The Query Complexity of Correlated Equilibria

Sergiu Hart * Noam Nisan †

May 17, 2013

Abstract

We consider the complexity of finding a Correlated Equilibrium in an n-player game in a model that allows the algorithm to make queries for players' utilities at pure strategy profiles. Many randomized regret-matching dynamics are known to yield an approximate correlated equilibrium quickly: in time that is polynomial in the number of players, n, the number of strategies of each player, m, and the approximation error, ϵ^{-1}. Here we show that both randomization and approximation are necessary: no efficient deterministic algorithm can reach even an approximate equilibrium and no efficient randomized algorithm can reach an exact equilibrium.

1 Introduction

The computational complexity of various notions of equilibria in games is of interest in many different models of computation. Perhaps the most striking positive result in this vein is the surprising power of regret-based algorithms in finding correlated equilibria. This is family of natural dynamics that is known to provably converge quickly to a correlated equilibrium for any game (e.g. Littlestone and Warmuth [1994], Foster and Vohra [1998], Hart and Mas-Colell [2000], Blum and Mansour [2007]). This is in stark contrast to the fact that there is no natural dynamic that is known to converge to a Nash equilibrium, and in fact no efficient algorithm is known either.

Looking at these dynamics from a strictly algorithmic point of view, these algorithms take utility functions $(u_1...u_n)$ of n players as input, and produce an (approximate) correlated equilibrium as output. Assuming that each player has m pure strategies the input size is $n \cdot m^n$, and yet the algorithm runs in time that is polynomial in n and m, a time

*Institute of Mathematics, Department of Economics, and Center for the Study of Rationality, Hebrew University of Jerusalem. Research partially supported by a European Research Council Advanced Investigator grant.

†Microsoft Research, Silicon Valley and Hebrew University of Jerusalem.

that is sub-linear (even poly-logarithmic) in the input size. Even though, in principle, the output may be of size similar to the input size, these algorithms produce an output whose support is also polynomial in n and m. Regret-based algorithms only need "black-box" access to the utility functions, making a sequence of queries $u_i(s_1...s_n)$ for pure strategy profiles $(s_1...s_n) \in S_1 \times \cdots \times S_n$.

As algorithms, these regret-based ones have two undesirable aspects: first, they are randomized, and second, they only produce an approximate equilibrium. Their running time is polynomial in the desired accuracy parameter and so to obtain an exact correlated equilibrium they require time that is exponential in n. This paper asks whether these deficiencies can be fixed. While for many problems with sub-linear algorithms it is clear that both randomization and approximation are required, this is not the case here. Usually, the necessity of randomized approximation is already implied by the verification of the result itself, also known as the certificate complexity or non-deterministic complexity[1]. However, correlated equilibria can be verified in time that is polynomial in their size which itself can be polynomial in n and m, so the certificate (non-deterministic) complexity of correlated equilibrium is small.

Another indication that exact or deterministic algorithms may be possible comes from the known LP-based algorithms for correlated equilibria (Papadimitriou and Roughgarden [2008], Jiang and Leyton-Brown [2011]) that produce, in time polynomial in m and n, a correlated equilibrium *exactly and deterministically* in the somewhat stronger model where the algorithm may query the utility black boxes also at profiles of *mixed* strategies (queries that also fit into the stronger communication model studied e.g. in Hart and Mansour [2010].)

In a very recent work, Babichenko and Barman [2013] showed that every *deterministic* algorithm in the basic model that finds an *exact* correlated equilibrium requires time that exponential in n. This leaves open the question of whether the success of the regret-based algorithms is due to the power of randomization (which is known to be critical for achieving low regret) or due to the relaxation allowing approximate equilibria rather than exact one.

In this paper we show that it is actually both of these, even when we limit ourselves to bi-strategy games (i.e. $S_i = \{0, 1\}$ for each player i.)[2] The following lower bounds

[1]Let us look, as an example, at the prototypical sub-linear algorithm of statistical sampling. For input $x \in \{0, 1\}^n$, the task is to compute – perhaps approximately – the fraction of 1's in the input: $\sum_i x_i/n$. A randomized approximation algorithm samples $O(\epsilon^{-2})$ entries and gets, with high probability, an ϵ-approximation. In this case it is easy to see that that both randomization and approximation are crucial since even if the answer was given to the algorithm – say, exactly half of the input bits are 1 – then verifying that this is so requires querying essentially all inputs, if you want to do it deterministically (even approximately) or do it exactly (even randomly).

[2]Notice that for bi-strategy games coarse equilibria are equivalent to correlated equilibria so the lower bounds obtained apply also to the easier problem of finding coarse equilibria.

apply in the strong "query model" ("decision tree model") which makes no assumptions on the algorithm (in particular not restricting its computational power) beyond the fact that it has "black-box" access to the utility functions that can provide it with the values $u_i(s_1...s_n)$ for an adaptively chosen sequence of pure strategy profiles $(s_1...s_n)$.[3]

Theorem A: Every deterministic algorithm that finds a 1/2-approximate correlated equilibrium in all bi-strategy games with 0/1 utilities requires $2^{\Omega(n)}$ queries in the worst case.

Theorem B: Every randomized (or deterministic) algorithm that finds an exact correlated equilibrium in all bi-strategy games with utilities specified as n-bits numbers in the range $[0, 1]$ requires $2^{\Omega(n)}$ expected cost in the worst case. Here the cost includes the number of queries made and the size of the support of the output produced[4].

Of course, these lower bounds apply also to the harder, perhaps more interesting, problem of finding a Nash equilibrium (since every Nash equilibrium is also a correlated equilibrium), but it is not difficult to see that, in contrast to the correlated case, for Nash equilibria these bounds are "trivial" as they apply also to the verification complexity.[5] However, if we allow both randomization and approximation then the verification complexity of Nash equilibrium becomes polynomial in n and m (since we can verify that each player i is best-responding – approximately – by sampling from the distributions of the other players). Thus the following remains as an open problem:

Open Problem: Does there exist a randomized algorithm, with only black box access to the player's utility functions, that finds an ϵ-Nash equilibrium for every n-player game where each player has at most m strategies, whose running time is polynomial in n, m, and ϵ^{-1}?

This seems to be the weakest model for which an efficient algorithm for Nash equilibrium is not known to be impossible.[6] Since this problem asks for sub-linear time algorithms, proving a lower bound does not seem out of reach (perhaps even using tech-

[3]Since we are proving lower bounds, the strong (un-realistic) model only strengthens our results. We should note that all the algorithms mentioned above are actually effective and not only do they make few queries but also the rest of the computation only requires polynomial-time.

[4]For zero-error algorithms, the lower bound applies also just to the number of queries made.

[5]Consider $n/2$ pairs of players, where each pair is playing matching pennies within itself. Clearly the only Nash equilibrium is where each player evenly randomizes between his two strategies. However verifying this equilibrium – even allowing randomized verification – requires looking at essentially all 2^n strategy profiles since if an adversary changed the utility of a player in any single profile, this would no longer be an equilibrium. For deterministic verification the adversary could change the utility at all non-queried profiles so that this is no longer even an approximate equilibrium.

[6]For example, this model is strictly weaker than the communication complexity model studied in Hart and Mansour [2010] and proving lower bounds in it would seem like a natural step towards proving lower bounds on the communication complexity of mixed Nash equilibria.

niques similar to those used in this paper), and would seem like the most natural first step towards understanding the computational complexity of Nash equilibrium. Recent related work appears in Fearnley et al. [2013].

2 Model and Preliminaries

2.1 Correlated Equilibrium

- **(Game)** We will consider games between n players, where each player's strategy set is $\{0,1\}$. Our players' utilities will be normalized between 0 and 1, so a game will be given by n utility functions $u_1...u_n$ where for each i we have $u_i : \{0,1\}^n \to [0,1]$.

- **(Notation)** For a strategy profile $v \in \{0,1\}^n$, we will use two types of notation: $v^{(i)}$ will denote the result of flipping the i'th bit of v, (i.e. where player i plays $1-v_i$ and all other j play v_j.) $v^{i\to b}$ will denote v with the i'th bit set to b (i.e. if $v_i = b$ then $v^{i\to b} = v$, and otherwise $v^{i\to b} = v^{(i)}$.)

- **(Regret)** Take a probability distribution x over the set of strategy profiles. I.e $x : \{0,1\}^n \to [0,1]$ with $\sum_{v\in\{0,1\}^n} x(v) = 1$. Take a player $1 \le i \le n$ and a possible strategy $b \in \{0,1\}$ for i. We say that the *regret of i from not playing b* is:

$$Regret_{i\to b}(x) = \sum_{v\in\{0,1\}^n} x(v)u_i(v^{i\to b}) - \sum_{v\in\{0,1\}^n} x(v)u_i(v).$$

- **(Correlated Equilibrium)** For $\epsilon \ge 0$, we will say that x is an ϵ-correlated equilibrium (ϵ-CE) if for all $i = 1...n$ and for all $b \in \{0,1\}$ we have that $Regret_{i\to b}(x) \le \epsilon$. For $\epsilon = 0$ we just say that its just a correlated equilibrium (CE).

2.2 Computational problem

The following is the (approximate) *CE problem:*

- **Input:** The utility functions $u_i : \{0,1\}^n \to [0,1]$ and a desired approximation parameter $\epsilon \ge 0$.

- **Queries:** We assume only black-box access to the utilities: a query is of the form $v \in \{0,1\}^n$ the reply to the query is the n-tuple of player utilities: $u_1(v)...u_n(v)$.

- **Output:** An ϵ-correlated equilibrium of the game. The equilibrium is given by listing the probability $x(v)$ for every strategy profile v in its support.

- **Cost:** The cost on a given input $u_1...u_n$ is the total number of queries made plus the size of the support of the equilibrium produced. The Cost of the algorithm is the worst case cost over all n-tuples of utility functions.

Comment: Having $\epsilon = 0$ when all utilities are given by t-bit numbers is equivalent to Setting $\epsilon = exp(-poly(n,t))$ (For one side, clearly one can ignore more than $O(\log n + \log \epsilon^{-1})$ bits of precision without hurting the output by more than ϵ, and for the second side, due to being a solution of a LP, it is known that no more than $poly(n,t)$ bits of precision are needed in an exact solution.)

We say that the CE problem can be *exactly solved* in polynomial time if there is an algorithm for it that runs in time $poly(n, \log \epsilon^{-1})$. We say that the CE problem can be *approximately solved* in polynomial time if there is an algorithm for it that runs in time $poly(n, \epsilon^{-1})$.

2.3 Concise Equilibrium Representations

Notice that in the definition of the CE problem we counted the size of the support of the equilibrium produced towards the cost of the algorithm. We could alternatively talk about the *weak-CE problem* where the algorithm is allowed to produce an equilibrium with arbitrarily large support whose size is not counted as part of the cost. Practically, the algorithm could use some concise representation of the equilibrium, e.g., as some mixture of product distributions. We show however that this would not make the problem significantly easier, as any approximate CE algorithm that makes a small number of queries can be converted to one that also produces a CE with small support.

Lemma 2.1 *Given an algorithm that solves the weak-CE problem with error ϵ in T queries, there exists an algorithm that solves the (strong) CE problem with error $O(\sqrt{\epsilon})$ and cost (including support size) $O(T)$.*

We first bound the amount of weight that a CE algorithm may put on un-queried profiles.

Lemma 2.2 *Consider any ϵ-CE algorithm that on some input $(u_1..u_n)$ queries a set Q of profiles and outputs an ϵ-CE x. Denote $Q' = \{v | v \in Q \text{ or } v^{(1)} \in Q\}$ and let $\alpha = max_{v \in Q'} u_1(v)$, then x puts weight of at most $2(\alpha + \epsilon)$ outside Q', i.e. $\sum_{v \notin Q'} x(v) \leq 2(\alpha + \epsilon)$.*

Proof: Assume by way of contradiction otherwise, and furthermore assume without loss of generality that at least half of this weight is on v's with $v_1 = 0$, i.e. $\sum_{v \notin Q' \text{ and } v_1 = 0} x(v) >$

$\alpha + \epsilon$. Now consider changing the input utilities in two different ways without changing the queried profiles. The first way just assigns $u_1(v) = 0$ for all $v \notin Q'$. In the second way we put $u_1(v) = 0$ for all $v \notin Q'$ with $v_1 = 0$ and $u_1(v) = 1$ for $v \notin Q'$ with $v_1 = 1$. Clearly the gap between $Regret_{1\to 1}$ for these two cases is exactly $\sum_{v \in Q' \, and \, v_1 = 0} x(v) > \alpha + \epsilon$. However, notice that since $\alpha = max_{v \in Q'} u_1(v)$, in the first way, $u_1(v)$ is bounded by α for all v and thus $|Regret_{1\to 1}| \leq \alpha$, it follows that in the second case we have $Regret_{1\to 1} > \epsilon$. This contradicts the correctness of the of the ϵ-CE algorithm on the input obtained by the second way of changing the original input. ∎

We now complete the proof of the computational equivalence between the two versions of the approximate CE problem.

Proof: (of lemma 2.1) Let us first run the weak-CE algorithm on α-scaled utilities $u'_i(v) = \alpha u_i(v)$ to obtain a ϵ-CE $x'(v)$ for the u''s (for α to be determined below). By scaling back we have that x' is an (ϵ/α)-CE for the original u's: $\sum_{v \in \{0,1\}^n} x'(v) u_i(v^{i \to b}) - \sum_{v \in \{0,1\}^n} x'(v) u_i(v) \leq \epsilon/\alpha$. As in the previous lemma, denote $Q' = \{v | v \in Q \, or \, v^{(1)} \in Q\}$ where Q is the set of queries made by the algorithm. Our output will be $x(v) = 0$ for $v \notin Q'$ and will be $x(v) = \beta x'(v)$ only for the $v \in Q'$, where $\beta = 1/\sum_{v \in Q'} x'(v)$ is a scaling factor ensuring $\sum_v x(v) = 1$ too. By the previous lemma $\sum_{v \notin Q'} x'(v) \leq 2(\alpha + \epsilon)$ so the regrets in x can be larger than the regrets in x' by at most twice that amount (once for omitting $v \notin Q'$ and once for scaling by β): $\sum_{v \in \{0,1\}^n} x(v) u_i(v^{i \to b}) - \sum_{v \in \{0,1\}^n} x(v) u_i(v) \leq \epsilon/\alpha + 4(\alpha + \epsilon)$. Choosing $\alpha = O(\sqrt{\epsilon})$ completes the proof. ∎

3 Deterministic Lower Bound for Approximation

This section proves that deterministic algorithms require exponentially many queries to compute even an approximate equilibrium. Notice that due to lemma 2.1 the same is also implied for the "weak-CE" problem that allows arbitrary concise representations of the output.

Theorem 3.1 *Every deterministic algorithm that finds a* $1/2$*-CE in every win-lose game (i.e.* $u_i \in \{0, 1\}$*) requires* $2^{\Omega(n)}$ *queries in the worst case.*

3.1 The Approximate Sink Problem

Our lower bound here will be based on analyzing the following combinatorial problem on the boolean hypercube, termed the *Approximate Sink (AS)* Problem.

Input: A labeling of the edges of the hypercube by directions. I.e. a for every edge $(v, v^{(i)})$ we have a weight $R(v, v^{(i)}) \in \{-1, 1\}$ such that $R(v, v^{(i)}) = -R(v^{(i)}, v)$.

Queries: The algorithm queries a vertex $v \in \{0,1\}^n$ and gets the directions $R(v, v^{(i)})$ of all edges adjacent to v.

Output: The algorithms wins when it queries a vertex v with in-degree more than $n/4$: $\sum_i R(v^{(i)}, v) \leq n/2$.

We will show that exponentially many queries are needed in order to find such a vertex. This will imply the theorem due to the following simple reduction:

Lemma 3.2 *If there exists a deterministic algorithm that finds a* $1/2$*-CE in every win-lose game with cost* T *then the AS problem can be solved with* T *queries.*

Proof: Given an AS instance we will build a CE instance with $R(v, v^{(i)}) = u_i(v) - u_i(v^{(i)})$ and run the approximate CE algorithm on it, translating every query the CE algorithm makes into an AS query. For $R(v, v^{(i)}) = 1$ we set $u_i(v) = 1$ and $u_i(v^{(i)}) = 0$, while for $R(v, v^{(i)}) = -1$ we set $u_i(v) = 0$ and $u_i(v^{(i)}) = 1$. Under this mapping, when the CE algorithm makes a query $(u_1(v)...u_n(v))$? it is immediately translated to the same query v on the original AS instance, and $R(v, v^{(i)}) = 1$ means $u_i(v) = 1$ while $R(v, v^{(i)}) = -1$ means $u_i(v) = 0$. When the CE algorithm produces an approximate equilibrium x, we will continue by querying all the profiles in the support of x, whose number was, by definition, already counted towards the cost of the CE algorithm.

Now take an $1/2$-equilibrium output by the CE algorithm. Summing up the inequalities of the CE we get: $\sum_{i,b} Regret_{i \to b} = \sum_v x(v) \sum_i (u_i(v^{(i)}) - u_i(v)) \leq n/2$, which implies that for some v in the support of x we have $\sum_i R(v^{(i)}, v) = \sum_i (u_i(v^{(i)}) - u_i(v)) \leq n/2$, as needed. ∎

3.2 Polite Algorithms

So we now prove the lower bound for the AS problem. The heart of our lower bound will be to show that every algorithm can be transformed to the following form:

Definition 3.3 *We will call an algorithm for the AS problem* polite *if whenever a vertex* v *is queried, at least* $3n/4$ *of its neighbors in the hypercube have not yet been queried.*

It is quite easy to show that polite algorithms cannot solve AS.

Lemma 3.4 *No deterministic polite algorithm can solve the AS problem.*

Proof: We will provide an adversary argument: whenever a vertex is queried, the adversary answers with all edges that were previously not committed to pointing out. Since there are at least $3n/4$ such edges, the in-degree is at most $n/4$, so this vertex cannot be an answer. ∎

3.3 Closure

To convert an algorithm to a polite form, we will need to make sure that vertices are queried before too many of their neighbors are. We will use the following notion:

Definition 3.5 *For a set $V \subset \{0,1\}^n$ of vertices in the hypercube, we will define its* closure*, V^* to be the smallest set containing V such that for every $v \notin V^*$ at most $n/8$ of its neighbors are in V^*.*

This set is well defined and can be obtained by starting with $V^* = V$ and repeatedly adding to V^* any vertex v that has more than $n/8$ of its neighbors already in V^*. Clearly the order of additions does not matter since the number of neighbors a vertex has in V^* only increases as other vertices are added to V^*. Clearly when the process stops then every $v \notin V^*$ has at most $n/8$ of its neighbors in V^*.

The point is that we will not need to continue this process of adding vertices for a long time.

Lemma 3.6 *For $|V| < 2^{n/8-1}$ we have that $|V^*| \leq 2|V|$.*

Proof: Assume by way of contradiction that $|V^*| > 2|V|$ and denote by U the set obtained during the process of building V^* after adding exactly $|V|$ vertices, thus $|U| = 2|V|$. let us denote by $e(U)$ the number of directed edges within U: $e(U) = |\{(u,i)|u \in U \text{ and } u^{(i)} \in U\}|$. We will provide conflicting lower and upper bounds for $e(U)$. For the lower bound, notice that every vertex that we added during the process adds at least $n/4$ edges to $e(U)$ ($n/8$ of its own edges as well as the $n/8$ opposite ones), so $e(U) \geq |U - V|n/4 = |V|n/4$. For the upper bound we will use the edge iso-perimetric inequality on the hypercube (Hart [1976]), that implies that for every subset of the hypercube $e(U) \leq |U| \log_2 |U|$. So we have $|V|n/4 \leq |U| \log_2 |U| = 2|V|(log_2|V| + 1)$, so $(1 + log_2|V|) \geq n/8$ contradicting the bound on the size of V. ∎

3.4 A Polite Simulation

So we can now provide our general simulation by polite algorithms, which completes the proof of the theorem.

Lemma 3.7 *Every algorithm that makes at most $T = 2^{n/8-1}$ queries can be simulated by a polite algorithm that makes at most $2T$ queries.*

Proof: For $t = 1...T$ denote the t'th query made by the original algorithm q_t, and denote $Q_t = \{q_1...q_t\}$ to be the set of all queries made until time t. Our polite algorithm will

simulate query q_t by querying all vertices in Q_t^*, i.e. completing the closure implied by adding q_t. Notice that $Q_t^* = (Q_{t-1} \cup \{q_t\})^* = (Q_{t-1}^* \cup \{q_t\})^*$. The difficulty is that we need to add the vertices in $Q_t^* - Q_{t-1}^*$ in a way that will maintain politeness, i.e. where each vertex is added before $n/4$ of its neighbours are.

To see that this is possible let us look at the vertices in $N_t = Q_t^* - Q_{t-1}^*$. First, the previous lemma implies that $|N_t| \leq |Q_t| = t$. By the edge-iso-perimetric inequality applied to N_t we have that $e(N_t) \leq t \log_2 t$, so some vertex $v \in N_t$ has at most $\log_2 t < n/8$ neighbors in N_t – this will be the last vertex our polite algorithm will query in this stage. Similarly, from the remaining elements $N' = N_t - \{v\}$ there is also a vertex v' with at most $\log_2(t-1) < n/8$ neighbors in N', and this vertex will be asked just before v. So we continue until we exhaust N_t. Now we claim that this order maintains politeness: since, by definition, every vertex in N_t has less than $n/8$ neighbors in Q_{t-1}^*, when we add the less than $n/8$ neighbors from N_t that appeared *before* it in the ordering of N_t, we still get less than $n/4$ neighbors preceding it. Finally, notice that the simulating algorithm queries exactly the at most $2T$ vertices in Q_T^* so its running time is as required. ∎

4 Randomized Lower Bound for Exact Computation

This section proves the lower bound for randomized algorithms. Recall that randomized algorithms can in fact compute an *approximate* CE with polynomially many queries (using regret matching). This section proves that they cannot compute an *exact* CE.

First let us formally define a randomized algorithm. A *randomized algorithm* is just a probability distribution over deterministic algorithms. For every input, this random choice of the algorithm results in the output being a random variable. We will say that a randomized algorithm solves a search problem (like our problem of finding an equilibrium) if for every input, the probability that the output is a correct solution is at least 1/2. The *cost* of a randomized algorithm on a given input is the expected cost made over the random choice of the algorithm, and the cost of a randomized algorithm is its cost for the worst case input. So our theorem for this section is.

Theorem 4.1 *Every randomized (or deterministic) algorithm that solves the CE problem exactly must have an expected $2^{\Omega(n)}$ cost (for the worst case input).*

This theorem applies even to randomized algorithms that produce a CE with any non-negligible probability. It also applies to the weak-CE version of the problem defined in section 2.3 but only for zero-error algorithms.[7]

[7]This is because lemma 2.1 holds also for zero-error randomized algorithms as it can be applied to

4.1 The Non-Negative Vertex Problem

Similarly to the deterministic case, here we will reduce the correlated equilibrium problem to the following combinatorial problem on the hypercube. It is essentially a weighted version of the Approximate Sink problem, with a stricter bound on the output quality.

The Non-Negative Vertex (*NNV*) problem:

Input: A labeling of the directed edges of the hypercube by integers where the convention is that $R(u,v) = -R(v,u)$.

Queries: A query is a vertex v in the hypercube. The answer to this query is the tuple of labels on all adjacent edges: $R(v, v^{(i)})$ for $i = 1...n$.

Output: The algorithm must output a vertex v in the hypercube with total non negative weight: $\sum_i R(v, v^{(i)}) \geq 0$.

Similarly to lemma 3.2, proving a randomized lower bound for the NNV problem will imply a similar one for the CE problem due to the following reduction.

Lemma 4.2 *The number of queries required, for a randomized NNV algorithm, to solve the NNV problem is at most the time needed for a randomized algorithm in order to solve the CE problem.*

Proof: We will convert a CE algorithm to an NNV one. Let $m = max_{u,v} R(u,v)$. Given an NNV instance we will build a CE instance ensuring that $u_i(v) - u_i(v^{(i)}) = R(v, v^{(i)})/m$: For positive $R(v, v^{(i)})$ we set $u_i(v) = R(v, v^{(i)})/m$ and $u_i(v^{(i)}) = 0$, while for negative $R(v, v^{(i)})$ we set $u_i(v) = 0$ and $u_i(v^{(i)}) = -R(v, v^{(i)})/m$. Under this mapping, when the CE algorithm makes a query v it is immediately translated to the same query v of the NNV black box, and the answer from the NNV black box directly provides the answer to the CE query.

Now take an equilibrium x output by the CE algorithm. Summing up the inequalities of the CE we get: $\sum_{i,b} Regret_{i \to b} = \sum_v x(v) \sum_i (u_i(v^{(i)}) - u_i(v)) \leq 0$, which implies that for some v in the support of x we have $\sum_i R(v^{(i)}, v) = \sum_i (u_i(v^{(i)}) - u_i(v)) \leq 0$, as needed to provide an answer to the NNV problem. ∎

We will continue proving the lower bound for randomized algorithms solving the NNV problem by exhibiting a distribution over NNV instances such that every *deterministic algorithm* requires exponentially many queries in order to succeed on a non-negligible fraction of inputs drawn according to this distribution. The lower bound for randomized CE algorithms follows thus completing the proof of theorem 4.1.

each deterministic algorithm in the support. (Our proof does not imply the extension to the weak-CE case for general randomized algorithms, even though we believe that the theorem itself does extend.)

4.2 A Path Construction

We will build hard instances of the NNV problem from paths in the hypercube. Let $(v_0, v_1, ..., v_L)$ be a (not necessarily simple) path in the hypercube, i.e. for each $0 \leq j < L$, the vertex v_{j+1} is obtained from v_j by flipping a single random bit. The NNV instance we build from this path will essentially give weight j to the edge (v_{j-1}, v_j), with weights added over the possible multiple times the path goes through a single edge. This way every time the path passes a vertex v at step j, the incoming edge gets weight $-j$ while the outgoing edge gets weight $j+1$, adding one to the total weight of edges going out of v. Formally:

Definition 4.3 *Let $(v_0, v_1 ... v_L)$ be a (not necessarily simple) path in the hypercube. The path induces the following labeling of the hypercube: $R(u,v) = \sum_{\{j|u=v_{j-1}, v=v_j\}} j - \sum_{\{j|u=v_j, v=v_{j-1}\}} j$.*

Lemma 4.4 *For each $v \neq v_L$ we have that $\sum_i R(v, v^{(i)}) = -|\{j|v_j = v\}|$. For v_L we have that $\sum_i R(v_k, v_k^{(i)}) = L - |\{j|v_j = v\}|$.*

Proof: Except for the last vertex, v_L, whenever the path goes goes into v at step j and exist it in step $j+1$, the total values of R's going out of this vertex decreases by 1 ($+j$ incoming and $-(j+1)$ outgoing). ∎

This means that if the path covers the whole hypercube then the only non-negative vertex in it will be the end of the path.

We now define a random distributions over paths that cover the whole hypercube and whose final end point is random. We start with some (fixed) Hamiltonian path in the hypercube. From that point on we continue with a random walk of length $L = n \cdot 2^{n/3}$. Why would it be hard for an algorithm to find the end of the path? We will show that an algorithm must essentially follow the path query by query. Otherwise it is looking for a length $L = n \cdot 2^{n/3}$ randomly-hidden needle in a 2^n size haystack. But probing places that are already known to be in the random part of the path will only allow the algorithm to advance sequentially over it, requiring exponential time to reach the end.

Lemma 4.5 *Any deterministic algorithm that runs in time $T < 2^{n/3}/n$ will be able to solve the NNV problem on at most fraction $O(2^{-n/3})$ of inputs drawn according to this distribution.*

Proof: Since all the information in our path-based instances of the NNV problem are determined by the path, we can imagine that the queries directly ask for this path information. Since the only non-negative vertex in these instances is the end of the path, our algorithm will really need to find it. Not only will this be hard to do, but it is even

hard to find any vertex that is near the tail of the path. This will be hard since, on one hand, the tail is a tiny fraction of the hypercube so can't be found "at random", and, on the other hand, the only possible "deliberate" way to find it is to follow the path step by step which takes exponential time. The reason that no "shortcuts" are possible when following the path is that the random walk in the hypercube mixes rapidly so you get no information about how the path continues beyond the very near vicinity. To formalize this line of reasoning we will introduce a variant of the problem that explicitly yields to the algorithm any information we think that it may get any handle on. Specifically, we tell the algorithm everything about the path except for its tail, and, furthermore, give it an additional n^2 vertices at the beginning of this tail every step. Once this is given to the algorithm, we will be able to show that the algorithm can never learn anything new.

The Hit-The-Path (HTP) problem:

INPUT: A random path $(v_0...v_L)$ in the hypercube of length $L = n2^{n/3}$, starting from a *revealed* vertex v_0.

QUERIES: At each step $t = 1...T$:

1. The algorithm may query a vertex q_t of the hypercube, depending the reveled information so far (which we will soon see is exactly the sequence $(v_0...v_{(t-1)n^2})$.)

2. The next n^2 vertices of the path, $v_{(t-1)n^2+1}...v_{tn^2}$, are revealed (independently of the query).

3. The algorithm wins immediately if q_t is on the *non-previously revealed* part of the path, $q_t \in \{v_{tn^2+1}...v_L\}$.

We will show that a T-query algorithm, for $T < 2^{n/3}/n$ can win with probability of at most $O(2^{-n/3})$. Let us formally see why this will imply the lemma. Suppose that we have an algorithm that succeeds in solving the NNV problem on a larger fraction of inputs drawn according to this distribution, we will use it to win instances of this HTP problem with at least the same probability. Whenever the NNV problem makes a query to v we make the same query in the HTP case. If we win, then we are done. Otherwise, we know that the non-revealed part of the path does not pass through v so the reply to the NNV query is completely determined by the revealed part of the path which we already have and can use to reply. If the NNV algorithm succeeds then it must have found the last vertex on the path (the only non-negative one), which is on the path and is only revealed after $L/n^2 = 2^{n/3}/n > T$ queries, so is still unrevealed and thus our HTP algorithm wins too.

We now continue with the lower bound for HTP, so fix a (deterministic) algorithm for the HTP problem that makes at most T queries. If it wins, then for some step $1 \leq t \leq T$

it won (for the first time) by finding an unrevealed vertex v_j for $tn^2 < j \leq L$. We will bound this probability (over the random choice of the path) for a fixed t and j, and then use the union bound to obtain an upper bound on the probability that the algorithm wins. Now let us look at the t'th query q_t made by the algorithm. If non of the previous queries won, then the only information the algorithm had when making this query is the revealed part of the path, i.e. $(v_1...v_{(t-1)n^2})$, and so the query is just a function of them, $q_t = q_t(v_1...v_{(t-1)n^2})$. So what is the probability that for some function on these inputs we have that $q_t(v_1...v_{(t-1)n^2}) = v_j$? Note that our construction of random paths means that v_j is obtained by taking a random walk of length $j - (t-1)n^2 \geq n^2$ from vertex $v_{(t-1)n^2}$. Now comes the crucial observation: as the mixing time of the hypercube is known to be $O(n \log n) < n^2$, (Diaconis et al. [1990]), this means that a random walk of length $l \geq n^2$ ends at an almost uniformly random vertex of the hypercube. Thus for any fixed $(v_1...v_{(t-1)n^2})$, we have that v_j is almost uniformly distributed over the hypercube and so $Pr[q_t(v_1...v_{(t-1)n^2}) = v_j] = (1+o(1))2^{-n}$. Multiplying this quantity by $T < 2^{n/3}/n$ (for all possible values of t) and then by $2^{n/3}n$ (for all possible values of j), we get the required upper bound for the probability of winning. ∎

This completes the proof of theorem 4.1.

5 Acknowledgements

We would like to thank Parikshit Gopalan for discussions leading to the proof of theorem 3.1.

References

Yakov Babichenko and Siddarth Barman. Query complexity of correlated equilibrium, manuscript, 2013.

Avrim Blum and Yishay Mansour. From external to internal regret. *J. Mach. Learn. Res.*, 8:1307–1324, December 2007. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1314498.1314543`.

Persi Diaconis, Ronald L. Graham, and John A. Morrison. Asymptotic analysis of a random walk on a hypercube with many dimensions. *Random Structures & Algorithms*, 1(1):51–72, 1990.

John Fearnley, Martin Gairing, Paul W. Goldberg, and Rahul Savani. Learning equilibria of games via payoff queries. *CoRR*, abs/1302.3116, 2013.

Dean Foster and Rakesh Vohra. Asymptotic calibration. *Biometrika*, 85:379–390, 1998.

Sergiu Hart. A note on the edges of the n-cube. *Discrete Mathematics*, 14:157–163, 1976.

Sergiu Hart and Yishay Mansour. How long to equilibrium? the communication complexity of uncoupled equilibrium procedures. *Games and Economic Behavior*, 69(1): 107–126, 2010.

Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68:1127–1150, 2000.

Albert Xin Jiang and Kevin Leyton-Brown. Polynomial-time computation of exact correlated equilibrium in compact games. In *Proceedings of the 12th ACM conference on Electronic commerce*, EC '11, pages 119–126, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0261-6. doi: 10.1145/1993574.1993593. URL `http://doi.acm.org/10.1145/1993574.1993593`.

Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, February 1994. ISSN 0890-5401. doi: 10.1006/inco.1994.1009. URL `http://dx.doi.org/10.1006/inco.1994.1009`.

Christos H. Papadimitriou and Tim Roughgarden. Computing correlated equilibria in multi-player games. *J. ACM*, 55(3):14:1–14:29, August 2008. ISSN 0004-5411. doi: 10.1145/1379759.1379762. URL `http://doi.acm.org/10.1145/1379759.1379762`.