Finally, formal methods often involves reasoning about multi-player games, and abstraction-refinement methodologies have been studied in this setting [Henzinger *et al.*, 2000; de Alfaro *et al.*, 2004; Ball and Kupferman, 2006]. Such games model on-going interactions between processes, say a system and its environment, and are infinite-duration games, thus they are again different from the NGs we study here.

In this paper we introduce and study an abstraction-refinement methodology for reasoning about network games. Given an NG $\mathcal{N}$ defined over a network $\langle V, E\rangle$, an *abstraction function* for $\mathcal{N}$ is a function $\alpha : V \to A$, for some set A of abstract vertices. We assume that A is smaller than V, thus the function α induces a partition of V. We define the *under-approximation of $\mathcal{N}$ with respect to α*, denoted $\mathcal{N}^{\downarrow}[\alpha]$, and the *over-approximation of $\mathcal{N}$ with respect to α*, denoted $\mathcal{N}^{\uparrow}[\alpha]$. Both approximations are NGs that have A as their set of vertices. The under- and over-approximation is in the definition of the edges and the cost functions. Intuitively, $\mathcal{N}^{\downarrow}[\alpha]$ is less appealing to the players than $\mathcal{N}$: they have fewer possible profiles, and the profiles that are possible are at least as expensive as the ones that correspond to them in $\mathcal{N}$. Accordingly, the edges under-approximate these in $\mathcal{N}$: there is an edge from an abstract vertex a to an abstract vertex a' if all the concrete vertices that are mapped by α to a have an edge to a concrete vertex that is mapped by α to a' (a.k.a. *must* transitions [Larsen, 1989; Dams *et al.*, 1997]). In addition, the cost of an abstract edge is essentially the maximal cost of a concrete edge that induces it. Dually, $\mathcal{N}^{\uparrow}[\alpha]$ is more appealing to the players: they have more and cheaper profiles than in $\mathcal{N}$. Accordingly, the edges in $\mathcal{N}^{\uparrow}[\alpha]$ over-approximate these in $\mathcal{N}$: there is an edge from a to a' if some concrete vertex that is in a has an edge to some concrete vertex that is mapped to a' (a.k.a. *may* transitions), and the cost of an abstract edge is essentially the minimal cost of a concrete edge that induces it. Traditional abstraction-refinement methodologies in formal verification focus on the transition relation. An extension that takes costs into account has been studied in [Avni and Kupferman, 2012], where the costs of a weighted automaton are also abstracted. Here, we take into account the cost functions as well as the load. Indeed, the merging of edges may lead to a spurious increased load in the abstraction.

We show how $\mathcal{N}^{\downarrow}[\alpha]$ and $\mathcal{N}^{\uparrow}[\alpha]$, which may be significantly smaller than $\mathcal{N}$, can be used in order to reason about the SO and the NE profiles of $\mathcal{N}$. Our methodology is based on the observation that each profile in $\mathcal{N}^{\downarrow}[\alpha]$ can be mapped to at least one profile in $\mathcal{N}$ with a lower or equal cost, and that each profile in $\mathcal{N}$ can be mapped to a profile in $\mathcal{N}^{\uparrow}[\alpha]$ with a lower or equal cost. Hence, for example, the cost of the SO in $\mathcal{N}$ is bounded from above and below by the costs of the SOs in $\mathcal{N}^{\downarrow}[\alpha]$ and $\mathcal{N}^{\uparrow}[\alpha]$, respectively. Moreover, refining α tightens these bounds, so the user can control the trade-off between preciseness and complexity.

A more technically-involved use of the under- and over-approximations is an algorithm we present for finding an NE in $\mathcal{N}$. The algorithm, which can be viewed as the NG-analogue of CEGAR, is based on the notion of an *abstract NE*: we say that a profile P in $\mathcal{N}^{\downarrow}[\alpha]$ is an abstract NE if no player has a beneficial deviation from P even in $\mathcal{N}^{\uparrow}[\alpha]$. Intuitively, an abstract NE has to face two challenges. First, the profile P has to exist in the under-approximation, where fewer strategies exist. In addition, deviations from P are possible in the over-approximation, where more strategies exist, and their cost is lower. Consequently, as we shall formally prove, an abstract NE can direct the search for a concrete NE: once we find an abstract NE P in $\mathcal{N}^{\downarrow}[\alpha]$, it is guaranteed that a concrete NE exists in $\mathcal{N}$ when restricted to profiles that agree with P. Our algorithm finds an abstract NE if one exists and then directs the search for a concrete NE in a much smaller state space. It is however not necessary that an abstract NE exists in every abstract game. When a candidate profile in $\mathcal{N}^{\downarrow}[\alpha]$ is an NE in $\mathcal{N}^{\downarrow}[\alpha]$ but is not an abstract NE in $\mathcal{N}$, we use the *spurious deviations* of the players in $\mathcal{N}^{\uparrow}[\alpha]$ in order to refine α, which narrows the search space.

Our experimental results demonstrate the efficiency of the methodology. The results show that the overhead required for abstraction becomes more negligible for larger systems.

This work belongs to a line of works that transfer con-

cepts and ideas between the areas of formal verification and algorithmic game theory: logics for specifying multi-agent systems [Alur *et al.*, 2002; Chatterjee *et al.*, 2007], studies of equilibria in games related to synthesis and repair problems [Chatterjee *et al.*, 2006; Chatterjee, 2006; Fisman *et al.*, 2010; Almagor *et al.*, 2015], and of non-zero-sum games in formal verification [Chatterjee *et al.*, 2004; Brihaye *et al.*, 2012]. Closest to this work are works that apply ideas from formal verification of NGs. This includes an extension of NGs to objectives that are richer than reachability [Avni *et al.*, 2016b], NGs in which the players select their paths dynamically [Avni *et al.*, 2016a], and efficient reasoning about NGs that are structures in a hierarchical manner [Kupferman and Tamir, 2017].

2 Preliminaries

2.1 Network Games

A network is a tuple $\langle V, E\rangle$, where V is a set of vertices and $E \subseteq V \times V$ is a set of directed edges. For an integer $k \in \mathbb{N}$, let $[k] = \{1, \ldots, k\}$. A *network game* (NG) is $\mathcal{N} = \langle k, V, E, \{l_e\}_{e\in E}, \langle s_i, t_i\rangle_{i\in[k]}\rangle$, where k is the number of players; $\langle V, E\rangle$ is a network; for $e \in E$, the cost function $l_e : [k] \to \mathbb{R}_{\geq 0}$ maps the load on edge e, namely the number of players that use edge e, to the cost each of them pays for using e with this load; and for $i \in [k]$, the pair $\langle s_i, t_i\rangle \in V \times V$ describes the objective of Player i: traversing $\mathcal{N}$ from the source vertex s_i to the target vertex t_i.

We distinguish between two types of cost functions. In *uniform cost-sharing games* (CS-NGs, for short) the players that use an edge share its cost equally. Formally, each edge e is associated with a weight $w_e \in \mathbb{R}_{\geq 0}$, and for all $x \in [k]$, we have $l_e(x) = \frac{w_e}{x}$. Thus, increasing the load in uniform cost-sharing games decreases the cost of the players. In contrast, in *congestion games* (CON-NGs, for short), the cost functions are non-decreasing, thus increasing the load also increases the cost for each player.

A *strategy* of a player $i \in [k]$ is a simple path π from s_i to t_i. Thus, $\pi = \langle v_1, v_2\rangle, \langle v_2, v_3\rangle, \ldots, \langle v_{n-1}, v_n\rangle$, with $v_1 = s_i$, $v_n = t_i$, and $(v_j, v_{j+1}) \in E$ for all $1 \leq j < n$. A *profile* is a tuple of strategies, one for each player. Consider a profile $P = \langle \pi_1, \pi_2, \ldots, \pi_k\rangle$ in the game. We sometimes refer to a path as the set of edges that it traverses, thus $\pi \subseteq E$. For an edge $e \in E$, we use $load_P(e)$ to denote the number of players that traverse the edge e in P. Each player that uses e then pays $l_e(load_P(e))$, and the cost of Player i in P, denoted $cost_i(P)$, is $\sum_{e\in\pi_i} l_e(load_P(e))$. The cost of the profile P, denoted $cost(P)$, is the total cost incurred by all the players, i.e., $cost(P) = \sum_{i=1}^{k} cost_i(P)$. For a profile P and a strategy π of player $i \in [k]$, let $P[i \leftarrow \pi]$ denote the profile obtained from P by replacing the strategy for Player i by π. Given a profile $P = \langle \pi_1, \ldots, \pi_k\rangle$, a *best response* (BR, for short) of a player, say Player k, in profile P is a strategy π'_k such that for all strategies π of Player k, we have that $cost_k(P[k \leftarrow \pi'_k]) \leq cost_k(P[k \leftarrow \pi])$.

Formally, a profile P is an NE iff for every Player i and every strategy π, we have $cost_i(P[i \leftarrow \pi]) \geq cost_i(P)$. A *social optimum* (SO) of a game $\mathcal{N}$ is a profile that attains the minimal cost over all profiles. We denote by $SO(\mathcal{N})$ the cost of an SO profile; i.e., $SO(\mathcal{N}) = \min_P cost(P)$. An SO can be thought of as an optimal solution imposed by a centralized authority, and need not be an NE.

2.2 Abstraction and Refinement

Consider an NG $\mathcal{N} = \langle k, V, E, \{l_e\}_{e\in E}, \langle s_i, t_i\rangle_{i\in[k]}\rangle$. We refer to V as the set of *concrete vertices*. Let $T = \{t_1, \ldots, t_k\}$. An *abstraction function* for $\mathcal{N}$ is a function $\alpha : V \to A$, for a set A of *abstract vertices*. We assume that $T \subseteq A$ and that α is such that for all $t_i \in T$, we have $\alpha(t_i) = t_i$ and $\alpha(v) \neq t_i$ for all $v \neq t_i$. We also assume that A is smaller than V, thus the function α induces a partition of V (with a singleton $\{t_i\}$ for each $t_i \in T$). Accordingly, we sometimes refer to abstract vertices as sets of concrete vertices. In particular, when α is clear from the context, we use $v \in a$, for $v \in V$ and $a \in A$, to indicate that $\alpha(v) = a$.

Consider the NG $\mathcal{N}$ and an abstraction function α. We define the *under-* and *over-*approximation of $\mathcal{N}$ formally. Given $\mathcal{N}$, α, and $b \in \{\downarrow, \uparrow\}$, we define $\mathcal{N}^b[\alpha] = \langle k, V, E^b, \{l^b_e\}_{e\in E^b}, \langle \alpha(s_i), \alpha(t_i)\rangle_{i\in[k]}\rangle$, where the under- and over-approximating transition relations $E^\downarrow, E^\uparrow \subseteq A \times A$, and the under- and over-approximating cost functions $l^\downarrow_e$ and $l^\uparrow_e$ are defined as follows.

Transition relations: Consider two abstract vertices $a, a' \in A$. Then, $E^\downarrow(a, a')$ iff for every concrete vertex $v \in a$, there is a concrete vertex $v' \in a'$ such that $E(v, v')$. Also, $E^\uparrow(a, a')$ iff there exist concrete vertices $v \in a$ and $v' \in a'$ such that $E(v, v')$. Note that $E^\downarrow \subseteq E^\uparrow$. For simplicity, we omit self-loops from $E^\downarrow$ and $E^\uparrow$, as they are not going to be used anyway in strategies. We follow the common terminology in formal verification and refer to the under- and over-approximating edges as *must* and *may* edges, respectively. We extend α to edges in the expected way. Thus, $\alpha(h)$, for an edge $h = \langle v, v'\rangle \in E$, is $\langle \alpha(v), \alpha(v')\rangle$. Note that $\alpha(h)$ is always in $E^\uparrow$ and may not be in $E^\downarrow$.

Cost Functions: The definition of the under- and over-approximating cost functions depends on the type of $\mathcal{N}$. We first describe the definition and then explain it.

- If $\mathcal{N}$ is a CON-NG, then
  - for every $e \in E^\downarrow$ and $x \in [k]$, we have $l^\downarrow_e(x) = \max\{l_h(x) : e = \alpha(h)\}$, and
  - for every $e \in E^\uparrow$ and $x \in [k]$, we have $l^\uparrow_e(x) = \min\{l_h(1) : e = \alpha(h)\}$.
- If $\mathcal{N}$ is a CS-NG, then
  - for every $e \in E^\downarrow$ and $x \in [k]$, we have $l^\downarrow_e(x) = \max\{l_h(1) : e = \alpha(h)\}$, and
  - for every $e \in E^\uparrow$ and $x \in [k]$, we have $l^\uparrow_e(x) = \min\{l_h(x) : e = \alpha(h)\}$.

The idea behind the definition is as follows. Recall that in the under-approximation $\mathcal{N}^\downarrow[\alpha]$, we want the strategies to be more expensive. This is why we take, in $l^\downarrow_e$, the maximal cost of edges that induce e. In CON-NGs, the cost increases with load and hence the cost function $l^\downarrow_e$ depends on x since we want more expensive profiles. In CS-NGs, we ignore x and assume that the load is 1. To see why, recall that an abstract edge $e \in E^\downarrow$ is obtained by merging several concrete edges. Consequently, the load on e is the sum

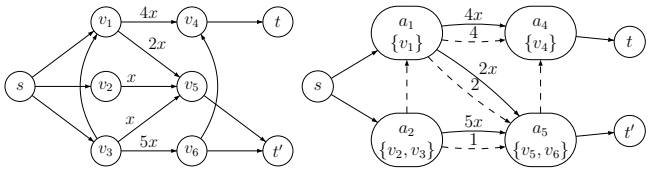

Figure 1: A CON-NG $\mathcal{N}$ (left) and its approximations $\mathcal{N}^{\downarrow}$ and $\mathcal{N}^{\uparrow}$, which share the same state space (right). Edges in $E^{\downarrow}$ are solid. Edges in $E^{\uparrow} \setminus E^{\downarrow}$ are dashed. Edges with no specified cost have cost 0.

of the loads on these concrete edges. This load is *fake*: it is only due to the merging of concrete edges and not due to an actual increased load. In CON-NGs, where the cost functions increase with an increased load, fake load goes well with generating more expensive profiles. In CS-NGs, however, increased load decreases the cost, and we have to cancel the fake load. This is done by dividing the load by itself, which bounds the fake load. Recall that in a CS-NG $\mathcal{N}$, each edge $h \in E$ has a weight w_h such that $l_h(x) = \frac{w_h}{x}$. Thus, as $l_h(1) = w_h$, the definition is equivalent to one with $l_e^{\downarrow}(x) = \max\{w_h : e = \alpha(h)\}$.

Dually, the over-approximating cost function aims at providing cheaper strategies. Accordingly, $l_e^{\uparrow}$ depends on the minimum cost function of edges that induce e. Here, we have to cancel fake load in CON-NGs, as fake load increases the cost and may cause the cost of an abstract edge to go beyond the cost of the concrete edges that induce it.

When α is clear from the context, we denote $\mathcal{N}^b[\alpha]$ by $\mathcal{N}^b$. When we refer to the cost of a profile P in $\mathcal{N}^b$, we use the notation $cost^b(P)$, to emphasize that the profile P is in $\mathcal{N}^b$. In Figure 1, we show an NG on the left and an abstraction to the right.

Let us emphasize the confusing fact that when we talk about an *under*-approximation, we take the *maximum* cost. This may seem counterintuitive. In order not to get confused, recall that the thing we are approximating is the range and attractiveness of possible profiles. In an under-approximation, we want both fewer and more expensive profiles. Dually, in an *over*-approximation, we take the *minimum* cost, as we want more and cheaper profiles. A similar intuition applies for the adjustment of the load.

Definition [Abstract NE] A profile $P = \langle \pi_1, \ldots, \pi_k \rangle$ in $\mathcal{N}^{\downarrow}$ is an *abstract NE* if no player has a beneficial deviation from P even in $\mathcal{N}^{\uparrow}$. Formally, for all $i \in [k]$ and strategies $\pi_i' \neq \pi_i$ of Player i in $\mathcal{N}^{\uparrow}$, we have $cost_i^{\downarrow}(P) \leq cost_i^{\uparrow}(P[i \leftarrow \pi_i'])$.

Intuitively, an abstract NE has to face two challenges. First, the profile P has to exist in the under-approximation, where fewer strategies exist. Second, existence of deviations from P is checked in the over-approximation, where more strategies may exist, and their cost is lower. Consequently, as we formally prove in Theorem 3, an abstract NE directs the search for a concrete NE: once we find an abstract NE P in $\mathcal{N}^{\downarrow}$, we know that a concrete NE exists in $\mathcal{N}$ when restricted to profiles that agree with P. Formally, given an NG $\mathcal{N}$ and a profile P in $\mathcal{N}^{\downarrow}$, the *restriction of* $\mathcal{N}$ *to* P is the NG $\mathcal{N}_{|P} = \langle k, V_{|P}, E_{|P}, \{l_e\}_{e \in E_{|P}}, \langle s_i, t_i \rangle_{i \in [k]} \rangle$, where $V_{|P} = \{v \in V : \alpha(v)$ appears in a strategy in $P\}$ and $E_{|P} = \{\langle v, v' \rangle \in E : \langle \alpha(v), \alpha(v') \rangle$ appears in a strategy in $P\}$.

The generation of $\mathcal{N}^{\downarrow}$ and $\mathcal{N}^{\uparrow}$ depends on the way $\mathcal{N}$ is given. When $\mathcal{N}$ is given in a succinct presentation, it is often possible to construct $\mathcal{N}^{\downarrow}$ and $\mathcal{N}^{\uparrow}$ on top of this succinct presentation. Of special interest, especially in the context of software-defined networks [Vissicchio *et al.*, 2014], are NGs in which $V = 2^X$ for some set X of variables. Then, abstraction functions are based on predicates on the variables in X, and the construction of $\mathcal{N}^{\downarrow}$ and $\mathcal{N}^{\uparrow}$ is done on top of this succinct presentation $\mathcal{N}$ that is based on the variables in X.

Consider two abstraction functions $\alpha_1 : V \to A_1$ and $\alpha_2 : V \to A_2$. We say that α_2 *refines* α_1, denoted $\alpha_2 \preceq \alpha_1$, if for all concrete vertices v and v', if $\alpha_2(v) = \alpha_2(v')$, then $\alpha_1(v) = \alpha_1(v')$. That is, the partition of V that is induced by α_2 refines the partition induced by α_1.

3 On Abstract SOs and NEs

In this section we study the theoretical properties of abstraction in NGs and show how reasoning about the (much smaller) under- and over-approximations of an NG $\mathcal{N}$ can be used for bounding the cost of an SO and for directing the search for an NE in $\mathcal{N}$. We first relate strategies and profiles in $\mathcal{N}$ with strategies and profiles in its approximations.

Consider a network $\mathcal{N}$ and a strategy π of Player i in $\mathcal{N}$. The strategy $\alpha(\pi)$ that corresponds to π in $\mathcal{N}^{\uparrow}$ is obtained from π by replacing each concrete edge h by the abstract edge $\alpha(h)$, and removing cycles in the obtained path in $\mathcal{N}^{\uparrow}$. Note that by the definition of $E^{\uparrow}$, the edge $\alpha(h)$ exists in $\mathcal{N}^{\uparrow}$. Formally, we define $\alpha(\pi)$ as follows. Let $\pi = \langle v_1, v_2 \rangle, \langle v_2, v_3 \rangle, \ldots, \langle v_{n-1}, v_n \rangle$. We first define $\alpha'(\pi) = \langle \alpha(v_1), \alpha(v_2) \rangle, \langle \alpha(v_2), \alpha(v_3) \rangle, \ldots, \langle \alpha(v_{n-1}), \alpha(v_n) \rangle$. Then, $\alpha(\pi)$ is obtained from $\alpha'(\pi)$ by removing cycles; that is, by repeatedly removing subsequences $\langle \alpha(v_j), \alpha(v_{j+1}) \rangle, \ldots, \langle \alpha(v_{j+m}), \alpha(v_{j+m+1}) \rangle$ with $\alpha(v_j) = \alpha(v_{j+m+1})$. A profile $P = \langle \pi_1, \ldots, \pi_k \rangle$ in $\mathcal{N}$ corresponds to the profile $\alpha(P) = \langle \alpha(\pi_1), \ldots, \alpha(\pi_k) \rangle$ in $\mathcal{N}^{\uparrow}$.

Consider now a strategy $\pi = \langle a_1, a_2 \rangle, \langle a_2, a_3 \rangle, \ldots, \langle a_{n-1}, a_n \rangle$ of Player i in $\mathcal{N}^{\downarrow}$. By the definition of $E^{\downarrow}$, for every concrete vertex v with $\alpha(v) = a_1$, and in particular for s_i – the source vertex of Player i, there is a path in $\mathcal{N}$ from v to some vertex v' with $\alpha(v') = a_{n-1}$. Also, by the definition of $\mathcal{N}^{\downarrow}$, we have that $a_n = t_i$ – the target vertex of Player i [2], and for all the concrete vertices v' with $\alpha(v') = a_{n-1}$, we have $E(v', t_i)$. Hence, the strategy π in $\mathcal{N}^{\downarrow}$ induces at least one path $\pi' = \langle v_1, v_2 \rangle, \langle v_2, v_3 \rangle, \ldots, \langle v_{n-1}, v_n \rangle$ in $\mathcal{N}$ such that $v_1 = s_i$ and $v_n = t_i$. Let $\alpha^{-1}(\pi)$ be the nonempty set of these paths. Finally, a profile $P = \langle \pi_1, \ldots, \pi_k \rangle$ in $\mathcal{N}^{\downarrow}$ corresponds to the set $\alpha^{-1}(P)$ of profiles $P' = \langle \pi_1', \ldots, \pi_k' \rangle$ in $\mathcal{N}$ in which for all $i \in [k]$, we have $\pi_i' \in \alpha^{-1}(\pi_i)$.

We now relate the costs of corresponding profiles in $\mathcal{N}$, $\mathcal{N}^{\downarrow}$, and $\mathcal{N}^{\uparrow}$.

Lemma 1 *Consider an NG $\mathcal{N}$ and an abstraction function α.*

1. *For every profile P in $\mathcal{N}$, the profile $\alpha(P)$ in $\mathcal{N}^{\uparrow}$ is such that $cost^{\uparrow}(\alpha(P)) \leq cost(P)$.*

[2] We note that this is the point where we use the fact that $\alpha(t_i)$ is a singleton, for every $i \in [k]$.

2. *For every profile P in $\mathcal{N}^{\downarrow}$ and profile $P' \in \alpha^{-1}(P)$ in $\mathcal{N}$, we have that $cost(P') \leq cost^{\downarrow}(P)$.*

Theorem 1 *For every NG $\mathcal{N}$ and abstraction function α, we have $SO(\mathcal{N}^{\uparrow}[\alpha]) \leq SO(\mathcal{N}) \leq SO(\mathcal{N}^{\downarrow}[\alpha])$.*

Recall that given two abstraction functions α_1 and α_2, if α_2 refines α_1, then we can view $\mathcal{N}^{\downarrow}[\alpha_1]$ as an under-approximation of $\mathcal{N}^{\downarrow}[\alpha_2]$, and view $\mathcal{N}^{\uparrow}[\alpha_1]$ as an over-approximation of $\mathcal{N}^{\uparrow}[\alpha_2]$. Accordingly, Theorem 1 can be viewed as a special case of Theorem 2 below, with α_2 being the most refined abstraction function (that is, $\alpha_2 : V \to V$, with $\alpha_2(v) = v$), and α_1 being the refinement function α studied there.

Theorem 2 *Consider an NG $\mathcal{N}$ and two abstraction functions α_1 and α_2. If $\alpha_2 \preceq \alpha_1$, then $SO(\mathcal{N}^{\uparrow}[\alpha_1]) \leq SO(\mathcal{N}^{\uparrow}[\alpha_2])$ and $SO(\mathcal{N}^{\downarrow}[\alpha_2]) \leq SO(\mathcal{N}^{\downarrow}[\alpha_1])$.*

Theorem 1 enables us to approximate the cost of an SO in $\mathcal{N}$ using the costs of the SO in the much smaller $\mathcal{N}^{\downarrow}$ and $\mathcal{N}^{\uparrow}$. We now turn to study how $\mathcal{N}^{\downarrow}$ and $\mathcal{N}^{\uparrow}$ can be used in order to direct the search for an NE in $\mathcal{N}$.

Theorem 3 *Consider an NG $\mathcal{N}$, an abstraction function α for it, and an abstract NE P in $\mathcal{N}^{\downarrow}[\alpha]$. There exists a profile in $\alpha^{-1}(P)$ that is a concrete NE in $\mathcal{N}$.*

Note that profiles in $\alpha^{-1}(P)$ can be searched for in $\mathcal{N}_{|P}$. Thus, as we elaborate in Section 4, an NE in $\mathcal{N}$ can be found by a sequence of best response moves restricted to $\mathcal{N}_{|P}$.

4 An Abstraction-Refinement Procedure for Finding an NE

In this section we describe an abstraction-refinement procedure for finding an NE in an NG . The input to the procedure is a concrete NG $\mathcal{N}$ and an abstraction function α for it. Experience in formal verification suggests that abstraction functions that are supplied by users familiar with the network, are the most successful ones. Alternatively, one can start with a coarse abstraction and refine it as we do in Section 5.

The output of the procedure is a concrete NE in $\mathcal{N}$. Since the state space of $\mathcal{N}^{\downarrow}$ and $\mathcal{N}^{\uparrow}$ is much smaller than that of $\mathcal{N}$, we would like to perform as many as possible computations on the approximating networks. Our procedure (see Fig. 2 for an overview) consists of two parts. The first, in which an abstract NE P_α is found, is done entirely in $\mathcal{N}^{\downarrow}$ and $\mathcal{N}^{\uparrow}$, and it is the procedure we have implemented. The second, in which a concrete NE is found, is done in $\mathcal{N}$, restricted to $\mathcal{N}_{|P_\alpha}$. Thus, as is the case of the CEGAR methodology in formal verification, there is no way to avoid $\mathcal{N}$ entirely, yet we can significantly restrict the part of it in which the search proceeds. Moreover, it is possible to refine α and tighten $\mathcal{N}_{|P_\alpha}$ further. In Section 5, we show that the restriction indeed significantly reduces the size of the network.

There are many ways to refine an abstraction; one can work naively, choose an arbitrary abstract vertex and split it in some way, possibly by adding predicates that appear in the description of the network or the strategies. Even such a naive refinement is guaranteed to eventually lead to a solution. The challenge is to choose the refinements cleverly so that a concrete answer is obtained when the approximating networks

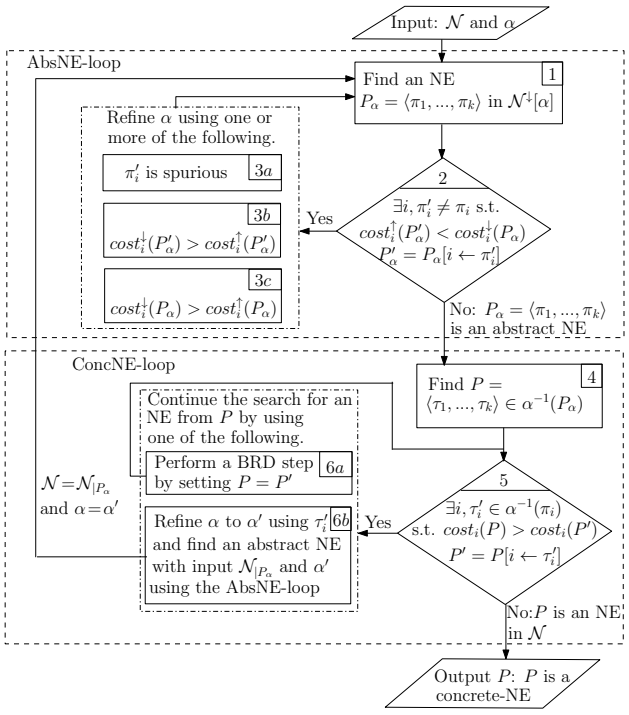

Figure 2: Finding an NE in $\mathcal{N}$.

are still much smaller than the concrete one. In CEGAR, the refinement is guided by a spurious counterexample. We follow this idea by always refining according to some path in the network that points to a spurious behavior of the approximations. We now describe the methodology in detail.

4.1 Part 1: AbsNE-Loop, Finding an Abstract NE

In the first part, our goal is to find, given $\mathcal{N}$ and α, an abstract NE. Recall that such a profile is an NE in $\mathcal{N}^{\downarrow}$ that is resistant to deviations of the players even in $\mathcal{N}^{\uparrow}$. Since $\mathcal{N}^{\downarrow}$ is an NG, it has an NE. In Step 1 in Fig. 2, we find such an NE P_α. This is done by the user's favorite algorithm for finding an NE in NGs. The important point for us is that this is done in the (much smaller) under-approximation of $\mathcal{N}$. Then, in Step 2, we check whether P_α is an abstract NE. Thus, we check whether players have beneficial deviations in $\mathcal{N}^{\uparrow}$. Again, this is done in the (much smaller) over-approximation of $\mathcal{N}$. If no player has a beneficial deviation in $\mathcal{N}^{\uparrow}$, then P_α is an abstract NE, we conclude this part of the procedure, and move to Step 4. Otherwise, there is a player $i \in [k]$ who can benefit from deviating to a strategy π'_i.

We use π'_i in order to guide the refinement. There are several reasons why P_α is an NE in $\mathcal{N}^{\downarrow}$ yet not an abstract NE in $\mathcal{N}$. Step 3 consists of three possible refinement steps among which the user can choose, corresponding to the above different reasons. Let us start with Step 3a. Since π'_i is a path in $\mathcal{N}^{\uparrow}$, there might not be a concrete path in $\mathcal{N}$ that corresponds to it, thus π'_i is a *spurious* path. Consider two adjacent abstract vertices a_1 and a_2 that π'_i traverses. If the edge between a_1 and a_2 is in $E^{\uparrow} \setminus E^{\downarrow}$, we can split a_1 into a'_1 and a''_1

such that a'_1 contains exactly the vertices that have a neighbor in a_2 (similarly we can split a_2). Note that after refinement, there is a must edge from a'_1 to a_2 and there is not even a may edge between a''_1 to a_2. Since π'_i is spurious, such a candidate vertex is guaranteed to exist. (We note that disconnectivity in $\mathcal{N}^{\downarrow}$ can be treated in a similar way.)

We continue to Steps 3b and 3c. They have to do with under- and over-approximations of the cost functions that cause π'_i to be a beneficial deviation in $\mathcal{N}^{\uparrow}$. By the definition of $l^{\downarrow}$ and $l^{\uparrow}$, we know that if $cost_i^{\downarrow}(P_\alpha[i \leftarrow \pi'_i]) > cost_i^{\uparrow}(P_\alpha[i \leftarrow \pi'_i])$, then the path π'_i traverses an abstract edge $e \in E^{\downarrow}$ with load x for which $l_e^{\downarrow}(x) > l_e^{\uparrow}(x)$. Assume that $e = \langle a_1, a_2 \rangle$. In Step 3b, we split a_1 or a_2 in order to tighten this gap. Finally, it may be that the cost of the strategy of Player i in P_α is too big. In Step 3c, we use the strategy π_i that Player i chooses in P_α in order to guide a similar refinement in order to tighten the gap in the costs between $cost_i^{\downarrow}(P_\alpha)$ and $cost_i^{\uparrow}(P_\alpha)$.

A refinement step can be a single refinement or a combination of the refinements that are described above. After completing such a step, we return to Step 1 and find a new NE in the new under-approximating $\mathcal{N}^{\downarrow}$, and repeat the process.

4.2 Part 2: ConcNE-Loop, Finding a Concrete NE

The second part of the procedure gets an abstraction function α as well as an abstract NE $P_\alpha = \langle \pi_1, \ldots, \pi_k \rangle$. The goal is to find a concrete NE P in $\mathcal{N}$ such that $P \in \alpha^{-1}(P_\alpha)$. By Theorem 3, such an NE exists. Recall the best-response dynamics (BRD) algorithm for finding an NE in NGs in which we start with an arbitrary profile, and iteratively allow the players to perform beneficial deviations. We follow this algorithm and start in Step 4 with an arbitrary profile $P = \langle \tau_1, \ldots, \tau_k \rangle$ in $\alpha^{-1}(P_\alpha)$. If P is an NE, we are done. Otherwise, there is a concrete beneficial deviation τ'_i for some Player i. Note that by Theorem 3, we can restrict the deviations of Player i to paths in $\alpha^{-1}(\pi_i)$. If the size of $\mathcal{N}_{|P_\alpha}$ is small, the user can choose Step 6a and try and find an NE in $\alpha^{-1}(P_\alpha)$ by performing a BRD step. However, when $\mathcal{N}_{|P_\alpha}$ is big, it makes sense to refine the abstraction by choosing Step 6b.

In Step 6b, we let the deviation τ'_i guide the refinement. We look for a vertex v from which τ_i and τ'_i differ, thus from v, one path continues with a vertex v' while the other with v'', where $v' \neq v''$, yet $\alpha(v') = \alpha(v'')$. We refine the abstraction function by splitting $\alpha(v')$ so that v' and v'' are no longer in the same abstract vertex. We would like to have as many must edges as possible between the new vertices. One way to do it is to make v'' a singleton abstract state, but it is also possible to split $\alpha(v')$ as well as $\alpha(v)$ to achieve this. Once we conclude a refinement, we return to the first part of the procedure, and look for an abstract NE with the new abstraction.

5 Experimental Results

We have implemented our methodology and tested the performance of its AbsNE loop on randomly-generated cost-sharing games. We examined the benefit of the abstraction, namely we compared the size of the original game with the game that is truncated to the abstract-NE we find. We also examined the practicality of our approach, namely the number of

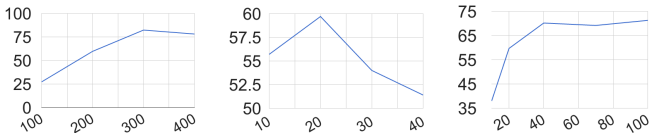

Figure 3: The number of iterations (y-axis) as $|V|$, k, and $|W|$ increase (x-axis).

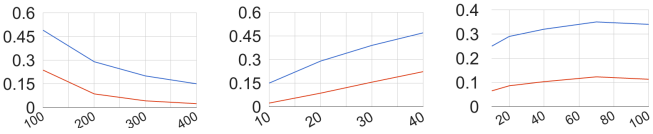

Figure 4: The ratio between the size (vertices in blue, edges in red) of the concrete and truncated networks as $|V|$, k, and $|W|$ increase.

CEGAR iterations until an abstract NE is found. We studied these questions for different parameters of the game; size of the graph, range of weights, and number of players. Our implementations are in Python, we use the library Networkx [Hagberg *et al.*, 2008] for graph generation and graph algorithms, and we ran our experiments on a personal computer, Intel Core i5 quad core 1.75 GHz processor, with 8 GB memory. Our results are encouraging: we are able to find an abstract NE relatively easily and it significantly reduces the size of the network making it easier to find a concrete NE.

We generate a random game, given the parameters $n, w, k \in \mathbb{N}$ and $p \in [0, 1]$. We use the Erdős-Réyni $G(n, p)$ model to generate the network. Then, for each edge in the graph, we choose at random a cost in a set $\{0, \ldots, w\}$. For each player $i \in [k]$, we choose at random a source vertex s_i and a reachable target vertex t_i. For the initial abstraction, we choose, for $i \in [k]$, a shortest path π_i between s_i and t_i, and we let every vertex that π_i traverses be a singleton abstract state. Thus, in the under approximation, we have at least one path from s_i to t_i. All other concrete vertices are mapped to one abstract state.

We perform three types of experiments. We focus on the number $|V|$ of vertices in the concrete network, the number k of players, and the range $|W|$ of weights on the edges. The number of edges is approximately $1/2 \cdot |V|^2$. We fix two of the parameters and increase the third. In Fig. 3, we see how increasing one of the parameters affects the number of iterations of the CEGAR loop. In Fig. 4, we compare the sizes of the truncated network, namely, $\alpha^{-1}(P_\alpha)$, and the original one; we show the ratio between the number of vertices in the two networks and the ratio between the number of edges.

We find the plots of the increasing number of vertices most encouraging. Since we fix the number of players, the part of the network that is being "used" becomes relatively smaller with increasing $|V|$, and an abstract NE has a good potential to shrink the network. Indeed, the ratios decrease. As further good news, the number of iterations reaches a ceiling and does not grow with $|V|$. Recall how we find an initial abstraction above. When k increases, there is a growing number of concrete vertices that are mapped to singleton abstract states in the initial abstraction. Thus, the abstraction is closer to the concrete network making it easier to find an abstract NE and increasing the ratios. For increasing $|W|$, we believe that at some point the large variance stops affecting the procedure.

References

[Albers *et al.*, 2014] S. Albers, S. Eilts, E. Even-Dar, Y. Mansour, and L. Roditty. On nash equilibria for a network creation game. *ACM Trans. Economics and Comput.*, 2(1):2:1–2:27, 2014.

[Almagor *et al.*, 2015] S. Almagor, G. Avni, and O. Kupferman. Repairing multi-player games. In *Proc. 26th CONCUR*, pages 325–339, 2015.

[Alur *et al.*, 2002] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *JACM*, 49(5):672–713, 2002.

[Anshelevich *et al.*, 2008] 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.

[Avni and Kupferman, 2012] G. Avni and O. Kupferman. Making weighted containment feasible: A heuristic based on simulation and abstraction. In *Proc. 23rd CONCUR*, pages 84–99, 2012.

[Avni *et al.*, 2016a] G. Avni, T.A. Henzinger, and O. Kupferman. Dynamic resource allocation games. In *Proc. 9th SAGT*, pages 153–166, 2016.

[Avni *et al.*, 2016b] G. Avni, O. Kupferman, and T. Tamir. Network-formation games with regular objectives. *I&C*, 251:165–178, 2016.

[Ball and Kupferman, 2006] T. Ball and O. Kupferman. An abstraction-refinement framework for multi-agent systems. In *Proc. 21st LICS*, 2006.

[Brihaye *et al.*, 2012] T. Brihaye, V. Bruyère, J. De Pril, and H. Gimbert. On subgame perfection in quantitative reachability games. *LMCS*, 9(1), 2012.

[Brown and Sandholm, 2015] N. Brown and T. Sandholm. Simultaneous abstraction and equilibrium finding in games. In *Proc. 24th IJCAI*, pages 489–496, 2015.

[Bruns and Godefroid, 1999] G. Bruns and P. Godefroid. Model checking partial state spaces with 3-valued temporal logics. In *Proc. 11th CAV*, pages 274–287, 1999.

[Chatterjee *et al.*, 2004] K. Chatterjee, R. Majumdar, and M. Jurdzinski. On Nash equilibria in stochastic games. In *Proc. 13th CSL*, pages 26–40, 2004.

[Chatterjee *et al.*, 2006] K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Games with secure equilibria. *TCS*, 365(1-2):67–82, 2006.

[Chatterjee *et al.*, 2007] K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In *Proc. 18th CONCUR*, pages 59–73, 2007.

[Chatterjee, 2006] K. Chatterjee. Nash equilibrium for upward-closed objectives. In *Proc. 15th CSL*, pages 271–286, 2006.

[Clarke *et al.*, 1999] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[Clarke *et al.*, 2003] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *JACM*, 50(5):752–794, 2003.

[Cousot and Cousot, 1977] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for the static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th POPL*, pages 238–252. ACM, 1977.

[Dams *et al.*, 1997] D. Dams, R. Gerth, and O. Grumberg. Abstract interpretation of reactive systems. *TOPLAS*, 19(2):253–291, 1997.

[de Alfaro *et al.*, 2004] L. de Alfaro, P. Godefroid, and R. Jagadeesan. Three-valued abstractions of games: Uncertainty, but with precision. In *Proc. 19th LICS*, pages 170–179, 2004.

[Fabrikant *et al.*, 2003] A. Fabrikant, A. Luthra, E. Maneva, C. Papadimitriou, and S. Shenker. On a network creation game. In *PODC*, pages 347–351, 2003.

[Fabrikant *et al.*, 2004] A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure nash equilibria. In *Proc. 36th STOC*, pages 604–612, 2004.

[Fisman *et al.*, 2010] D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *Proc. 16th TACAS*, pages 190–204, 2010.

[Gilpin and Sandholm, 2007a] A. Gilpin and T. Sandholm. Better automated abstraction techniques for imperfect information games, with application to texas hold'em poker. In *Proc. 6th AAMAS*, pages 1168–1175, 2007.

[Gilpin and Sandholm, 2007b] A. Gilpin and T. Sandholm. Lossless abstraction of imperfect information games. *JACM*, 54(5):25, 2007.

[Gilpin *et al.*, 2008] A. Gilpin, T. Sandholm, and T. B. Sørensen. A heads-up no-limit texas hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs. In *Proc. 7th AAMAS*, pages 911–918, 2008.

[Hagberg *et al.*, 2008] A. A. Hagberg, D. A. Schult, and P.J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proc. 7th SciPy*, pages 11–15, 2008.

[Henzinger *et al.*, 2000] T.A. Henzinger, R. Majumdar, F.Y.C. Mang, and J-F Raskin. Abstract interpretation of game properties. In *Proc. 7th SAS*, pages 245–252, 2000.

[Johnson *et al.*, 1988] D. S. Johnson, C. H. Papadimtriou, and M. Yannakakis. How easy is local search? *JCSS*, 37(1):79–100, 1988.

[Kupferman and Tamir, 2017] O. Kupferman and T. Tamir. Hierarchical network formation games. In *Proc. 23rd TACAS*, pages 229–246, 2017.

[Larsen, 1989] K.G. Larsen. Modal specifications. In *Automatic Verification Methods for Finite State Systems, Proc. Int. Workshop*, pages 232–246, 1989.

[Meyers, 2006] C. A. Meyers. *Network Flow Problems and Congestion Games: Complexity and Approximation Results*. PhD thesis, MIT, 2006.

[Rosenthal, 1973] R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *IJGT*, 2:65–67, 1973.

[Roughgarden and Tardos, 2002] T. Roughgarden and E. Tardos. How bad is selfish routing? *JACM*, 49(2):236–259, 2002.

[Shoham and Grumberg, 2004] S. Shoham and O. Grumberg. Monotonic abstraction-refinement for CTL. In *Proc. 10th TACAS*, pages 546–560, 2004.

[Syrgkanis, 2010] V. Syrgkanis. The complexity of equilibria in cost sharing games. In *WINE*, pages 366–377, 2010.

[Tardos and Wexler, 2007] E. Tardos and T. Wexler. *Algorithmic Game Theory*. Cambridge University Press, 2007. Chapter 19: Network Formation Games and the Potential Function Method.

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