Learning Equilibria of Games via Payoff Queries

JOHN FEARNLEY, University of Liverpool
MARTIN GAIRING, University of Liverpool
PAUL GOLDBERG, University of Liverpool
RAHUL SAVANI, University of Liverpool

A recent body of experimental literature has studied *empirical game-theoretical analysis*, in which we have partial knowledge of a game, consisting of observations of a subset of the pure-strategy profiles and their associated payoffs to players. The aim is to find an exact or approximate Nash equilibrium of the game, based on these observations. It is usually assumed that the strategy profiles may be chosen in an on-line manner by the algorithm. We study a corresponding computational learning model, and the query complexity of learning equilibria for various classes of games. We give basic results for bimatrix and graphical games. Our focus is on symmetric network congestion games. For directed acyclic networks, we can learn the cost functions (and hence compute an equilibrium) while querying just a small fraction of pure-strategy profiles. For the special case of parallel links, we have the stronger result that an equilibrium can be identified while only learning a small fraction of the cost values.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Non-numerical Algorithms and Problems—Computations on discrete structures.

Additional Key Words and Phrases: Payoff query complexity, strategic-form game, congestion game, equilibrium computation, approximate Nash equilibrium.

1. INTRODUCTION

Suppose that we have a game G with a known set of players, and known strategy sets for each player. We want to design an algorithm to solve G, where the algorithm can only obtain information about G via *payoff queries*. In a payoff query, the algorithm proposes pure strategies for the players, and is told the resulting payoffs. The general research issue is to identify bounds on the number of payoff queries needed to find an equilibrium, subject to the assumption that G belongs to some given class of games.

1.1. Motivation

Given a game, especially one with many players, it is unreasonable to assume that anyone maintains an explicit representation of its payoff function, even if the game in question has a concise representation. However, in practice a reasonable modelling assumption is that given, say, a strategy profile for the players, we can determine their payoffs, or some estimate of the payoffs. We are interested in algorithms that find Nash equilibria using a sequence of queries, where a query proposes a strategy profile and gets told the payoffs. We would like to know under what conditions an algorithm can find a solution based on knowledge of some but not all of the game's payoffs, which is particularly important when there are many players, and the number of pure-strategy profiles is large. This kind of challenge (where you get observations of profile/payoff-vector pairs, and you want to find an approximate equilibrium, as opposed to the unobserved payoffs) has been the subject of

experimental work [Vorobeychik et al. 2007; Wellman 2006; Jordan et al. 2008; Duong et al. 2009], where [Jordan et al. 2008] focuses on the case (highly relevant to this work) where the algorithm selects a sequence of pure profiles and gets told the resulting payoffs. *In this paper, we introduce the study of payoff-query algorithms from the algorithmic complexity viewpoint.* We are interested in upper and lower bounds on the query complexity of classes of games.

From the theoretical perspective, we are studying a constrained class of algorithms for computing equilibria of games. The study of such constraints —especially when they lead to lower bounds or impossibility results— informs us about the approaches that a successful algorithm needs to apply. In the context of equilibrium computation, other kinds of constraint include *uncoupled* algorithms for computing equilibria [Hart and Mas-Colell 2003, 2006], communication-constrained algorithms [Hart and Mansour 2010; Daskalakis et al. 2010; Goldberg and Pastink 2012], and *oblivious* algorithms [Daskalakis and Papadimitriou 2009]. Of course, the restriction to polynomial-time algorithms is the best-known example of such a constraint. Based on the algorithms and open problems identified in this paper, we find this to be quite a compelling motivation for the further study of the payoff-query model. There are various related kinds of query models that are suggested by the payoff queries studied here, which may also be of similar theoretical interest; we discuss these in Section 6.

1.2. Games and query models

In this paper we introduce the study of payoff-queries for strategic-form games. We also consider two models of concisely represented games: *graphical games* [Kearns et al. 2001], where players are nodes in a given graph and the payoff of a player only depends on the strategies of its neighbors in the graph, and *symmetric network congestion games* [Fabrikant et al. 2004], where the strategy space of the players corresponds to the set of paths that connect two nodes in a network.

For a strategic-form game, we assume that initially the querying algorithm only knows n the number of players and k the number of pure strategies that each player has. Generally we seek algorithms that are polynomial in these parameters.

Definition 1.1. A *payoff query* to a strategic-form game G selects a pure-strategy profile s for G, and is given as response, the payoffs that G's players derive from s.

There are k^n pure-strategy profiles in a game, and one could learn the game exhaustively using this many payoff queries. We are interested in algorithms that require only a small fraction of this trivial upper bound.

For a symmetric network congestion game, we assume that initially the algorithm only knows the number of players n, and the pure strategy space, given by the graph and the common source/destination pair.

Definition 1.2. A *payoff query* to a congestion game selects an assignment of at most n players to every pure strategy of the congestion game and learns the costs of the pure strategies with the assigned loads.

This query model removes the restriction that a query corresponds to a strategy profile with exactly n players. In this model, a query corresponds to a strategy profile with at most $n \times m$ players. Here, m denotes the number of resources in the congestion game, which, for network congestion games, is the number of edges. This query model is more powerful than the query model for strategic-form games, but it is very natural for congestion games. We prove both lower and upper bounds on the payoff query complexity of congestion games under this model.

Definition 1.3. The *payoff query complexity* of a class of games $\mathcal{G}$, with respect to some solution concept such as exact or approximate Nash equilibrium, is defined as follows. It is the smallest N such that there is some algorithm $\mathcal{A}$ that, given N payoff queries to any game $G \in \mathcal{G}$ (where initially none of the payoffs of G are known) can find a solution of G.

The definition imposes no computational bound on the algorithm $\mathcal{A}$. It is to some extent inspired by the work on query-based learning initiated by Angluin [Angluin 1987], in the context of computational learning theory. Note that $\mathcal{A}$ may select the queries in an on-line manner, so queries can depend on the responses to previous queries.

1.3. Overview of results

We study a variety of different settings. We start by considering bimatrix games, and our first result is a lower bound for computing an exact Nash equilibrium: computing an exact Nash equilibrium in a $k \times k$ bimatrix game has payoff query complexity k^2, even for zero-sum games. In other words, we have to query every pure strategy profile.

We then turn our attention to approximate Nash equilibria, where we obtain some more positive results. With the standard assumption that all payoffs lie in the range $[0, 1]$, we show that when ϵ lies in the range $\frac{1}{2} \le \epsilon < 1 - \frac{1}{k}$, the payoff query complexity to find an ϵ-Nash equilibrium lies in $[k, 2k - 1]$. When $\epsilon \ge 1 - \frac{1}{k}$, no payoff queries are needed at all, because an ϵ-Nash equilibrium is always achieved when both players play the uniform distribution over their strategies.

The query complexity of computing an approximate Nash equilibrium when $\epsilon < \frac{1}{2}$ appears to be a challenging problem, and we provide an initial lower bound in this direction. We show that for a $k \times k$ bimatrix game, the payoff query complexity of finding an ϵ-Nash equilibrium, for $\epsilon \le \frac{1}{8}$, is at least $k \cdot (\frac{1}{32/\log(k)+64\epsilon})$, i.e., for small ϵ this lower bound is $\Omega(k \cdot \log k)$. This gives an interesting contrast with the $\epsilon \ge \frac{1}{2}$ case. Whereas we can always compute a $\frac{1}{2}$-approximate Nash equilibrium using $O(k)$ payoff queries, there exists an $\epsilon < \frac{1}{2}$ for which this is not the case.

Having studied payoff query complexity in bimatrix games, it is then natural to look for improved payoff query complexity results in the context of "structured" games. In particular, we are interested in *concisely represented* games, where the payoff query complexity may be much smaller than the number of pure strategy profiles. As an initial result in this direction, we consider graphical games, where we show that for constant d the payoff query complexity of degree d graphical games is polynomial. This algorithm works by discovering every payoff in the game, however unlike bimatrix games, this can be done without querying every pure strategy profile.

Finally, we focus on two different models of congestion games. We first consider *parallel links*, where the game has a start and end vertex, and m different links between them. We show both lower and upper bounds for this model. If n denotes the number of players, then we obtain a $\log(n)$ payoff query lower bound, and a $\mathcal{O}\left(\log(n) \cdot \frac{\log^2(m)}{\log\log(m)}\right)$ payoff query upper bound. Note that there are $n \cdot m$ different payoffs in a parallel links game, and so our upper bound implies that you do not need to discover the entire payoff function in order to solve a parallel links game.

We also consider the more general case of symmetric network congestion games on directed acyclic graphs. We show that if the game has m edges and n players, then we can find a Nash equilibrium using $m \cdot n$ payoff queries. The algorithm discovers every payoff in the game, but it only queries a small fraction of the pure strategy profiles.

2. RELATED WORK

In the existing literature, there has so far been mainly experimental work on payoff queries. There is also another body of work, both theoretical and experimental, on best- and better-

response dynamics, which relevant because these dynamics generally work by exploring the space of pure profiles, and receiving feedback consisting of payoffs. The difference is that they purport to model a decentralised process of selfish behaviour by the players, while the payoff query model envisages a centralised algorithm that is less constrained. In this section, we give a survey the relevant literature.

2.1. Payoff queries

In *Empirical game-theoretic analysis* [Wellman 2006; Jordan et al. 2010], a game is presented to the analyst via a set of observations of strategy profiles (usually, pure) and their corresponding payoffs. This set of profiles/payoff-vector pairs is called an *empirical game*. In some settings the strategy profiles are randomly generated, but it is typically feasible to obtain observations via the payoff queries we study here. The *profile selection problem* [Jordan et al. 2008] is the challenge of choosing helpful strategy profiles. The *strategy exploration problem* [Jordan et al. 2010] is the special case of finding the best way to limit the search to a small subset of a large set of strategies.

Jordan et al. [Jordan et al. 2008] envisage a setting where a game (called a *base game*) has a corresponding *game simulator*, an implementation in software, which is amenable to payoff queries; a more general scenario allows the observed payoffs to be sampled from a distribution associated with the strategy profile. The distribution is sometimes considered to be due to a noise process, and called the *noisy payoff model* in [Jordan et al. 2008]. (In this paper we just consider deterministic payoffs, the "revealed payoff model" in [Jordan et al. 2008].) As noted in Vorobeychik et al. [Vorobeychik et al. 2007], a profile can be repeatedly queried to sample from the distribution of payoffs, and thus get an estimate of the expected values. The two interacting challenges are to identify helpful queries, and to use them to find pure-strategy profiles that have low regret (where *regret* refers to the largest incentive to deviate, amongst the players.)

Vorobeychik et al. [Vorobeychik et al. 2007] study the *payoff function approximation task*, in which a game belongs to a known class, and there is a "regression" challenge to determine certain parameters; the information about the game consists of a random sample of pure profiles and resulting payoff vectors. However, success is measured by the extent that the players' predicted behaviour is close to the behaviour associated with the true payoffs, rather than how well the true payoff functions are estimated.

Work on specific classes of multi-player games includes the following. Duong et al. [Duong et al. 2009] studies algorithms for learning graphical games; we consider a graphical game learning algorithm in Section 4. Jordan et al. [Jordan et al. 2008] apply payoff-query learning to various kinds of games generated by GAMUT [Nudelman et al. 2004], including a class of congestion games (Section 5.2). Vorobeychik et al. [Vorobeychik et al. 2007] investigate a first-price auction and also a scheduling game, where payoffs are described via a finite random sample of profile/payoff vector pairs. Earlier, Sureka and Wurman [Sureka and Wurman 2005] study search for pure Nash equilibria of strategic-form games (mostly with 5 players and 10 pure strategies).

Most of the experimental work (e.g. [Sureka and Wurman 2005; Jordan et al. 2008; Duong et al. 2009]) uses *local search*, in which profiles that get queried are typically very similar (differing in just one player's strategy) from previously queried profiles. Jordan et al. [Jordan et al. 2008] experiment with local-search type algorithms in which when a player has the incentive to deviate, the tested profile is updated with that deviation. Sureka and Wurman [Sureka and Wurman 2005] study search for pure equilibria via best-response dynamics while maintaining a tabu list, introduced to reduce the risk of cycles. In contrast, the algorithms we present here for congestion games exploit a non-local search approach, which results in improved performance.

2.2. Best-response dynamics and local search

There is a large body of literature that studies best- and better-response dynamics for classes of potential games, and gives bounds on the number of steps required for convergence to pure-strategy equilibria. In this literature, the dynamics are intended to model the behavior of players who make rational responses to other players, in a decentralized setting. The dynamics of [Even-Dar et al. 2003; Feldmann et al. 2003; Goldberg 2004; Chien and Sinclair 2011] are local search processes in that each pure profile is obtained from the previous one by letting a single player move. Bound on the convergence of deterministic best-response dynamics were considered in [Even-Dar et al. 2003; Feldmann et al. 2003]. The better-response dynamics considered by Goldberg [Goldberg 2004] is the basic randomized local search algorithm, and bounds are obtained for its convergence to exact equilibrium. Chien and Sinclair [Chien and Sinclair 2011] study another local search, the ϵ-*Nash dynamics*, and its convergence to approximate equilibria. Other papers, e.g. [Fischer et al. 2006; Berenbrink et al. 2007] analyse a strongly-distributed dynamics in which multiple players can move in the same time step; consequently the dynamics is not a local search. However, these dynamical systems could all be simulated by payoff query algorithms in which at each step, at most nk queries are made to determine the change in payoffs available to players as a result of unilateral deviations. This paper begins to answer the question: how much better could a payoff query algorithm do, if it were not subject to that constraint?

Finally, Alon et al. [Alon et al. 2011] consider payoff-query algorithms for finding the costs of paths in graphs. The consider *weight discovery protocols* where the aim is to determine the costs of edges, and *shortest path discovery protocols* where the aim is to find a shortest path. The latter objective is more similar to what we consider, since it can avoid the need to learn the entire payoff function; also a shortest path is an equilibrium strategy for the one-player case.

3. BIMATRIX GAMES

In this section, we give simple bounds on the payoff-query complexity of computing approximate Nash equilibria for bimatrix games. We assume that all payoffs lie in the range $[0, 1]$, which is a standard when finding approximate Nash equilibria.

Observation 1. The payoff query complexity of finding an exact Nash equilibrium of a $k \times k$ bimatrix game is k^2. This holds even for zero-sum games.

To see this, consider generalised matching pennies, where the column player pays 1 to the row player whenever both players choose the same strategy, otherwise the row player pays 1 to the column player.[1] This game has a unique Nash equilibrium, namely when both players randomize uniformly over their strategies. Now suppose each payoff in the game is perturbed by a small quantity; for a zero-sum game the perturbations should preserve the zero-sum property. For small perturbations, there will still be a unique fully-mixed solution, but it can only be known exactly if all the payoffs are known exactly. Observation 1 raises the question of whether better bounds exist for approximate Nash equilibria. We have the following result:

THEOREM 3.1. *For any* $\frac{1}{2} \leq \epsilon < 1 - \frac{1}{k}$, *the payoff query complexity to find an ϵ-approximate Nash equilibrium of a 2-player strategic-form $k \times k$ game, is in* $[k, 2k - 1]$.

PROOF. For the upper bound, we can simulate the algorithm of [Daskalakis et al. 2009b] as follows to obtain a $\frac{1}{2}$-approximate Nash equilibrium (which is then an ϵ-Nash equilibrium). Let s_1 be an arbitrary pure strategy of the row player. Query all k pure profiles where the row player plays s_1, which allows us to find the column player's best response

---

[1]By rescaling, we can make this game comply with our assumption that payoffs should lie in the range $[0, 1]$.

to s_1, call it s_2. Query the additional $k-1$ pure profiles where the column player plays s_2, from which we find the row player's best response to s_2; call it s_3. A $\frac{1}{2}$-approximate Nash equilibrium is obtained by letting the column player play pure strategy s_2 while the row player randomizes equally between s_1 and s_3.

For the lower bound, suppose that the row player has matrix R^ℓ for $\ell \in [n]$, where R^ℓ pays the row player 1 for playing row ℓ and 0 for any other row. Then in a ϵ-approximate Nash equilibrium, the row player must play row ℓ with probability $> \frac{1}{k}$. In order to identify ℓ, we need k queries. $\square$

For $\epsilon \geq 1 - \frac{1}{k}$, an ϵ-Nash equilibrium is obtained by letting both players play the uniform distribution over their strategies, and no payoff queries are needed. The interesting challenge, is to determine the payoff query complexity for constant values of $\epsilon < \frac{1}{2}$. For $\epsilon < \frac{1}{2}$ we have no non-trivial upper bound on the payoff query complexity of finding an ϵ-Nash equilibrium. We next show a *lower* bound (Theorem 3.5), which is increasing in k and ϵ^{-1}, and for large enough k, ϵ^{-1}, it can be seen to be higher than the upper bound for $\epsilon \geq \frac{1}{2}$. Thus, in an algorithm-independent sense, there are some positive values of ϵ, for which computing an ϵ-Nash equilibrium is harder than it is for other values of ϵ strictly less than one.

Let $\mathcal{G}_\ell$ be the class of strategic-form games where the column player has ℓ pure strategies and the row player has $\binom{\ell}{\ell/2}$ pure strategies (where we assume ℓ is even). Let $G_\ell \in \mathcal{G}_\ell$ be the win-lose constant-sum game in which each row of the row player's payoff matrix has $\frac{\ell}{2}$ 1's and $\frac{\ell}{2}$ 0's, all rows being distinct. The column player's payoffs are one minus the row player's payoffs. (G_ℓ is similar to the class of games used in Theorem 1 of [Feder et al. 2007].) Note that the value of G_ℓ is $\frac{1}{2}$ since either player can obtain payoff $\frac{1}{2}$ by using the uniform distribution over their pure strategies.

LEMMA 3.2. *Suppose that in game G_ℓ, the column player uses a mixed strategy in which some single pure strategy has probability $\alpha > 1/\ell$. Then the row player can obtain payoff $> \frac{1}{2} + \frac{\alpha}{2} - \frac{1}{2\ell}$.*

PROOF. Let j be a column that the column player plays with probability α. Let R_j be the set of rows where the row player obtains payoff 1 for column j. Suppose the row player plays the uniform distribution over rows in R_j. When the column player plays j, the row player receives payoff 1. Let $j' \neq j$ be a column, and consider the payoffs to the row player where j' intersects R_j. A fraction $\frac{\ell/2-1}{\ell-1}$ of these entries pay the row player 1, while a fraction $\frac{\ell/2}{\ell-1}$ pay the row player 0. Consequently whenever the column player plays $j' \neq j$, the row player's expected payoff is $\frac{\ell/2-1}{\ell-1}$. Thus with probability α the row player receives payoff 1, and with probability $1-\alpha$ he receives payoff $\frac{\ell/2-1}{\ell-1}$, from which the result follows, by straightforward manipulations. (In particular, the row player's payoff is $\alpha + (1-\alpha)\frac{\ell/2-1}{\ell-1}$ $= \frac{1}{2} + \frac{1}{2}\alpha - \frac{1-\alpha}{2(\ell-1)} > \frac{1}{2} + \frac{1}{2}\alpha - \frac{1-1/\ell}{2(\ell-1)} = \frac{1}{2} + \frac{1}{2}\alpha - \frac{1}{2\ell}$.) $\square$

COROLLARY 3.3. *Assume $\alpha > \frac{1}{k}$, and let $\epsilon = \frac{1}{4}(\alpha - \frac{1}{\ell})$. In any ϵ-Nash equilibrium of G_ℓ, the column player plays any individual column with probability at most α.*

PROOF. If, alternatively, the column player plays any column j with probability $> \alpha$, then by Lemma 3.2, the row player can obtain payoff $> \frac{1}{2} + \frac{\alpha}{2} - \frac{1}{2\ell}$. Hence in any ϵ-Nash equilibrium, the row player obtains payoff $> \frac{1}{2} + \frac{\alpha}{2} - \frac{1}{2\ell} - \epsilon = \frac{1}{2} + \epsilon$. Hence the column player obtains payoff $< \frac{1}{2} - \epsilon$. Since the value of G_ℓ is $\frac{1}{2}$, the column player's regret is $> \epsilon$, thus we do not have an ϵ-Nash equilibrium. $\square$

LEMMA 3.4. *For any $\epsilon < \frac{1}{8}$, $\ell \geq 8$, the payoff query complexity of finding an ϵ-Nash equilibrium for games in $\mathcal{G}_\ell$ is $\geq \binom{\ell}{\ell/2} \cdot \left(\frac{1}{16\epsilon+4/\ell}\right)$.*

PROOF. Let $\mathcal{A}$ be any payoff query learning algorithm; consider what happens when $\mathcal{A}$ is run on G_ℓ. Let S be the mixed strategy profile found by $\mathcal{A}$. For $\alpha = 4\epsilon + \frac{1}{\ell}$, by Corollary 3.3 no column is played with probability $> \alpha$, in S. We also know that in S, the row player's payoff is $\leq \frac{1}{2} + \epsilon$, since S is an ϵ-Nash equilibrium of a game with value $\frac{1}{2}$. Suppose for a contradiction that $\mathcal{A}$ made fewer than $\binom{\ell}{\ell/2} \cdot (\frac{1}{16\epsilon + 4/\ell})$ payoff queries. Then some row r received fewer than $(\frac{1}{16\epsilon + 4/\ell})$ queries.

If columns are selected at random from the column player's distribution in S, there is probability at most $\alpha(\frac{1}{16\epsilon + 4/\ell}) = \frac{1}{4}$ that row r contains payoffs that have already been queried by $\mathcal{A}$. Now suppose that we modify G_ℓ by replacing all un-queried entries of r with payoffs of 1 for the row player. $\mathcal{A}$ would output the same strategy profile S, but the payoff to the row player for playing r is $\geq \frac{3}{4}$. So the row player's regret is $\geq \frac{3}{4} - (\frac{1}{2} + \epsilon)$, that is, at least $\frac{1}{4} - \epsilon$. This gives us a contradiction, for $\epsilon < \frac{1}{8}$. □

THEOREM 3.5. *For $k \times k$ strategic-form games, the payoff query complexity of finding an ϵ-Nash equilibrium, for $\epsilon \leq \frac{1}{8}$, is at least $k \cdot (\frac{1}{32/\log(k) + 64\epsilon})$.*

PROOF. Let k' be the largest number of the form $\binom{\ell}{\ell/2}$ that is at most k. We have $k' \geq k/4$ and $\ell \geq \log(k)/2$. By Lemma 3.4, $\mathcal{G}_\ell$ has query complexity $\geq \binom{\ell}{\ell/2} \cdot (\frac{1}{16\epsilon + 4/\ell})$ to find an ϵ-Nash equilibrium. That is, $k'(\frac{1}{4/\ell + 16\epsilon}) \geq \frac{k}{4}(\frac{1}{4/\ell + 16\epsilon}) \geq \frac{k}{4}(\frac{1}{8/\log(k) + 16\epsilon}) = k(\frac{1}{32/\log(k) + 64\epsilon})$. A game G in $\mathcal{G}_\ell$ can be written down as a $k \times k$ game, by duplicating rows and columns, and approximate equilibria are preserved. □

Note that for small ϵ the lower bound given by Theorem 3.5 is $\Omega(k \cdot \log k)$.

4. GRAPHICAL GAMES

In this section, we give a simple payoff query-based algorithm for graphical games. In a *graphical game* [Kearns et al. 2001] we assume the players lie at the vertices of a degree-d graph, and a player's payoff is a function of the strategies of just himself and his neighbors. The number of payoff values needed to specify such a game is $n \cdot k^{d+1}$ which, in contrast with strategic-form games, is polynomial (assuming d is a constant).

Previously, Duong et al. [Duong et al. 2009] have carried out experimental work on payoff queries for graphical games. They compare a number of techniques; the algorithm we give here is polynomial-time but would likely be less efficient in practice. Similar to [Duong et al. 2009], we assume the underlying graph G is unknown, and we want to induce the structure of G, and corresponding payoffs.

THEOREM 4.1. *For constant d, payoff query complexity of degree d graphical games is polynomial.*

Algorithm 1 learns the entire payoff function with polynomially many queries.

There are a couple of important caveats regarding Theorem 4.1. First, although the payoff query complexity is polynomial, the computational complexity is (probably) not polynomial, since it is PPAD-complete to actually compute an approximate Nash equilibrium for graphical games [Daskalakis et al. 2009a]. Second, while Algorithm 1 avoids querying all (exponentially-many) pure-strategy profiles, it works in a brute-force manner that learns the entire payoff function. We prefer payoff query-based algorithms that allow a solution to be found without actually learning all the payoffs, as we achieve in Theorem 3.1, or the main result of Section 5.1.

5. CONGESTION GAMES

In this section, we give bounds on the payoff-query complexity of finding a pure Nash equilibrium in symmetric network congestion games. A congestion game is defined by a

---

Algorithm 1 GRAPHICALGAMES

1: Initialize graph G's vertices to be the player set, with no edges
2: Let S be the set of pure profiles in which at least $n - (d + 1)$ players play 1.
3: Query each element of S.
4: **for all** players p, p' **do**
5:     **if** $\exists s, s' \in S$ that differ only in p's payoff and p''s strategy **then**
6:         add directed edge (p, p') to graph
7:     **end if**
8: **end for**
9: **for all** players p **do**
10:     Let N_p be p's neighborhood in G
11:     Use elements of S to find p's payoffs as a function of strategies of N_p
12: **end for**

---

tuple $\Gamma = (N, E, (S_i)_{i \in N}, (f_e)_{e \in E})$. Here, $N = \{1, 2, \ldots, n\}$ is a set of n players and E is a set of resources. Each player chooses as her *strategy* a set $s_i \subseteq E$ from a given *set of available strategies* $S_i \subseteq 2^E$. Associated with each resource $e \in E$ is a non-negative, non-decreasing function $f_e : \mathbb{N} \mapsto \mathbb{R}^+$. These functions describe *costs* to be charged to the players for using resource e. An outcome (or strategy profile) is a choice of strategies $\mathsf{s} = (s_1, s_2, ..., s_n)$ by players with $s_i \in S_i$. For an outcome s define $n_e(\mathsf{s}) = |i \in N : e \in s_i|$ as the number of players that use resource e. The *cost* for player i is defined by $c_i(\mathsf{s}) = \sum_{e \in s_i} f_e(n_e(\mathsf{s}))$. A *pure Nash equilibrium* is an outcome s where no player has an incentive to deviate from her current strategy. Formally, s is a pure Nash equilibrium if for each player $i \in N$ and $s_i' \in S_i$, which is an alternative strategy for player i, we have $c_i(\mathsf{s}) \leq c_i(\mathsf{s}_{-i}, s_i')$. Here (s_{-i}, s_i') denotes the outcome that results when player i changes her strategy in s from s_i to s_i'.

In a *network congestion game*, resources correspond to the edges in a directed multigraph $G = (V, E)$. Each player i is assigned an origin node o_i, and a destination node d_i. A strategy for player i consists of a sequence of edges that form a directed path from o_i to d_i, and the strategy set S_i consists of all such paths. In a *symmetric* network congestion game all players have the same origin and destination nodes. We write a symmetric network congestion game as $\Gamma = (N, V, E, (f_e)_{e \in E}, o, d)$. Note that V, E, o, and d collectively define the strategy space $(S_i)_{i \in N}$. We consider two types of network, directed acyclic graphs, and the special case of parallel links. We assume that initially we only know the number of players n and the strategy space. The latency functions are completely unknown initially. A payoff query presents an assignment of at most n players to every strategy of the congestion game and learns the costs of strategies with the assigned loads. As a shorthand for defining strategy profiles, we use notation of the form $\mathsf{s} \leftarrow (1 \mapsto p, 3 \mapsto q)$. This example defines s to be a four-player strategy profile that assigns 1 player to p and 3 players to q, where p and q will be paths from source to sink in a symmetric network congestion game. We use Query(s) to denote a query with an assignment s of at most n players to the pure strategies. It returns a function c_{s}, which gives the cost of each strategy when s is played.

5.1. Parallel links

In this section, we consider congestion games on m parallel links. A simple construction with two parallel links shows a lower bound of $\log n$. The essence of the construction is as follows. Link 1 has a constant cost of 1 for any number of players. Link 2 has a cost of 0 which switches at some point to 2. To find a pure equilibrium a querier must find this switch, and an adversary has a simple strategy to ensure that the querier can do no better than binary search. In the rest of the section, we provide a payoff query algorithm that finds a pure Nash equilibrium using $\mathcal{O}\left(\log(n) \cdot \frac{\log^2(m)}{\log \log(m)}\right)$ queries.

Our algorithm works in phases. In each phase we move players only in some multiple of $\delta = k^t$, for some integer parameter k, and we determine an equilibrium with respect to this group size δ. To find out how many groups to move between the links we perform a double binary search: In the first binary search, we guess the number of moved groups, which will give us a target cost value. We use this target value in a second binary search to check whether our guess on the number of groups to move was correct. In each subsequent iteration δ becomes smaller by a factor k until in the last iteration $\delta = 1$ and a Nash equilibrium is found.

The algorithm (PARALLELLINKS) is depicted in Algorithm 2. We will show how this algorithm can be implemented with the stated number of queries. The algorithm is inspired by an algorithm from [Gairing et al. 2008].

Throughout the algorithm there will be special link a. For a congestion game Γ, an integer δ, and a special link a we define a δ-equilibrium as follows:

Definition 5.1 (δ-*equilibrium*). A strategy profile s is δ-equilibrium if $\delta | n_i(\mathsf{s})$ for all $i \in [m] \setminus \{a\}$, and for all links $i, j \in [m]$ with $n_i(\mathsf{s}) \geq \delta$ we have $f_i(n_i(\mathsf{s})) \leq f_j(n_j(\mathsf{s}) + \delta)$.

Intuitively, we can think of a δ-equilibrium s as a Nash equilibrium in a transformed game where the players (of the original game) are partitioned into groups of size δ and each group represents a player in the transformed game, and the remaining ($n \bmod \delta$) players are fixed to link a.

We start with an informal description of algorithm PARALLELLINKS. The algorithm is parameterised by an integer $k \geq 2$. It starts by finding a n-equilibrium s by putting all players together on the link a that minimises the induced cost. Observe that by Definition 5.1, s is also a δ-equilibrium for any $\delta \geq n$. We can find a with a single payoff query, by querying for n players on each link. Note, that throughout the algorithm, link a will take a special role since the number of players assigned to any other link will always be a multiple of δ. The algorithm then works in $T + 1$ phases, where $T = \lfloor \frac{\log(n)}{\log(k)} \rfloor$. Each phase is one iteration of the **for**-loop. The **for**-loop is governed by a variable t, which is initially T and decreases by 1 in each iteration. Within any iteration, the algorithm uses the function REFINEPROFILE to transform a k^{t+1}-equilibrium into a k^t-equilibrium. We can bound the number of reassignments of groups of players to different links needed to achieve this by the following lemma:

LEMMA 5.2. *We can convert a k^{t+1}-equilibrium s into a k^t-equilibrium s' by moving at most $2k$ groups of $\delta = k^t$ players to any individual link and at most km groups of δ players in total.*

PROOF. Since s is k^{t+1}-equilibrium, we have $f_i(n_i(\mathsf{s})) \leq f_j(n_j(\mathsf{s}) + k^{t+1})$ for all $i \in [m] \setminus \{a\}, j \in [m]$. Moreover, either (a) $f_a(n_a(\mathsf{s})) \leq f_j(n_j(\mathsf{s}) + k^{t+1})$ for all $j \in [m]$ or (b) $n_a(\mathsf{s}) < k^{t+1}$. In case (a), this implies that each link $j \in [m]$ can in total receive at most k groups of size $\delta = k^t$ from links $i \in [m]$. In case (b), this implies that each link $j \in [m]$ can in total receive at most k groups of size $\delta = k^t$ from links $i \in [m] \setminus \{a\}$. Moreover, since $n_a(\mathsf{s}') < k^{t+1}$, we can move at most k groups of size $\delta = k^t$ from link a. In any case, in total we move at most km groups. All links receive and loose players only in multiples of $\delta = k^t$ which ensures that $k^t | n_i(\mathsf{s}')$ for all $i \in [m] \setminus \{a\}$ is maintained. $\square$

Refining the profile. Our algorithm uses the function REFINEPROFILE to implement the refinement as described in Lemma 5.2. REFINEPROFILE determines the number of groups q which have to be moved by binary search on q in $[0, km]$. Since by Lemma 5.2 each link receives at most $2k$ groups of players, we spend $2k$ payoff queries to determine for all links $i \in [m]$, the cost function values $f_i(n_i(\mathsf{s}) + r \cdot \delta)$ for all integers $r \leq 2k$. We define Q as the multi-set of these cost function values and $C_{min}(q)$ as the $(q + 1)$ smallest value in Q. Intuitively, $C_{min}(q)$ is the cost of the $(q + 1)$-th group of players that we would move. We

---

Algorithm 2 PARALLELLINKS

1: $a \leftarrow \arg\min_{i \in [m]} f_i(n)$ ▷ 1 query
2: initialize strategy profile s by putting all players on link a
3: $T \leftarrow \lfloor \frac{\log(n)}{\log(k)} \rfloor$
4: **for** $t = T, T-1, \ldots, 1, 0$ **do**
5:      $\delta \leftarrow k^t$
6:      s $\leftarrow$ REFINEPROFILE$(\mathsf{s}, \delta, 0, km)$
7: **end for**
8: **return** s

9: **function** REFINEPROFILE$(\mathsf{s}, \delta, q_{min}, q_{max})$
10:      $q \leftarrow \lfloor \frac{q_{min}+q_{max}}{2} \rfloor$
11:      **Parallel** for all links $i \in [m]$
12:          Query for costs $f_i(n_i(\mathsf{s}) + r\delta)$ for all integer $1 \le r \le 2k$ ▷ $2k$ queries
13:      **EndParallel**
14:      $Q \leftarrow$ the ordered multiset of $2km$ non-decreasing costs from the above queries
15:      $C_{min}(q) \leftarrow (q+1)$-th smallest element of Q
16:      $p_i \leftarrow$ number of times $i \in [m]$ contributes a cost to the q smallest elements of Q
17:      **Parallel** for all links $i \in [m]$
18:          **if** $f_i(n_i(\mathsf{s}) - \lfloor \frac{n_i(\mathsf{s})}{\delta} \rfloor \cdot \delta) > C_{min}(q)$ **then**   ▷ 1 query; only relevant for link a
19:              $q_i \leftarrow \lfloor \frac{n_i(\mathsf{s})}{\delta} \rfloor$
20:          **else** (using binary search on $q_i \in [0, \min\{km, \lfloor \frac{n_i(\mathsf{s})}{\delta} \rfloor\}]$)
21:              $q_i \leftarrow \min\{q_i : f_i(n_i(\mathsf{s}) - q_i\delta) \le C_{min}(q)\}$ ▷ $\log(km)$ queries
22:          **end if**
23:      **EndParallel**
24:      **if** $\sum_{i \in [n]} q_i = q$ **then**
25:          modify s by removing q_i and adding p_i groups of δ players to every link $i \in [m]$
26:          **return** s
27:      **else if** $\sum_{i \in [n]} q_i < q$ **then**
28:          **return** REFINEPROFILE$(\mathsf{s}, \delta, q_{min}, q-1)$
29:      **else** $(\sum_{i \in [n]} q_i > q)$
30:          **return** REFINEPROFILE$(\mathsf{s}, \delta, q+1, q_{max})$
31:      **end if**
32: **end function**

---

use $C_{min}(q)$ to find out how many groups of players q_i we need to remove from each link $i \in [m]$ so that on each link $i \in [m]$ the cost is at most $C_{min}(q)$ or we can't remove any further groups as there are less than δ players assigned to it (which can only happen on link a). By Lemma 5.2, we need to remove at most km groups of players in total. Therefore, we can determine $q_i \in [0, \min\{km, \lfloor \frac{n_i(\mathsf{s})}{\delta} \rfloor\}]$ by binary search in parallel on all links, with $\mathcal{O}(\log(km))$ payoff queries. Now, if $\sum_{i=1}^{n} q_i = q$, we can construct a k^t-equilibrium by removing q_i and adding p_i groups of δ players to link $i \in [m]$; note that for every $i \in [m]$, either $q_i = 0$ or $p_i = 0$. If $\sum_{i=1}^{n} q_i \ne q$, our guess for q was not correct and we have to continue the binary search on q.

The algorithm maintains the following invariant:

LEMMA 5.3. REFINEPROFILE$(\mathsf{s}, \delta, 0, km)$ *returns a δ-equilibrium.*

PROOF. Observe that $\delta = k^t$. In the first iteration of the **for**-loop $t = T$ and REFINEPRO-FILE($\mathsf{s}, \delta, 0, km$) gets a n-equilibrium as input, which is also a k^{T+1}-equilibrium as all players are assigned to link a and $k^{T+1} > n$. So to proof the claim, it suffices to show that REFINE-PROFILE($\mathsf{s}, k^t, 0, km$) returns a k^t-equilibrium if s is a k^{t+1}-equilibrium. For the s returned by REFINEPROFILE and the q in its returning call, we have $f_i(n_i(\mathsf{s})) \leq C_{min}(q) \leq f_i(n_i(\mathsf{s}) + \delta)$ for all $i \in [m] \setminus \{a\}$. The left inequality follows from line 21 of the algorithm. The right inequality follows from the definition of $C_{min}(q)$ as the $(q+1)$-th smallest element in Q in line 15 of the algorithm. For link a, we have $f_a(n_a(\mathsf{s})) \leq C_{min}(q) \leq f_a(n_a(\mathsf{s}) + \delta)$ or we have $f_a(n_a(\mathsf{s})) > C_{min}(q)$ and $n_a(\mathsf{s}) < \delta$, where the first case follows from lines 21 and 15 as before, and the second case corresponds to line 18. Noting that REFINEPROFILE maintains that for the returned s we have $\delta | n_i(\mathsf{s})$ for all $i \in [m] \setminus \{a\}$, as it only moves groups of size δ, the claim follows. $\square$

LEMMA 5.4. REFINEPROFILE($\mathsf{s}, \delta, 0, km$) *makes* $O(\log(km)(k + \log(km)))$ *queries.*

PROOF. For each value of q in the binary search, we make $\mathcal{O}(k)$ payoff queries to determine $C_{min}(q)$ and $\mathcal{O}(\log(km))$ payoff queries to determine the q_i's in parallel for all links $i \in [m]$. The binary search on q adds a factor $\log(km)$. $\square$

THEOREM 5.5. *Algorithm* PARALLELLINKS *returns a pure Nash equilibrium and can be implemented with* $\mathcal{O}\left(\log(n) \cdot \frac{\log^2(m)}{\log\log(m)}\right)$ *payoff queries.*

PROOF. In the last iteration of the **for**-loop, we have $\delta = 1$, so Lemma 5.3 implies that s is a pure Nash equilibrium. To find the best link in line 1 of the algorithm, we need one payoff query. For any $k \geq 2$, the algorithm does $T+1 = \mathcal{O}\left(\frac{\log(n)}{\log(k)}\right)$ iterations of the **for**-loop. In each iteration we do $O(\log(km)(k + \log(km)))$ payoff queries. Choosing $k = \Theta(\log(m))$ yields the stated upper bound. $\square$

5.2. Symmetric Network Congestion Games on DAGs

In this section, we consider symmetric network congestion games on directed acyclic graphs. Throughout this section, we will consider the game $\Gamma = (N, V, E, (f_e)_{e \in E}, o, d)$, where (V, E) is a DAG. We use the $\prec$ relation to denote a topological ordering over the vertices in V. We assume that, for every vertex $v \in V$, there exists a path from o to v, and there exists a path from v to d. If either of these conditions does not hold for some vertex v, then v cannot appear on an o-d path, and so it is safe to delete v.
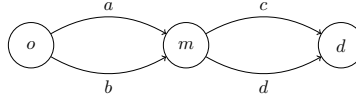
Preprocessing. Before we present the algorithm, we describe a required preprocessing step. We say that edges e and e' are *dependent* if visiting one implies that we must visit the other. More formally, e and e' are dependent if, for every o-d path p, we either have $e, e' \in p$, or we have $e, e' \notin p$. We preprocess the game to ensure that there are no pairs of dependent edges. To do this, we check every pair of edges e and e', and test whether they are dependent. If they are, then we *contract* e', i.e., if $e' = (v, u)$, then we delete e', and set $v = u$. The following lemma shows that this preprocessing is valid.

LEMMA 5.6. *There is an algorithm that, given a congestion game Γ, where (V, E) is a DAG, produces a game Γ' with no pair of dependent edges, such that every Nash equilibrium of Γ' can be converted to a Nash equilibrium of Γ. The algorithm and conversion of equilibria take polynomial time and make zero payoff queries. Moreover, payoff queries to Γ' can be trivially simulated with payoff queries Γ.*

Thus, we assume that our congestion game contains no pair of dependent edges.

Equivalent cost functions. Our approach is to devise a querying strategy to determine the cost function of each edge. That is, for each $e \in E$, we give an algorithm to discover

$f_e(i)$ for all i. One immediate observation is that we can never hope to find the actual cost functions. Consider the following one-player congestion game.

If we set $f_a(1) = f_b(1) = 1$ and $f_c(1) = f_d(1) = 0$, then all o-d paths have cost 1. However, we could also achieve the same property by setting $f_a(1) = f_b(1) = 0$ and setting $f_c(1) = f_d(1) = 1$. Thus, it is impossible to learn the actual cost functions using payoff queries. Fortunately, we do not need to learn the actual cost function in order to solve the congestion game. We define two cost functions to be equivalent if they assign the same cost to every strategy profile.

Definition 5.7 (*Equivalence*). Two cost functions f and f' are equivalent if for every strategy profile $\mathsf{s} = (s_1, s_2, \ldots, s_n)$, we have $\sum_{e \in s_i} f_e(n_e(\mathsf{s})) = \sum_{e \in s_i} f'_e(n_i(\mathsf{s}))$, for all i.

Clearly, the Nash equilibria of a game cannot change if we replace its cost function f with an equivalent cost function f'. We give an algorithm that constructs a cost function f' that is equivalent to f. We say that $(f'_e)_{e \in E}$ is a partial cost function if for some $e \in E$ and some $i \le n$, $f'_e(i)$ is undefined. We say that f'' is an extension of f' if f'' is a partial cost function, and if $f''_e(i) = f'_e(i)$ for every $e \in E$ and $i \le n$ for which $f'_e(i)$ is defined. We say that f'' is a total extension of f' if f'' is an extension of f', and if $f''_e(i)$ is defined for all $e \in E$ and all $i \le n$.

Definition 5.8 (*Partial equivalent cost function*). Let f be a cost function. We say that f' is a partial equivalent of f if f' is a partial cost function, and if there exists a total extension f'' of f' such that f'' is equivalent to f.

Our algorithm will begin with a partial cost function f^0 such that $f_e^0(i)$ is undefined for all $e \in E$ and all $i \le n$. Clearly f^0 is a partial equivalent of f. It then constructs a sequence of partial cost functions $f^1, f^2, \ldots$, where each f^a is an extension of f^{a-1}, and each f^a is a partial equivalent of f. Therefore, the algorithm will eventually arrive at a total cost function f^a that is equivalent to the original cost function f.

The algorithm proceeds inductively. It begins by constructing a partial equivalent cost function f^a such that $f_e^a(1)$ is defined for every edge $e \in E$. Then, in each subsequent step, it takes a partial equivalent cost function f^a such that $f_e^a(j)$ is defined whenever $j \le i$, and it constructs a partial equivalent cost function $f^{a'}$, where $f_e^{a'}(j)$ is defined whenever $j \le i + 1$.

5.3. Symmetric Network Congestion Games on DAGs: The one-player case

We begin by describing a method that computes a partial equivalent cost function f^a such that $f_e^a(1)$ is defined for every edge $e \in E$. The algorithm begins with the partial cost function f^0. The algorithm processes vertices according to the topological ordering $\preceq$. When the algorithm processes a vertex $k \in V$, it begins with a partial equivalent cost function f^a such that $f_e^a(1)$ is defined for every edge $e = (v, u)$ with $u \prec k$, for some vertex k. It then produces a partial equivalent cost function f^{a+1} such that $f_e^a(1)$ is defined for every edge $e = (v, u)$ with $u \preceq k$. There are two cases to consider, depending on whether $k = d$ or not.

The $k \ne d$ case. We use the procedure shown in Algorithm 3 to process k. Lines 1 through 3 simply copy the old cost function f^a into the new cost function f^{a+1}. This ensures that f^{a+1} is an extension of f^a. The algorithm then picks an arbitrary k-d path p. The loop on lines 5 through 10 compute the function t, which for each incoming edge (e, k), gives the cost of allocating one player to ep. Note, in particular, that the value of the

---

Algorithm 3 PROCESSK

Input: A partial equivalent cost function f^a, such that $f^a_e(1)$ is defined for all edges (v, u) with $u \prec k$.

Output: A partial equivalent cost function f^{a+1}, such that $f^a_e(1)$ is defined for all edges (v, u) with $u \preceq k$.

1: **for all** e for which $f^a_e(1)$ is defined **do**
2:     $f^{a+1}_e(1) \leftarrow f^a_e(1)$
3: **end for**
4: $p \leftarrow$ an arbitrary k-d path
5: **for all** $e = (v, k) \in E$ **do**
6:     $p' \leftarrow$ an arbitrary o-v path
7:     $\mathsf{s} \leftarrow (1 \mapsto p'ep)$
8:     $c_\mathsf{s} \leftarrow \mathrm{Query}(\mathsf{s})$
9:     $t(ep) \leftarrow c_\mathsf{s}(p'ep) - \sum_{e' \in p'} f^a_{e'}(1)$
10: **end for**
11: $m \leftarrow$ edge $e = (v, k)$ that minimises $t(ep)$
12: $f^{a+1}_m(1) \leftarrow 0$
13: **for all** $e = (v, k) \in E$ with $e \neq m$ **do**
14:     $f^{a+1}_e(1) \leftarrow t(ep) - t(mp)$
15: **end for**

---

expression $\sum_{e' \in p'} f^a_{e'}(1)$ is known to the algorithm, because every vertex visited by p' has already been processed. The algorithm then selects m to be the edge that minimises t, and sets the cost of m to be 0. Once it has done this, lines 13 through 15 compute the costs of the other edges relative to m.

When we set the cost of m to be 0, we are making use of equivalence. Suppose that the actual cost of m is c_m. Setting the cost of m to be 0 has the following effects:

— Every incoming edge at k has its cost reduced by c_m.
— Every outgoing edge at k has its cost increased by c_m.

This maintains equivalence with the original cost function, because for every path p that passes through k, the total cost of p remains unchanged. The following lemma formalises this and proves that f^{a+1} is indeed a partial equivalent cost function.

LEMMA 5.9. *Let $k \neq d$ be a vertex, and let f^a be a partial equivalent cost function such that $f^a_e(1)$ is defined for all edges $e = (v, u)$ with $u \prec k$. When given these inputs, Algorithm 3 computes a partial equivalent cost function f^{a+1} such that $f^{a+1}_e(1)$ is defined for all edges $e = (v, u)$ with $u \preceq k$.*

The $k = d$ case. When the algorithm processes d, it will have a partial cost function f^a such that $f^a_e(1)$ is defined for every edge $e = (v, u)$ with $u \neq d$. The algorithm is required to produce a partial cost function f^{a+1} such that $f^a_e(1)$ is defined for all $e \in E$. We use Algorithm 4 to do this. Lines 1 through 3 ensure that f^{a+1} is equivalent to f^a. Then, the algorithm loops through each incoming edge $e = (v, d)$, and line 8 computes $f^{a+1}_e(1)$. Note, in particular, that $f^a_{e'}(1)$ is defined for every edge $e' \in p$, and thus the computation on line 8 can be performed. Lemma 5.10 shows that Algorithm 4 is correct.

LEMMA 5.10. *Let $k \neq d$ be a vertex, and let f^a be a partial equivalent cost function defined for all edges (v, u) with $u \prec d$. When given these inputs, Algorithm 4 computes a partial equivalent cost function f^{a+1}.*

Query complexity. The algorithm makes exactly $|E|$ payoff queries in order to find the one-player costs. When Algorithm 3 processes a vertex k, it makes exactly one query for

---

Algorithm 4 PROCESSD

Input: A partial equivalent cost function f^a, such that $f_e^a(1)$ is defined for all edges $e = (v, u)$ with $u \prec d$.

Output: A partial equivalent cost function f^{a+1}, such that $f_e^a(1)$ is defined for all edges $e \in E$.

1: **for all** e for which $f_e^a(1)$ is defined **do**
2:      $f_e^{a+1}(1) \leftarrow f_e^a(1)$
3: **end for**
4: **for all** $e = (v, d) \in E$ **do**
5:      $p \leftarrow$ an arbitrary o-v path
6:      $\mathsf{s} \leftarrow (1 \mapsto pe)$
7:      $c_\mathsf{s} \leftarrow \text{Query}(\mathsf{s})$
8:      $f_e^{a+1}(1) \leftarrow c_\mathsf{s}(pe) - \sum_{e' \in p} f_{e'}^a(1)$
9: **end for**

---

each incoming edge (v, k) at k. The same property holds for Algorithm 4. This implies that, in total, the algorithm makes $|E|$ queries.

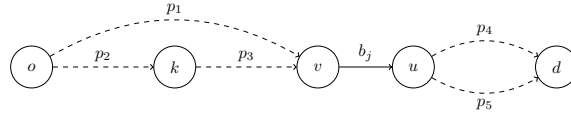**5.4. Symmetric Network Congestion Games on DAGs: Many-player games**

We now assume that we have a partial equivalent cost function f^a such that $f_e^a(j)$ is defined whenever $j \leq i$. We give an algorithm to produce a partial cost function $f^{a'}$, such that $f_e^a(j)$ is defined whenever $j \leq i + 1$.

We will proceed as in the one-player case, by processing vertices according to their topological order. The algorithm is complicated by *bridges*. An edge e is a bridge between two vertices v and u, if every v-u path contains e. Furthermore, if we fix a vertex $k \in V$, then we say that an edge e is a k-bridge if e is a bridge between k-d. The following lemma can be proved using the max-flow min-cut theorem.

LEMMA 5.11. *Let v and u be two vertices. There are two edge disjoint paths between v and u if, and only if, there is no bridge between v and u.*

Bridges. Given a vertex k, we show how to determine the cost of the k-bridges. Let b_1, $b_2, \ldots, b_m$ denote the list of k-bridges sorted according to the topological ordering $\preceq$. That is, if $b_1 = (v_1, u_1)$, and $b_2 = (v_2, u_2)$, then we have $v_1 \prec v_2$, and so on. Our algorithm is given a partial cost function f^a, such that $f_e^a(j)$ is defined for all $j \leq i$, and returns a cost function f^{a+1} that is an extension of f^a where, for all l, we have that $f_{b_l}^{a+1}(i+1)$ is defined.

Our algorithm processes the k-bridges in reverse topological order, starting with the final bridge b_m. Suppose that we are processing the bridge $b_j = (v, u)$. We will make one payoff query to find the cost of b_j, which is described by the following diagram.

The dashed lines in the diagram represent paths. They must satisfy some special requirements, which we now describe. The paths p_4 and p_5 must be edge disjoint, apart from k-bridges. The following lemma shows that we can always select two such paths.

LEMMA 5.12. *For each k-bridge $b_j = (v, u)$, there exists two paths p_4 and p_5 from u to d such that $p_4 \cap p_5 = \{b_{j+1}, b_{j+2}, \ldots b_m\}$.*

On the other hand, the paths p_1, p_2, and p_3 must satisfy a different set of constraints, which are formalised by the following lemma.

LEMMA 5.13. *Let $b_j = (v, u)$ be a k-bridge, let p_2 be an arbitrarily chosen o-k path. There exists an o-k path p_1 and a k-v path p_3 such that: p_1 and p_3 are edge disjoint; and if p_1 visits k, then p_2 and p_1 use different incoming edges for k.*

Algorithm 5 FINDKBRIDGES(k)

Input: A vertex k, and a partial equivalent cost function f^a, such that $f_e^a(j)$ is defined for
every $j \leq i$.
Output: A partial equivalent cost function f^{a+1}, such that f^{a+1} is an extension of f^a, and
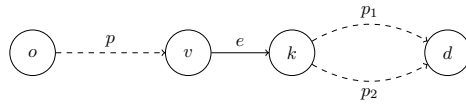f_e^{a+1} is defined for every e that is a k bridge.
1: **for all** e and j for which $f_e^a(j)$ is defined **do**
2:     $f_e^{a+1}(j) \leftarrow f_e^a(j)$
3: **end for**
4: **for** $j = $ m to 1 **do**
5:     $p_4, p_5 \leftarrow$ paths chosen according to Lemma 5.12
6:     $p_1, p_2, p_3 \leftarrow$ paths chosen according to Lemma 5.13
7:     $\mathsf{s} \leftarrow (1 \mapsto p_1 b_j p_4, i \mapsto p_2 p_3 b_j p_5)$
8:     $c_{\mathsf{s}} \leftarrow$ Query(s)
9:     $f_{b_j}^{a+1}(i+1) \leftarrow c_{\mathsf{s}}(p_1 b_j p_4) - \sum_{e \in p_1} f_e^{a+1}(n_e(\mathsf{s})) - \sum_{e \in p_4} f_e^{a+1}(n_e(\mathsf{s}))$
10: **end for**

Algorithm 5 shows how the cost of placing $i + 1$ players on each of the k-bridges can be discovered. Note that on line 9, since s assigns one player to p_1, we have $n_e(\mathsf{s}) = 1$ for every $e \in p_1$. Therefore, $f_e^{a+1}(n_e(\mathsf{s}))$ is known for every edge $e \in p_1$. Moreover, for every edge $e \in p_4$, we have that $n_e(\mathsf{s}) = i + 1$ if e is a k-bridge, and we have $n_e(\mathsf{s}) = 1$, otherwise. Since the algorithm processes the k-bridges in reverse order, we have that $f_e^{a+1}(n_e(\mathsf{s}))$ is defined for every edge $e \in p_4$. The following lemma shows that line 9 correctly computes the cost of b_j.

LEMMA 5.14. *Let k be a vertex, and let f^a be a partial equivalent cost function, such that $f_e^a(j)$ is defined for every $j \leq i$. Algorithm 5 computes a partial equivalent cost function f^{a+1}, such that f^{a+1} is an extension of f^a, and f_e^{a+1} is defined for every e that is a k-bridge.*

Incoming edges of k. We now describe the second part of the many-player case. After finding the cost of each k-bridge, we find the cost of each incoming edge at k. The following diagram describes how we find the cost of $e = (v, k)$, an incoming edge at k .

The path p is an arbitrarily chosen path from o to v. The paths p_1 and p_2 are chosen according to the following lemma.

LEMMA 5.15. *There exist two k-d paths p_1, p_2 such that every edge in $p_1 \cap p_2$ is a k-bridge.*

Algorithm 6 shows how we find the cost of putting $i + 1$ players on each edge e that is incoming at k. Consider line 9. Note that every vertex in p is processed before k is processed, and therefore $f_{e'}^{a+1}(i+1)$ is known for every $e' \in p$. Moreover, for every edge $e' \in p_1$, we have that $n_{e'}(\mathsf{s}) = i + 1$ if e' is a k-bridge, and we have $n_{e'}(\mathsf{s}) = 1$ otherwise. In either case, the $f_{e'}^{a+1}(n_{e'}(\mathsf{s}))$ is known for every edge $e' \in p_1$. The following lemma show that line 9 correctly computes $f_e^{a+1}(i+1)$.

411

Algorithm 6 MULTIPROCESSK

Input: A vertex k, and a partial equivalent cost function f^a, such that $f_e^a(j)$ is defined for
  all $e \in E$ when $j \leq i$, all $e = (v, u)$ with $u \prec k$ when $j = i + 1$, and all k-bridges when
  $j = i + 1$.

Output: A partial equivalent cost function f^a, such that $f_e^a(j)$ is defined for all $e \in E$
  when $j \leq i$, and for all $e = (v, u)$ with $u \preceq k$ when $j = i + 1$.

  1: **for all** e and j for which $f_e^a(j)$ is defined **do**
  2:     $f_e^{a+1}(j) \leftarrow f_e^a(j)$
  3: **end for**
  4: **for all** $e = (v, k) \in E$ **do**
  5:     $p \leftarrow$ an arbitrary o-v path
  6:     p_1, p_2 paths chosen according to Lemma 5.15
  7:     $\mathsf{s} \leftarrow (1 \mapsto pep_1, i \mapsto pep_3)$
  8:     $c_\mathsf{s} \leftarrow \mathrm{Query}(\mathsf{s})$
  9:     $f_e^{a+1}(i + 1) \leftarrow c_\mathsf{s}(pep_1) - \sum_{e' \in p} f_{e'}^{a+1}(n_{e'}(\mathsf{s})) - \sum_{e' \in p_1} f_{e'}^{a+1}(n_{e'}(\mathsf{s}))$.
 10: **end for**

LEMMA 5.16. *Let k be a vertex, and let f^a be a partial equivalent cost function, such
that $f_e^a(j)$ is defined for all $e \in E$ when $j \leq i$, all $e = (v, u)$ with $u \prec k$ when $j = i + 1$,
and all k-bridges when $j = i + 1$. Algorithm 6 produces a partial equivalent cost function
f^a, such that $f_e^a(j)$ is defined for all $e \in E$ when $j \leq i$, and for all $e = (v, u)$ with $u \preceq k$
when $j = i + 1$.*

Query complexity. We argue that the algorithm can be implemented so that the costs
for $(i + 1)$ players can be discovered using at most $|E|$ many payoff queries. Every time
Algorithm 5 discovers the cost of placing $i + 1$ players on a k-bridge, it makes exactly one
payoff query. Every time Algorithm 6 discovers the cost of an incoming edge (v, k), it makes
exactly one payoff query. The key observation is that the costs discovered by Algorithm 5
do not need to be rediscovered by Algorithm 6. That is, we can modify Algorithm 6 so
that it ignores every incoming edge (v, k) that has already been processed by Algorithm 5.
This modification ensures that the algorithm uses precisely $|E|$ payoff queries to discover
the edge costs for $i + 1$ players.

6. CONCLUSIONS AND FURTHER WORK

We first consider open questions in the setting of payoff queries, which has been the main
setting for the results in this paper. We then consider alternative query models.

Open questions concerning payoff queries. In the context of strategic-form games, various
questions have been raised by our results. Theorem 3.5 gives a lower bound on the payoff
query complexity of computing an ϵ-Nash equilibrium of an $k \times k$ bimatrix game in terms of
k, for small positive ϵ; what about for larger $\epsilon < \frac{1}{2}$? What about a non-trivial upper bound
for $\epsilon < \frac{1}{2}$? Also, what is the exact expression for the payoff query complexity of bimatrix
games when $\epsilon \geq \frac{1}{2}$? It would also be interesting to investigate the payoff query complexity of
finding an ϵ-Nash equilibrium in an n-player strategic-form game for small $n > 2$. However,
for polynomial-time algorithms only a weak upper bound of $\epsilon \leq 1 - \frac{1}{n}$ is known [Briest
et al. 2008; Hémon et al. 2008]. To achieve a better ϵ with only polynomially-many queries,
either non-polynomial-time algorithms would need to be used, or the $1 - \frac{1}{n}$ upper bound
would need to be improved.

For congestion games, our lower bound of $\log n$ arises from a game with two parallel
links. The upper bound is only a poly-logarithmic factor off from this lower bound, with
the factor depending on m and not n. Thus a lower bound construction that also depends
on m would be interesting. For DAGs it is unclear whether the payoff query complexity

is sub-linear in n. Non-trivial lower and upper bounds for more general settings, such as asymmetric network congestion games (DAG or not) or general (non-network) congestion games would also be interesting.

Other query models. We have defined a payoff query as given by a *pure* (not mixed) profile s, since that is of main relevance to empirical game-theoretic modelling. Furthermore, if s was a mixed profile, it could be simulated by sampling a number of pure profiles from s and making the corresponding sequence of pure payoff queries. An alternative definition might require a payoff query to just report a single specified player's payoff, but that would change the query complexity by a factor at most n.

Our main results have related to exact payoff queries, though other query models are interesting too. A very natural type of query is a *best-response query*, where a strategy s is chosen, and the algorithm is told the players' best responses to s. In general s may have to be a mixed strategy; it is not hard to check that pure-strategy best response queries are insufficient; even for a two-player two-action game, knowledge of the best responses to pure profiles is not sufficient to identify an ϵ-Nash equilibrium for $\epsilon < \frac{1}{2}$. Fictitious Play ([Fudenberg and Levine 1998], Chapter 2) can be regarded as a query protocol that uses best-response queries (to mixed strategies) to find a Nash equilibrium in zero-sum games, and essentially a 1/2-Nash equilibrium in general-sum games [Goldberg et al. 2013]. We can always synthesize a pure best-response query with $n(k - 1)$ payoff queries. Hence, for questions of polynomial query complexity, payoff queries are at least as powerful as best-response queries. Are there games where best-response queries are much more useful than payoff queries? If k is large then it is expensive to synthesize best-response queries with payoff queries. The DMP-algorithm [Daskalakis et al. 2009b] finds a $\frac{1}{2}$-Nash equilibrium via only two best-response queries, whereas Theorem 3.1 notes that $\Theta(k)$ payoff queries are needed.

A *noisy* payoff query outputs an observation of a random variable taking values in $[0, 1]$ whose expected value is the true payoff. Alternative versions might assume that the observed payoff is within some distance ϵ from the true payoff. Noisy query models might be more realistic, and they are suggested by by the experimental papers on querying games. However in a theoretical context, one could obtain good approximations of the expected payoffs for a profile s, by repeated sampling. It would interesting to understand the power of different query models.

REFERENCES

ALON, N., EMEK, Y., FELDMAN, M., AND TENNENHOLTZ, M. 2011. Economical graph discovery. In *Proc. of ICS*. 476–486.

ANGLUIN, D. 1987. Learning regular sets from queries and counterexamples. *Information and Computation 75*, 2, 87–106.

BERENBRINK, P., FRIEDETZKY, T., GOLDBERG, L., GOLDBERG, P., AND MARTIN, R. 2007. Distributed selfish load balancing. *SIAM Journal on Computing 37*, 1163–1181.

BRIEST, P., GOLDBERG, P. W., AND RÖGLIN, H. 2008. Approximate equilibria in games with few players. *CoRR abs/0804.4524*.

CHIEN, S. AND SINCLAIR, A. 2011. Convergence to approximate Nash equilibria in congestion games. *Games and Economic Behavior 71*, 2, 315–327.

DASKALAKIS, C., FRONGILLO, R., PAPADIMITRIOU, C., PIERRAKOS, G., AND VALIANT, G. 2010. On learning algorithms for Nash equilibria. In *Proc. of SAGT*. 114–125.

DASKALAKIS, C., GOLDBERG, P., AND PAPADIMITRIOU, C. 2009a. The complexity of computing a Nash equilibrium. *SIAM Journal on Computing 39,* 1, 195–259.

DASKALAKIS, C., MEHTA, A., AND PAPADIMITRIOU, C. H. 2009b. A note on approximate Nash equilibria. *Theoretical Computer Science 410,* 17, 1581–1588.

DASKALAKIS, C. AND PAPADIMITRIOU, C. 2009. On oblivious PTAS's for Nash equilibrium. In *Proc. of 41st STOC.* 75–84.

DUONG, Q., VOROBEYCHIK, Y., SINGH, S., AND WELLMAN, M. 2009. Learning graphical game models. In *Proceedings of the 21st IJCAI.* 116–121.

EVEN-DAR, E., KESSELMANN, A., AND MANSOUR, Y. 2003. Convergence time to Nash equilibria. In *Proc. of ICALP.* 502–513.

FABRIKANT, A., PAPADIMITRIOU, C. H., AND TALWAR, K. 2004. The complexity of pure Nash equilibria. In *Proc. of STOC.* 604–612.

FEDER, T., NAZERZADEH, H., AND SABERI, A. 2007. Approximating Nash equilibria using small-support strategies. In *Proc. of 8th ACM EC.* 352–354.

FELDMANN, R., GAIRING, M., LÜCKING, T., MONIEN, B., AND RODE, M. 2003. Nashification and the coordination ratio for a selfish routing game. In *Proc. of ICALP.* 514–526.

FISCHER, S., RÄCKE, H., AND VÖCKING, B. 2006. Fast convergence to Wardrop equilibria by adaptive sampling methods. In *Proc. of the 38th STOC.* pp. 653–662.

FUDENBERG, D. AND LEVINE, D. 1998. *The Theory of Learning in Games.* MIT Press.

GAIRING, M., LÜCKING, T., MAVRONICOLAS, M., MONIEN, B., AND RODE, M. 2008. Nash equilibria in discrete routing games with convex latency functions. *Journal of Computer and System Sciences 74,* 1199–1225.

GOLDBERG, P. 2004. Bounds for the convergence rate of randomized local search in a multiplayer, load-balancing game. In *Proc. of PODC.* 131–140.

GOLDBERG, P. AND PASTINK, A. 2012. On the communication complexity of approximate Nash equilibria. In *Proc. of 5th SAGT.* LNCS 7615. 192–203.

GOLDBERG, P. W., SAVANI, R., SØRENSEN, T. B., AND VENTRE, C. 2013. On the approximation performance of fictitious play in finite games. *International Journal of Game Theory*, 1–25.

HART, S. AND MANSOUR, Y. 2010. How long to equilibrium? The communication complexity of uncoupled equilibrium procedures. *Games and Economic Behavior 69*, 107–126.

HART, S. AND MAS-COLELL, A. 2003. Uncoupled dynamics do not lead to Nash equilibrium. *American Economic Review 93,* 5, 1830–1836.

HART, S. AND MAS-COLELL, A. 2006. Stochastic uncoupled dynamics and Nash equilibrium. *Games and Economic Behavior 57,* 2, 286–303.

HÉMON, S., DE ROUGEMONT, M., AND SANTHA, M. 2008. Approximate Nash equilibria for multi-player games. In *SAGT.* 267–278.

JORDAN, P., SCHVARTZMAN, L., AND WELLMAN, M. 2010. Strategy exploration in empirical games. In *Proc. of 9th AAMAS.* 1131–1138.

JORDAN, P., VOROBEYCHIK, Y., AND WELLMAN, M. 2008. Searching for approximate equilibria in empirical games. In *Proc. of 7th AAMAS, Vol. 2.* 1063–1070.

KEARNS, M., LITTMAN, M., AND SINGH, S. 2001. Graphical models for game theory. In *Proc. of the 17th UAI.* 253–260.

NUDELMAN, E., WORTMAN, J., SHOHAM, Y., AND LEYTON-BROWN, K. 2004. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In *Proc. of 3rd AAMAS.* 880–887.

SUREKA, A. AND WURMAN, P. 2005. Using tabu best-response search to find pure strategy Nash equilibria in normal form games. In *Proc: of AAMAS.* 1023–1029.

VOROBEYCHIK, Y., WELLMAN, M., AND SINGH, S. 2007. Learning payoff functions in infinite games. *Machine Learning 67*, 145–168.

WELLMAN, M. 2006. Methods for empirical game-theoretic analysis. In *Proc. of AAAI.* 1552–1555.