Strategizing against Learners in Bayesian Games

Yishay Mansour MANSOUR.YISHAY@GMAIL.COM
Blavatnik school of computer science, Tel Aviv University and Google Research
Mehryar Mohri MOHRI@GOOGLE.COM
Google Research and Courant Institute of Mathematical Sciences
Jon Schneider JSCHNEI@GOOGLE.COM
Google Research
Balasubramanian Sivan BALUSIVAN@GOOGLE.COM
Google Research

Editors: Po-Ling Loh and Maxim Raginsky

Abstract

We study repeated two-player games where one of the players, the learner, employs a no-regret learning strategy, while the other, the optimizer, is a rational utility maximizer. We consider general Bayesian games, where the payoffs of both the optimizer and the learner could depend on the type, which is drawn from a publicly known distribution, but revealed privately to the learner. We address the following questions: (a) what is the bare minimum that the optimizer can guarantee to obtain regardless of the no-regret learning algorithm employed by the learner? (b) are there learning algorithms that cap the optimizer payoff at this minimum? (c) can these algorithms be implemented efficiently? While building this theory of optimizer-learner interactions, we define a new combinatorial notion of regret called polytope swap regret, that could be of independent interest in other settings.

Keywords: Stackelberg value; swap regret; Bayesian games

1. Introduction

How should one play a two-player repeated game? A commonly employed strategy when dealing with a repeated setting is to use a no-regret learning algorithm. Such algorithms assign higher weight to actions that achieved good performance in previous rounds of the game. An important danger lurks when one uses a learning algorithm to play a repeated game: the opponent (who we will call the "optimizer") could be a rational utility-maximizer who might try to explicitly exploit the fact that the learning algorithm chooses its actions based on past performance. Can one design learning algorithms that do not get fooled because they learn from past actions? Can we precisely characterize the class of learning algorithms that are robust from being manipulated in this way? What are the meaningful outcomes and benchmarks when studying this optimizer-learner interaction?

Recent work by Deng et al. (2019) initiated the study of optimizer-learner interactions in general 2-player bimatrix games. They showed that regardless of, and without knowledge of, the specific no-regret-learning algorithm used by the learner, the optimizer can always guarantee himself at least the Stackelberg value[1] of the game by playing a static fixed strategy each round. More interestingly, they show that for a large class of no-regret learning algorithms called mean-based algorithms,

1. The Stackelberg variant of a two-player game is a one-shot two-stage game where the optimizer moves first and publicly commits to a (possibly mixed) strategy, and the learner then best responds to this strategy. The equilibrium

there are games where the optimizer can badly mislead the learner and profit immensely by playing a *dynamic strategy* that varies over time. In particular, the optimizer can architect situations where the learners' responses in certain rounds are far from their best response, owing to the force of memory that is inherent in these learning algorithms. However, Deng et al. (2019) also show that if the learner were to use a more sophisticated learning algorithm, namely, a no-swap-regret algorithm, then the optimizer is unable to get anything more than the utility he is able to get in the Stackelberg equilibrium of the game. I.e., a no-swap-regret algorithm is *sufficient* to prevent the optimizer from benefiting from dynamic strategic behavior.

Questions. In this paper, we primarily focus on two questions. First, the results of Deng et al. (2019) immediately motivate the following question: in general 2-player games (the same set of games studied in Deng et al. (2019); we call these *standard* games), is a no-swap-regret algorithm also *necessary* for the learner to cap the optimizer's payoff of Stackelberg value? Or can the learner run algorithms that, despite having large swap regret, cap the optimizer's payoff at the Stackelberg value of the game? In other words, we seek to characterize the precise class of learning algorithms that ensure that the optimizer cannot benefit from dynamic strategic behavior.

Second, we seek to develop the theory of optimizer-learner interaction in the significantly more general class of Bayesian games, and develop a complete understanding of the landscape there. These games arise naturally in various economic settings, for example an optimizer selling an item to a learner where the learner's private value for the item is their type (Braverman et al., 2018). Formally, a Bayesian game begins with one of C *contexts* ("types") $c \in [C]$ being drawn from a publicly known distribution $\mathcal{D}$ with probability p_c of outputting context c. This context c is told to the learner but not to the optimizer. Based on the context c, the learner chooses an action $j \in [N]$; simultaneously, the optimizer chooses an action $i \in [N]$. The optimizer then receives utility $u_O(i, j, c)$, and the learner receives utility $u_L(i, j, c)$ – note that we allow both utilities to depend on the context c. I.e., instead of a single bi-matrix in the *standard* game, a Bayesian game can be thought of as specified by C bi-matrices.

As in standard games, it is straightforward to show that by playing a static fixed strategy, an optimizer can obtain at least the *Bayesian Stackelberg* value of the game per round, as long as the learner runs a no-(contextual)-regret learning algorithm (we prove this in Lemma 16). For the Bayesian setting, we would like to understand (a) are there learning algorithms that are robust to dynamic strategic behavior (that cap the optimizer payoff at this Stackelberg value)? in particular, what is the right generalization of swap regret? (b) can these algorithms be implemented efficiently? if not, what guarantees can we provide for efficient learning algorithms?

1.1. Our Results

For the first question on standard games, we show that no-swap-regret algorithms are the precise class of algorithms that are robust against dynamic strategic behavior of the optimizer. Specifically, for any learning algorithm that has a swap-regret of R, we construct games where the optimizer *can earn $R/2$ more than the Stackelberg value of the game*. In particular therefore, if the learner had, say, a linear swap regret, the optimizer would earn linearly more than the Stackelberg value of

that results in this two-stage game when both players play optimally is called a Stackelberg equilibrium. We note here that the optimizer's utility in a Stackelberg equilibrium is at least as high as the utility he can get in any (pure or mixed-strategy) equilibrium thereby showing that playing against a learner is more beneficial than playing against another optimizer (i.e., a rational utility maximizer). Formally defined in Section 2.

the game. The main novelty in the proof of this result lies in the game construction: the payoffs in the game we design should ensure a delicate balance between the optimizer's payoff being high, while the Stackelberg value not being too high. We present the precise details of the construction in Section 3. Apart from completing the picture for standard games, this result also provides a new characterization of swap regret as a measure of robustness against strategic behavior.

For Bayesian games, the challenges are multi-fold. Unlike standard games, it is not clear what the correct generalization of swap regret should be. Many seemingly natural choices of regret definition turn out to be incorrect. For example, motivated by the fact that running an independent low external regret algorithm per context results in low external regret, one may consider running an independent low swap regret algorithm for each context. But this does not work! In particular, there are simple games where an optimizer can earn linearly more than the Bayesian Stackelberg value by playing a dynamic strategy against such a learning algorithm (we give one example in Theorem 11).

In this paper we provide a nuanced generalization of swap-regret to the Bayesian setting that we call *polytope swap regret*. We prove that this notion of regret has the guarantee that any learner playing a learning algorithm with $o(T)$ polytope regret *is guaranteed to asymptotically cap the optimizer payoff at the Bayesian Stackelberg value*. While we do not yet know that a low polytope-swap regret is necessary to cap the optimizer payoff at the Bayesian Stackelberg value (i.e., that it is tight in the same way swap regret is for standard games), we provide another generalization of swap regret called *linear swap regret* such that *a low linear swap regret is necessary to cap the optimizer payoff at the Bayesian Stackelberg value*.

Polytope swap regret actually extends far beyond just the Bayesian setting, and can be thought of as a generalization of swap regret to the setting of online linear optimization. The idea behind polytope swap regret stems from viewing the learner's actions – mappings from contexts to a distribution over actions – as points in the polytope $\mathcal{P} = \Delta([N])^C \subseteq \mathbb{R}^{N \times C}$. Our "swap functions" then allow the vertices of this polytope to be swapped with each other. Every point inside the polytope (including the learner's actions) can be written as a convex combination of the vertices of $\mathcal{P}$, to which this swap function can be applied. Of course, there may be many ways to write a given point as a convex combination of vertices: we consider the most permissive definition of regret by choosing the decomposition that leads to the least regret in hindsight. I.e., we say that polytope swap regret is high only if every vertex decomposition of the learner's actions will generate high regret by applying a swap function to the vertices. Precise definitions are given in Section 4 and 4.3. When we restrict these swap functions to be linear maps, we obtain the linear swap regret. Interestingly, when $\mathcal{P}$ is the simplex $\Delta([N])$ both these notions of swap regret (polytope and linear), are equal (it is possible to implement any swap function on vertices via a linear map), and both reduce to the ordinary notion of swap regret.

These twin concepts, with polytope swap regret being sufficient and linear swap regret being necessary, raise the question of which of these could potentially be both necessary and sufficient. To answer this, we first show that these two concepts are not the same by separating polytope and linear swap regret (Theorem 8). We then show in Theorem 9 that linear swap regret is not a sufficient condition. We conjecture that polytope swap regret is the right notion that captures robustness against strategic behavior in Bayesian games, by being both necessary and sufficient. We leave the necessity of polytope swap regret as a concrete open question.

Efficient Algorithms for Bayesian Games. A natural question is whether we can given efficient algorithms for these generalizations of swap regret. We address this issue in Appendix D. In par-

ticular, we show how to construct a low polytope swap regret algorithm given any low swap regret algorithm as input, such that this algorithm incurs a regret of at most $O(\sqrt{TV \log V})$ and runs in time $O(\text{poly}(V))$, where V is the number of vertices of the polytope under consideration. For Bayesian games, the number of vertices of the polytope $\mathcal{P} = \Delta([N])^C$ is N^C, and thus, in cases where the number of contexts C is small, these bounds are manageable. When C is large, we analyze two other algorithms: the "low-swap-regret per context" algorithm mentioned above, and a generalization of the external to internal regret reduction of Blum and Mansour (2007). While neither algorithm (provably) has low polytope regret, we show they both provide some robustness by capping the optimizer's payoff at some variant of the Bayesian Stackelberg value of the game. Finally, we conjecture this exponential dependence on C is necessary to achieve low polytope regret – we contribute one piece of evidence towards this by showing that the Bayesian Stackelberg value itself is APX-hard to compute in general Bayesian games (Theorem 15).

1.2. Related Work

While the introduction discusses the work closest to ours, namely Deng et al. (2019), we discuss further related work in detail in Appendix A.

2. Model and preliminaries

Notation We write $[N]$ to denote the set $\{1, 2, \ldots, N\}$. For a finite set S, we write $\Delta(S)$ to denote the set of distributions over S. We defer most proofs to Appendix F for the sake of brevity.

2.1. Games and equilibria

We begin this paper by considering finite bimatrix games (which we refer to as *standard games*). A standard game is a game between two players, who we refer to as the *optimizer* and the *learner*. The optimizer must choose one of M actions (labeled 1 through M) and learner must simultaneously choose one of N actions (labeled 1 through N). If the optimizer chooses action $i \in [M]$ and the learner chooses action $j \in [N]$, then the optimizer receives utility $u_O(i, j)$ and the learner receives utility $u_L(i, j)$. We will assume all utilities are bounded in $[-1, 1]$ (so $|u_O(i, j)| \leq 1$, and $|u_L(i, j)| \leq 1$). To simplify analysis in the sections that follow, we will eliminate the role of randomness (which is mostly tangential to the main points of this paper) by allowing the optimizer and learner to directly play mixed strategies in $\Delta([M])$ and $\Delta([N])$, and deterministically receive the corresponding expected reward. That is, when the optimizer plays $\alpha \in \Delta([M])$ and $\beta \in \Delta([N])$, the optimizer's utility is given (deterministically) by $u_O(\alpha, \beta) = \sum_{i=1}^{M} \sum_{i=1}^{N} \alpha_i \beta_j u_O(i, j)$ (and the learner's utility is computed similarly).

In the second part of this paper, we extend our study to a specific subclass of Bayesian games where the learner is randomly assigned a *type*, unknown to the optimizer (we refer to such games simply as *Bayesian games*). Such games arise naturally in various economic settings, for example an optimizer selling an item to a learner where the learner's private value for the item is their type (Braverman et al., 2018). More formally, a Bayesian game begins with one of C *contexts* ("types") $c \in [C]$ being drawn from a publicly known distribution $\mathcal{D}$ with probability p_c of outputting context c. This context c is told to the learner but not to the optimizer. Based on the context c, the learner chooses an action $j \in [N]$; simultaneously, the optimizer chooses an action $i \in [N]$. The optimizer

then receives utility $u_O(i, j, c)$, and the learner receives utility $u_L(i, j, c)$ – note that we allow both utilities to depend on the context c.

As with standard games, we eliminate the role of randomness in Bayesian games by allowing the optimizer and learner to play mixed strategies and assigning rewards deterministically. As with standard games, the optimizer plays a mixed strategy $\alpha \in \Delta([M])$. The learner simultaneously plays a function $\beta : [C] \to \Delta([N])$ mapping contexts to mixed strategies (representing which strategy the learner would play for each context). The optimizer then receives reward

$$u_O(\alpha, \beta) = \sum_{k=1}^{C} p_k u_O(\alpha, \beta(c_k), c_k) = \sum_{c=1}^{C} \sum_{i=1}^{M} \sum_{i=1}^{N} p_c \alpha_i \beta(c)_j u_O(i, j, c).$$

The learner's reward is computed similarly.

We are interested in settings where the optimizer and learner repeatedly play a game for T rounds. We write α^t to denote the optimizer's strategy in round t and β^t to denote the learner's strategy in round t. We will also insist that this repeated game is *full information*, in the sense that after each round, either player should be able to figure out their counterfactual utility if they had played a different mixed strategy that round (for example, this is the case when the mixed actions α^t and β^t of both players are made publicly known after round t and that both the optimizer and learner have full knowledge of their utility functions). This will allow the learner to play this game by running the learning algorithms detailed in the next section.

2.2. Learning algorithms, regret, and swap regret

As their name suggests, the learner will play the game by running an online learning algorithm to select their actions. We will consider the following (fractional, full-information, and deterministic) model for online learning:

A learner will face a decision between N actions for each of T rounds. An adversary begins by obliviously[2] selecting T reward vectors $r^1, r^2, \ldots, r^T \in [0, 1]^N$, where r_i^t represents the reward if the learner picks action i in round t. Then, for each round $1 \le t \le T$ the learner selects a distributional action $\beta^t \in \Delta([N])$ deterministically as a function of the reward vectors in previous rounds, i.e., $r_1, r_2, \ldots, r_{t-1}$. The learner then receives reward $\sum_{j=1}^{N} \beta_j^t r_j^t$ and the full reward vector r_t for round t is revealed to the learner.

Note that a learner can use such a learning algorithm to play in standard games. We evaluate a learning algorithm by providing bounds on some form of "regret". We consider two such notions: the *external regret* of an algorithm, and the *swap regret* of an algorithm. The external regret (or simply "regret") of a learning algorithm on a specific problem instance represents the gap between the total reward obtained by the learning algorithm and the best reward obtainable by playing the best single fixed action in hindsight; it is given by:

$$\mathsf{Reg} = \left(\max_{j^* \in [N]} \sum_{t=1}^{T} r_{j^*}^t \right) - \left(\sum_{t=1}^{T} \sum_{j=1}^{N} \beta_j^t r_j^t \right).$$

A learning algorithm is *low-regret* if it sustains $o(T)$ external regret on any problem instance with T rounds (and a fixed number of actions). It is well known that there exist efficient low-regret

2. Since the learner here is deterministic, it actually does not make a difference whether we allow the adversary to be adaptive or not; an oblivious adversary can simply simulate the actions of the learner.

algorithms in this setting which sustain regret at most $O(\sqrt{T \log N})$ (Littlestone and Warmuth, 1994b; Freund and Schapire, 1997b). Interestingly, the property of being low-regret is not sufficient to guarantee good performance in the optimizer-learner settings described in Section 2.1; there are games where the optimizer can get much more than their Stackelberg value if the learner plays certain low-regret algorithms. As we will show, to guarantee that this does not occur, the learner must play an algorithm with low swap-regret.

The swap regret of a learning algorithm on a specific problem instance represents the gap between the total reward obtained by the learning algorithm, and the maximum award they could obtain in hindsight if they had applied a deterministic *swap function* to their actions (i.e., playing action 2 instead of action 1 every time they played action 1 with any weight). Formally, we can define the swap regret as follows:

$$\mathsf{SwapReg} = \left(\max_{\pi:[N]\to[N]} \sum_{t=1}^{T} \sum_{j=1}^{N} \beta_j^t r_{\pi(j)}^t \right) - \left(\sum_{t=1}^{T} \sum_{j=1}^{N} \beta_j^t r_j^t \right).$$

A learning algorithm is *low-swap-regret* if it sustains $o(T)$ swap regret on any problem instance with T rounds. As with external regret, it is known there exist low-swap-regret algorithms. For example, the construction of Blum and Mansour (2007) demonstrates how to devise an algorithm with swap regret $O(\sqrt{TN \log N})$.

Until now, we have described a form of online learning that can be used to play standard games. To play Bayesian games, we need a form of online *contextual* learning – we defer discussion of this to the beginning of Section 4.

2.3. Stackelberg equilibria and strategies

One of the primary benchmarks that we will use to measure the performance of the optimizer is the optimizer's value in the Stackelberg equilibrium of the one-shot game.

Let G be a standard game, and for each mixed strategy $\alpha \in \Delta([M])$, define the learner's best-response function $\mathsf{BR}(\alpha) = \operatorname{argmax}_{j\in[N]} u_L(\alpha, j)$. We then define the Stackelberg value of G to be the value

$$\mathsf{Val}(G) = \max_{\alpha} \max_{\beta\in\mathsf{BR}(\alpha)} u_O(\alpha, \beta).$$

We can similarly define the Stackelberg value $\mathsf{Val}(G)$ for a Bayesian game, with the only difference that now the learner's best response $\mathsf{BR}(\alpha) = \operatorname{argmax}_{\beta\in[N]^{[C]}} u_L(\alpha, j)$ is taken over all strategies $\beta(c) : [C] \to [N]$ mapping contexts to actions.

Intuitively, the Stackelberg value represents the maximum value the optimizer can obtain by playing a fixed strategy against a strategic learner. Note that: a) we allow the optimizer to play a mixed strategy instead of just a pure strategy (so this is what is occasionally referred to as a Stackelberg mixed strategy, e.g. in Conitzer (2016)) and b) we break ties for the learner in favor of the optimizer.

The Stackelberg value is a benchmark that arises naturally in our setting for the following reason[3]: if the optimizer is playing a game G for T rounds against a learner running a low-regret

3. For the case of standard games, this was shown in Deng et al. (2019); we include the straightforward generalization to Bayesian games (and more generally, polytope games) in Appendix E.

algorithm, then the optimizer can guarantee (under some mild conditions on G) that they receive reward at least $\mathsf{Val}(G)T - o(T)$. Moreover, the optimizer can accomplish this by playing their fixed Stackelberg strategy every round. The central goal of this paper is to understand when the optimizer can significantly outperform this benchmark by playing a dynamic strategy.

3. Standard games

We begin our discussion with standard games. In Deng et al. (2019), the authors show that if an optimizer is playing a learner with low swap regret, the optimizer can get no more than $\mathsf{Val}(G)T + o(T)$ utility. We complement this result by showing that low swap regret is *necessary*; if a learner is playing an algorithm that is not low swap regret, then it is possible to construct a game G where an optimizer can gain significantly ($\Omega(T)$) more than their Stackelberg value by playing some dynamic strategy against this learner.

The main argument is encapsulated in the following lemma, which shows how to convert a high swap-regret online learning instance for the learner into a game where the optimizer outperforms Stackelberg.

Lemma 1 *Let $\mathcal{A}$ be a learning algorithm which incurs swap-regret R on some online learning instance with N actions and T rounds. Then there exists a standard game G (with N actions for the learner and $M = 2^N$ actions for the optimizer) such that if an optimizer plays T rounds of G against a learner running $\mathcal{A}$, the optimizer can receive a total reward of $\mathsf{Val}(G)T + \frac{1}{2}R$.*

Proof Recall that in our model, an online learning instance with N actions and T rounds is completely specified by a sequence of T reward vectors $r^t \in [-1,1]^N$. Fix r^t to be the bad instance mentioned in the theorem statement for algorithm $\mathcal{A}$, and let $\beta^t \in \Delta([N])$ denote the action of the learner in round t. Since the regret of $\mathcal{A}$ on this bad instance is R, this implies that for some swap function $\pi : [N] \to [N]$, we have that

$$\left(\sum_{t=1}^{T}\sum_{j=1}^{N}\beta_j^t r_{\pi(j)}^t\right) - \left(\sum_{t=1}^{T}\sum_{j=1}^{N}\beta_j^t r_j^t\right) = R. \tag{1}$$

We will now show how to use the reward r^t to construct a game where the optimizer can do better than the Stackelberg equilibrium. For now, we will construct a game G where the learner has N actions and the optimizer has $M = T$ actions – we will later show how to decrease M to a value independent of T.

We begin by specifying the learner's payoffs. Unsurprisingly, these will be drawn directly from the learner's rewards in the online learning problem: specifically, if the optimizer plays an action $1 \le i \le T$ and the learner plays action $j \in [N]$, the learner will receive the reward they would have received if they played j in the ith round of the online learning instance, namely $u_L(i,j) = r_j^i$. Note that this has the property that if the optimizer plays action t in round t, the learning algorithm $\mathcal{A}$ will see exactly the learning instance mentioned above and therefore play β^t in each round t.

The more interesting aspect of constructing this game is selecting payoffs for the optimizer. We do this as follows. If the optimizer plays action i and the learner plays action j, we set $u_O(i,j) = \frac{1}{2}(r_{\pi(j)}^i - r_j^i)$ (note that since $r^i \in [-1,1]^N$, this scaling ensures that $|u_O(i,j)| \le 1$). This lets us rewrite (1) in the form

$$\sum_{t=1}^{T} u_O(t, \beta^t) = \frac{R}{2}. \tag{2}$$

But note that the LHS of (2) is exactly the total payoff the optimizer receives if they play action t in round T against a learner running $\mathcal{A}$. Therefore the optimizer can obtain a total reward of $R/2$ against such a learner.

We now argue that the Stackelberg value of this game is at most 0. To do this, imagine that in a single-shot Stackelberg instance of G, the optimizer plays a mixed strategy $\alpha \in \Delta([M])$ and the learner best-responds by playing $j \in [N]$. Then note that

$$u_O(\alpha, j) = \frac{1}{2}\sum_{i=1}^{M} \alpha_i \left(r^i_{\pi(j)} - r^i_j\right) = \frac{1}{2}\left(u_L(\alpha, \pi(j)) - u_L(\alpha, j)\right).$$

But since j is a best response to α for the learner, we must have that $u_L(\alpha, j) \geq u_L(\alpha, \pi(j))$; it follows that $u_O(\alpha, j) \leq 0$, and therefore $\mathsf{Val}(G) \leq 0$. This completes our proof of the existence of a game (albeit one with many actions for the optimizer) where the optimizer can get at least $R/2$ more than the Stackelberg value of the game.

We now show how to construct a game G' with the same property, but where the optimizer only has $M = 2^N$ actions. To do this, observe that if you fix an action $i \in [M]$ for the optimizer, both the optimizer's payoff $u_O(i, \cdot)$ and learner's payoff $u_L(i, \cdot)$ are linear functions of r^i. This motivates the following construction. For each $1 \leq i \leq M$ let s^i be the ith element of $S = \{-1, 1\}^N$ (for some arbitrary labelling of the M elements of S), and let:

$$\begin{aligned} u_L(i, j) &= s^i_j \\ u_O(i, j) &= \frac{1}{2}(s^i_{\pi(j)} - s^i_j). \end{aligned}$$

Since S contains the vertices of $[-1, 1]^N$, by Caratheodory's theorem, for each $r^t \in [-1, 1]^N$, there exists an $\alpha^t \in \Delta([M])$ such that $r^t = \sum_{i=1}^{M} \alpha^t_i s^i$. In particular, this implies that $u_L(\alpha^t, j) = r^t_j$ and $u_O(\alpha^t, j) = (r^t_{\pi(j)} - r^t_j)/2$, so by playing the sequence of actions α^t, the optimizer can still guarantee total reward $R/2$. Moreover, the Stackelberg value of G' is still 0, by the same logic as before. ■

Remark 2 *There is a sense in which the exponential dependence of M on N in Lemma 1 is unnecessary. In particular, if there exists a bad instance for $\mathcal{A}$ where all the rewards lie in $[-1/N, 1/N]^N$, then we can conduct the same construction at the end of Lemma 1, but with the set of $2N$ vectors $S = \{e_1, -e_1, e_2, -e_2, \ldots, e_N, -e_N\}$ (in general, it is only necessary that it is possible to write each reward vector r^t as a convex combination of elements of S). Furthermore, even the game with 2^N actions is quite structured, and it is possible for the optimizer to efficiently compute the Stackelberg equilibrium of this game – see Appendix B for details.*

We now apply Lemma 1 to show that if the learner is *not* running a low-swap-regret algorithm, there is a game G where an optimizer can get $\Omega(T)$ more than the Stackelberg value.

Theorem 3 *If $\mathcal{A}$ is not a low-swap-regret learning algorithm, then there exists a game G where if an optimizer plays T rounds of G against a learner running $\mathcal{A}$, the optimizer can get reward at least $\mathsf{Val}(G)T + \Omega(T)$ for infinitely many values of T.*

4. Bayesian games

We now begin our exploration of Bayesian games. The model for online learning introduced in Section 2.2 covers algorithms a learner can use to play standard games. In order to play a Bayesian game, the learner needs to use an algorithm for *online contextual learning*.

Our model for online contextual learning will be similar to the model for online learning presented in Section 2.2 (in that it will be fractional, full-information, and deterministic), with some differences analogous to the difference between standard games and Bayesian games. Specifically, in online contextual learning:

- There is a publicly known distribution $\mathcal{D}$ over C contexts, where context $c \in [C]$ occurs with probability p_c.
- Instead of picking a sequence of T reward vectors in $[0,1]^N$, the adversary picks T reward vectors in $r^t \in [0,1]^{N \times C}$, where $r^t_{i,c}$ represents the reward if the learner picks action $i \in [N]$ in context $c \in [C]$.
- Instead of selecting a single mixed action in round t, the learner instead picks a function $\beta^t : [C] \to \Delta([N])$ mapping contexts to distributions over actions.
- In round t, the learner receives reward $\sum_{c=1}^{C} \sum_{j=1}^{N} p_c \beta^t(c)_j r^t_{j,c}$. The learner wishes to maximize their total reward over all T rounds.

It is straightforward to define a notion of external regret for online contextual learning. If we let

$$\mathsf{Reg} = \left(\max_{f^* : [C] \to [N]} \sum_{t=1}^{T} \sum_{c=1}^{C} p_c r^t_{f^*(c),c} \right) - \left(\sum_{t=1}^{T} \sum_{c=1}^{C} \sum_{j=1}^{N} p_c \beta^t(c)_j r^t_{j,c} \right),$$

then Reg represents the difference in utilities on this problem instance between the learning algorithm and the learner who in each context c plays the best-in-hindsight action $f^*(c)$ for this context. It is similarly straightforward to construct algorithms for online contextual learning which incur external regret at most $O(\sqrt{T \log N})$ in the above setting (for example, one can simply run a low-regret online learning algorithm per context).

Interestingly, it is much less clear what the correct analogue of swap regret for online contextual learning is. Many obvious guesses (such as the notion of regret obtained by running a low-swap-regret algorithm per context) turn out to be "incorrect", in that they do not allow us to prove analogues of Theorem 3 for Bayesian games (we explore these in more detail in Section D.2). Ultimately, in this section we will define two notions of swap-regret with the following guarantees:

- **Polytope swap regret**: If a contextual learning algorithm $\mathcal{A}$ has low polytope swap regret, then an optimizer can get at most $\mathsf{Val}(G)T + o(T)$ reward when playing a Bayesian game G for T rounds against a learner running $\mathcal{A}$.
- **Linear swap regret**: If a contextual learning algorithm $\mathcal{A}$ *does not have* low linear swap regret, there exists a Bayesian game G where if an optimizer plays T rounds of G against a learner using $\mathcal{A}$, the optimizer can get at least $\mathsf{Val}(G)T + \Omega(T)$ reward (for infinitely many values of T).

4.1. Polytope games and polytope learning

Instead of defining these notions of swap regret directly for Bayesian games, we will find it convenient to define a more general class of games and learning algorithms that generalizes both forms of games (standard and Bayesian) and learning (regular and contextual) that we have considered so

far. We call this class of games *polytope games*, and the corresponding variety of learning *polytope learning*. Polytope learning will turn out to be essentially equivalent to online linear optimization (the major difference being that we restrict the action set to be a polytope as opposed to an arbitrary convex set), but we refer to it this way to emphasize the connection to polytope games.

In a polytope game G, the learner must select a point x belonging to some bounded polytope $\mathcal{P} \subseteq [-1,1]^d$. Simultaneously, the optimizer must select a point $q = (r,s)$ belonging to some polytope $\mathcal{Q} \subseteq [-1,1]^d \times [-1,1]^d$ (i.e, both r and s are d-dimensional vectors). The optimizer then receives utility $\langle s, x\rangle$, and the learner receives utility $\langle r, x\rangle$. As with standard games and Bayesian games, we can easily define the Stackelberg value $\mathsf{Val}(G)$ of this game to be the maximum value an optimizer can guarantee by playing a fixed[4] action $(r,s) \in \mathcal{Q}$ with the learner best responding.

To play a repeated polytope game, a learner can run a polytope learning algorithm. An instance of polytope learning is specified by a polytope $\mathcal{P}$ and a sequence of T d-dimensional reward vectors $r^1, r^2, \ldots, r^T \in [-1,1]^d$. At the beginning of round t, the learner must select a point $x^t \in \mathcal{P}$; the learner then receives reward $\langle r^t, x^t\rangle$. The goal of the learner is to maximize their total reward. As with our previous learners, we will restrict our attention to deterministic algorithms (i.e., algorithms $\mathcal{A}$ that choose x^t deterministically as a function of $r^1, r^2, \ldots, r^{t-1}$).

Here are some examples of polytope games and polytope learning:

1) Standard games and online learning. Given a standard game G, let $\mathcal{P} = \Delta([N])$ and let $\mathcal{Q} = \mathrm{conv}(\{(r_i, s_i)\}_{i=1}^M)$, where for each $i \in [M]$

$$\begin{aligned} r_i &= (u_L(i,1), u_L(i,2), \ldots, u_L(i,N)) \in \mathbb{R}^N \\ s_i &= (u_O(i,1), u_O(i,2), \ldots, u_O(i,N)) \in \mathbb{R}^N. \end{aligned}$$

Then the polytope game G' defined by $\mathcal{P}$ and $\mathcal{Q}$ is "equivalent" to the standard game G in the following sense: the map f which sends $\alpha \in \Delta([M])$ to the point $q = \sum_{i=1}^M \alpha_i (r_i, s_i) \in \mathcal{Q}$ and the identity map g which sends $\beta \in \Delta([N])$ to $\beta \in \mathcal{P}$ together have the property that $u_O(\alpha, \beta) = u_O(f(\alpha), g(\beta))$ and $u_L(\alpha, \beta) = u_L(f(\alpha), g(\beta))$; moreover, f and g are both surjective onto $\mathcal{P}$ and $\mathcal{Q}$ respectively. Intuitively, we can translate any strategy profile in G to an "equivalent" strategy profile in G' and vice versa. Note that there may be multiple strategy profiles in G that map to the same strategy profile in G'; this corresponds to the fact that it is possible for two different mixed strategies for the learner to always result in the same payoffs.

Similarly, polytope learning over the simplex $\mathcal{P} = \Delta([N])$ is equivalent to online learning. Here the reduction is immediate – the online learning problem in Section 2.2 is exactly polytope learning with $\mathcal{P} = \Delta([N])$.

2) Bayesian games and contextual learning. Given a Bayesian game G, let $\mathcal{P} = \Delta([N])^C \subseteq \mathbb{R}^{N \times C}$ (one can think of $\mathcal{P}$ as the convex hull of all C-by-N stochastic matrices), and let $\mathcal{Q} = \mathrm{conv}(\{(r_i, s_i)\}_{i=1}^M)$, where $s_i \in \mathbb{R}^{N \times C}$ is defined via $s_{i,j,c} = p_c u_O(i,j,c)$ and $r_i \in \mathbb{R}^{N \times C}$ is defined via $r_{i,j,c} = p_c u_L(i,j,c)$. Then the polytope game G' defined by $\mathcal{P}$ and $\mathcal{Q}$ is equivalent to the Bayesian game G (in the same sense and for the same reason as above).

Similarly, polytope learning over the polytope $\mathcal{P} = \Delta([N])^C$ is the contextual learning problem we defined at the beginning of this section (with the subtle difference that the reward vectors $r_{i,c}^t$ should be scaled by the probabilities p_c).

4. Note that since the reward function is linear and the action spaces of both the optimizer and learner are convex, there is no need to consider mixed actions – for either player, any mixed action is equivalent to some single action.

In general, we will want to fix a polytope $\mathcal{P}$ (e.g. the Bayesian game polytope) and consider the class of polytope games where the learner's actions belong to $\mathcal{P}$. We call such games *$\mathcal{P}$-games*; note that such games are entirely specified by the optimizer's action set $\mathcal{Q}$. Similarly, when discussing polytope learning, we will want to fix a polytope $\mathcal{P}$ and consider the class of polytope learning instances / algorithms where the learner's actions must belong to $\mathcal{P}$. We call this problem *$\mathcal{P}$-learning* for short, and such algorithms *$\mathcal{P}$-learning algorithms*.

4.2. Linear swap regret

We begin by defining a variant of swap regret for polytope learning that we call *linear swap regret*. In this variant of swap regret, a learner compares their utility to the utility they would have received if they applied a static linear transformation to each of their actions.

Formally, let $\mathcal{P}$ be a polytope and consider an instance of the $\mathcal{P}$-learning problem where the rewards are $r^1, r^2, \ldots, r^T$ and the actions of the learner are given by $x^1, x^2, \ldots, x^T$. Then the linear swap regret of the learner on this instance is given by

$$\mathsf{LinSwapReg} = \max_{M \in \mathcal{M}(\mathcal{P})} \sum_{t=1}^{T} \langle r^t, Mx^t \rangle - \sum_{t=1}^{T} \langle r^t, x^t \rangle,$$

where $\mathcal{M}(\mathcal{P})$ is the set of all linear transformations $M : \mathbb{R}^d \to \mathbb{R}^d$ that satisfy $Mx \in \mathcal{P}$ for all $x \in \mathcal{P}$ (i.e., the set of linear transformations that are contractions of $\mathcal{P}$).

As with other forms of regret, we say that a polytope learning algorithm $\mathcal{A}$ has low linear swap regret if it sustains $o(T)$ linear swap regret on any problem instance with T rounds. We now show that in order for the optimizer to get no more than the Stackelberg value, it is necessary for the learner to run a low linear swap regret algorithm. As in Section 3, we begin by showing this is true on a per instance level.

Lemma 4 *Fix a polytope $\mathcal{P} \subseteq [-1, 1]^d$. Let $\mathcal{A}$ be a $\mathcal{P}$-learning algorithm which has linear swap regret R on some problem instance. Then there exists a $\mathcal{P}$-game such that if an optimizer plays T rounds of G against a learner running $\mathcal{A}$, the optimizer can receive a total reward of $\mathsf{Val}(G)T + R/(\lambda + 1)$, where $\lambda = \max_{M \in \mathcal{M}(\mathcal{P})} \|M\|_1$.*

When $\mathcal{P} = \Delta([N])$, linear swap regret reduces to the ordinary notion of swap regret. Indeed, the set $\mathcal{M}(\Delta([N]))$ of linear contractions of $\Delta([N])$ is exactly the set of N-by-N stochastic matrices, and the extreme points of $\mathcal{M}(\Delta([N]))$ are the N-by-N 0/1-matrices which contain exactly one 1 in each row. These matrices correspond to (and act the same way on π) as the N^N swap functions $\pi : [N] \to [N]$.

As with standard games, we can apply Lemma 4 to show that if $\mathcal{A}$ does not have low linear swap regret, then it is possible for an optimizer to get strictly more than their Stackelberg value by playing against a learner running $\mathcal{A}$ in a fixed game.

Theorem 5 *Fix a polytope $\mathcal{P} \subseteq [-1, 1]^d$. If a $\mathcal{P}$-learning algorithm $\mathcal{A}$ does not have low linear swap regret, then there exists a $\mathcal{P}$-game G where if an optimizer plays T rounds of G against a learner running $\mathcal{A}$, the optimizer can get at least $\mathsf{Val}(G)T + \Omega(T)$ reward (for infinitely many values of T).*

4.3. Polytope swap regret

We now define a second notion of swap regret for $\mathcal{P}$-learning algorithms which we call *polytope swap regret*. Whereas having low linear swap regret is a necessary condition to guarantee that the optimizer receives at most $\mathsf{Val}(G)T + o(T)$ utility, we will show that having low polytope swap regret is a sufficient condition for the same guarantee.

The intuition behind polytope swap regret is that we want to compete against an arbitrary swap function π on the vertices of $\mathcal{P}$ (i.e., a function that maps each vertex of $\mathcal{P}$ to some other vertex). This makes sense if the learner only ever plays actions which are vertices of $\mathcal{P}$, but it is less clear how π should act on an interior point x of $\mathcal{P}$. One way to define an action of π on x is to write x as a convex combination of vertices, use π to map each of these vertices to (possibly) new vertices, and take the corresponding convex combination of these new vertices to obtain a new point x'. This works, but in many cases there will many ways to write x as a convex combination of the vertices of $\mathcal{P}$. When computing polytope swap regret, we will choose the best (regret-minimizing) decompositions of all actions x^i in our given problem instance that minimizes the worst-case regret for the worst possible swap function π.

Formally, given a polytope $\mathcal{P}$, let $\mathcal{V}(\mathcal{P})$ be the set of vertices of $\mathcal{P}$, and let $V = |\mathcal{V}(\mathcal{P})|$ be the number of vertices of $\mathcal{P}$. We say that $\rho \in \Delta(\mathcal{V}(\mathcal{P}))$ is a *vertex decomposition* of a point $x \in \mathcal{P}$ if $\sum_{v \in \mathcal{V}(\mathcal{P})} \rho_v v = x$; likewise, given a vertex decomposition ρ, we will let $\overline{\rho} = \sum_{v \in \mathcal{V}(\mathcal{P})} \rho_v v$ denote the point in $\mathcal{P}$ for which ρ is a vertex decomposition.

We call functions $\pi : \mathcal{V}(\mathcal{P}) \to \mathcal{V}(\mathcal{P})$ that map vertices of $\mathcal{P}$ to vertices of $\mathcal{P}$ *vertex swap functions*. We extend π to act on vertex decompositions by letting $\pi(\rho)_v = \sum_{v' \in \pi^{-1}(v)} \rho_{v'}$; note that under this definition, $\overline{\pi(\rho)} = \sum_{v \in \mathcal{V}(\mathcal{P})} \rho_v \pi(v)$.

Finally, consider an instance of the $\mathcal{P}$-learning problem with reward vectors $r^1, r^2, \ldots, r^T \in [-1, 1]^d$, where a learner running algorithm $\mathcal{A}$ plays actions $x^1, \ldots, x^T \in \mathcal{P}$. We define the polytope swap regret of $\mathcal{A}$ on this instance as

$$\mathsf{PolySwapReg} = \left(\min_{\rho^t \,|\, \overline{\rho^t} = x^t} \max_{\pi : \mathcal{V}(\mathcal{P}) \to \mathcal{V}(\mathcal{P})} \sum_{t=1}^{T} \left\langle r^t, \overline{\pi(\rho^t)} \right\rangle \right) - \sum_{t=1}^{T} \langle r^t, x^t \rangle.$$

Here the outer minimum is over all sequences of vertex decompositions ρ^t of the actions x^t, and the inner maximum is over all vertex swap functions π. An alternate way of thinking about this benchmark is in the form of a zero-sum game: the learner, after they have played all their actions x^t (but before they see π) chooses a vertex decomposition ρ^t for each of their actions. The adversary then observes these vertex decompositions and responds with the vertex swap function π which maximizes the counterfactual utility of the transformed action sequence obtained by applying π to each ρ^t. The learner wishes to choose their original decompositions to minimize this maximum counterfactual utility.

As always, we say the $\mathcal{P}$-learning algorithm $\mathcal{A}$ has low polytope swap regret if it incurs at most $o(T)$ polytope swap regret for all $\mathcal{P}$-learning instances.

Theorem 6 *Let $\mathcal{A}$ be a $\mathcal{P}$-learning algorithm with low polytope swap regret. Then if an optimizer plays T rounds of a $\mathcal{P}$-game G against a learner running $\mathcal{A}$, the optimizer can get at most $\mathsf{Val}(G)T + o(T)$ reward.*

4.4. Separating linear and polytope swap regret in Bayesian games

We establish separation of linear and polytope swap regrets in Appendix C and pose an open question about polytope swap regret (Question 1).

Acknowledgements

YM received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 882396), the Israel Science Foundation(grant number 993/17), Tel Aviv University Center for AI and Data Science (TAD), and the Yandex Initiative for Machine Learning at Tel Aviv University.

The authors would like to thank Jieming Mao and Yuan Deng for helpful discussions about this work.

References

Shipra Agrawal, Constantinos Daskalakis, Vahab S. Mirrokni, and Balasubramanian Sivan. Robust repeated auctions under heterogeneous buyer behavior. In *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, page 171, 2018.

Robert J. Aumann. Subjectivity and correlation in randomized strategies. *Journal of Mathematical Economics*, 1(1):67 – 96, 1974. ISSN 0304-4068.

Avrim Blum and Yishay Mansour. From external to internal regret. In Peter Auer and Ron Meir, editors, *Learning Theory*, 2005.

Avrim Blum and Yishay Mansour. From external to internal regret. *Journal of Machine Learning Research*, 8:1307–1324, 2007.

Mark Braverman, Jieming Mao, Jon Schneider, and Matt Weinberg. Selling to a no-regret buyer. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 523–538, 2018.

Nicolò Cesa-Bianchi and Gábor Lugosi. Potential-based algorithms in on-line prediction and game theory. *Machine Learning*, 51(3):239–261, Jun 2003.

Nicolò Cesa-Bianchi, Yoav Freund, David Haussler, David P. Helmbold, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. *J. ACM*, 44(3):427–485, May 1997. ISSN 0004-5411.

Miroslav Chlebík and Janka Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Information and Computation*, 206(11):1264–1275, 2008.

Vincent Conitzer. On stackelberg mixed strategies. *Synthese*, 193(3):689–703, 2016.

Yuan Deng, Jon Schneider, and Balasubramanian Sivan. Strategizing against no-regret learners. *Advances in neural information processing systems*, 32, 2019.

Dean P. Foster and Rakesh V. Vohra. A randomization rule for selecting forecasts. *Operations Research*, 41(4):704–709, 1993.

Dean P. Foster and Rakesh V. Vohra. Calibrated learning and correlated equilibrium. *Games and Economic Behavior*, 21(1):40 – 55, 1997.

Dean P. Foster and Rakesh V. Vohra. Asymptotic calibration. *Biometrika*, 85(2):379–390, 06 1998.

Dean P. Foster and Rakesh V. Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29(1):7 – 35, 1999.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997a.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997b.

Yoav Freund and Robert E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1):79 – 103, 1999.

Guru Guruganesh, Jon Schneider, and Joshua R Wang. Contracts under moral hazard and adverse selection. In *Proceedings of the 22nd ACM Conference on Economics and Computation*, pages 563–582, 2021.

James Hannan. Approximation to bayes risk in repeated plays. *Contributions to the Theory of Games*, 3:97–139, 1957.

Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

N. Littlestone and M.K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212 – 261, 1994a.

Nick Littlestone and Manfred K Warmuth. The weighted majority algorithm. *Information and computation*, 108(2):212–261, 1994b.

Bernhard Von Stengel and Shmuel Zamir. Leadership games with convex strategy sets. *Games and Economic Behavior*, 69(2):446–457, 2010.

Appendix A. Related Work

There is a very large amount of literature on the outcome of interaction between strategic agents in single-shot and repeated games, offering us reasonably complete picture of the landscape here. Likewise, when learning agents interact repeatedly in a game, we have a good understanding of what equilibria they lead to, depending on the learning algorithms employed by the learners (more on this below). However, the nature of this interaction between a strategic agent and a learner is much less studied, and has gained momentum only in the last few years. The work closest to ours in this space is that of Deng et al. (2019) who initiate the study of this optimizer-learner interaction and draw some very interesting conclusions detailed earlier. The recent work of Braverman et al. (2018) is also quite close to ours. They study the specific 2-player Bayesian game of an auction between a single seller and single buyer. The seller's choice of the auction to run represents his action, and the buyer's bid represents her action. Our work generalizes both Deng et al. (2019) and Braverman et al. (2018) by studying general Bayesian games, and also addresses questions beyond what was asked in those works. Both Deng et al. and Braverman et al. show that regardless of the specific algorithm used by the learner (buyer), as long as the buyer plays a no-regret learning algorithm, the optimizer (seller) can always earn at least the Stackelberg value in a single shot game. Our Lemma 16 in Appendix is a direct generalization of both these results to arbitrary Bayesian games without any structure. Both Deng et al. and Braverman et al. show that there exist no-regret strategies for the learner (buyer) that guarantee that the optimizer (seller) cannot get anything better than the single-shot optimal payoff. Our Theorem 6 on polytope swap regret is a directly generalization of both these results showing that when learner has a low polytope swap regret, the optimizer cannot earn much more than the Bayesian Stackelberg value of the game. Neither Deng et al. nor Braverman et al. provide necessary and sufficient conditions for the learner to cap the optimizer payoff at Stackelberg value. We provide such necessary and sufficient conditions for standard games in[5] Theorem 3. We provide a partial answer for Bayesian games, where we get a sufficient condition that we conjecture to be necessary as well.

To discuss literature on the outcome of learning agents interacting with each other, we begin with the different notions of regret. The usual notion of regret, without the swap qualification, is often referred to as external-regret (see Hannan (1957), Foster and Vohra (1993), Littlestone and Warmuth (1994a), Freund and Schapire (1997a), Freund and Schapire (1999), Cesa-Bianchi et al. (1997)). There is a stronger notion of regret called internal regret that was defined earlier in Foster and Vohra (1998), which allows all occurrences of a given action x to be replaced by another action y. Many no-internal-regret algorithms have been designed (see for example Hart and Mas-Colell (2000), Foster and Vohra (1997, 1998, 1999), Cesa-Bianchi and Lugosi (2003)). The still stronger notion of swap regret was introduced in Blum and Mansour (2005), and it allows one to simultaneously swap several pairs of actions. Blum and Mansour show how to efficiently convert a no-regret algorithm to a no-swap-regret algorithm. One of the reasons behind the importance of internal and swap regret is their close connection to the central notion of correlated equilibrium introduced by Aumann (1974). In a general n players game, a distribution over action profiles of all the players is a correlated equilibrium if every player has zero internal regret. When all players use algorithms with no-internal-regret guarantees, the time averaged strategies of the players converges to a correlated equilibrium (see Hart and Mas-Colell (2000)). When all players simply use algorithms with no-external-regret guarantees, the time averaged strategies of the players converges

5. Theorem 3 establishes necessity, as sufficiency is already known from Deng et al. (2019).

to the weaker notion of coarse correlated equilibrium. When the game is a zero-sum game, the time-averaged strategies of players employing no-external-regret dynamics converges to the Nash equilbrium of the game.

On the special 2-player game of selling to a buyer in an auction, Agrawal et al. (2018) study a setting similar to Braverman et al. but also consider other types of buyer behavior apart from learning, and show to how to robustly optimize against various buyer strategies in an auction.

Appendix B. Efficiently computing the Stackelberg equilibrium of the lower bound game

In this section, we show that it is possible for the optimizer to efficiently compute the Stackelberg equilibrium of the lower-bound game introduced in Lemma 1 in time polynomial in N, despite that game having 2^N actions. This will follow for the same reason as mentioned in Remark 2, namely that this game has a very strong linear structure and is really just defined by a vector of dimension N.

Lemma 7 *Let G be the game defined in the proof of Lemma 1, with N actions for the learner and $M = 2^N$ actions for the optimizer. It is possible to compute the Stackelberg strategy for the optimizer in time polynomial in N.*

Proof Consider the following alternate succinct description of the game G constructed in Lemma 1. The optimizer selects a vector $r \in [-1, 1]^N$ and the learner chooses an action $j \in [N]$; the optimizer then receives reward $r_{\pi(j)} - r_j$, and the learner receives reward r_j. This corresponds to the original game in the following way; for each $v \in S = \{-1, 1\}^N$, consider the mixed strategy $\alpha \in \Delta(S)$ for the optimizer where the optimizer plays v with probability α_v. Then, if we let $r = \sum_{v \in S} \alpha_v v$, it follows from the definitions of $u_L(i, j)$ and $u_R(i, j)$ that the rewards are as described above (in particular, both the learner and optimizer's reward functions are linear functions of v, so they can be written as a linear function of r).

Now, it is possible to find the Stackelberg equilibrium of this new game in polynomial time by solving a sequence of small LPs. In particular, for each j, we can solve a straightforward linear program to find the optimal mixed action $r(j)$ which maximizes the optimizer's reward conditioned on j being the best response for the learner. Taking the maximum value over all LPs (and the corresponding strategy) gives a Stackelberg strategy for the original game. ■

Appendix C. Separating linear and polytope swap regrets

In Sections 4.2 and 4.3, we presented two different definitions of swap regret – linear swap regret and polytope swap regret – for the very general setting of polytope learning. Together, they form necessary and sufficient conditions for which learning algorithms are robust to strategic behavior in polytope games: algorithms with low polytope swap regret are always robust, whereas algorithms with high linear swap regret are not robust. Interestingly, when $\mathcal{P}$ is the simplex $\Delta([N])$ both these notions of swap regret are equal, and they both reduce to the ordinary notion of swap regret in online learning.

We now return our attention to the setting of Bayesian games and contextual learning, where the relevant polytope $\mathcal{P} = \Delta([N])^C$ is the product of C N-simplices. Just as swap regret is the

"correct" regret definition for characterizing robustness in standard games, we want to understand what is the correct definition of regret for characterizing robustness in Bayesian games. Is it polytope swap regret, or linear swap regret, or some as-yet-undefined notion that lies between these two regret measures?

We do not yet know the answer to this question, but we provide some partial progress towards resolving it. To begin, we show that unlike for standard games, for the Bayesian game polytope polytope swap regret and linear swap regret can differ significantly (even when $K = C = 2$).

Theorem 8 *Let $\mathcal{P} = \Delta([N])^C$. There exists a $\mathcal{P}$-learning instance with $K = C = 2$ where* $\mathsf{PolySwapReg} = \Omega(T)$ *and where* $\mathsf{LinSwapReg} = 0$.

This separation implies that at least one of polytope swap regret and linear swap regret do not tightly characterize robustness against strategic behavior in Bayesian games. It turns out (as we discovered through computational search) that linear swap regret is not tight: it is possible to extend the example in Theorem 8 to a Bayesian game where the optimizer can get $\Omega(T)$ more than their Stackelberg value, despite the learner running a low linear regret learning algorithm.

Theorem 9 *Let $\mathcal{P} = \Delta([N])^C$. There exists a $\mathcal{P}$-game (i.e., Bayesian game) G with $K = C = 2$ and a low linear regret $\mathcal{P}$-learning algorithm (i.e., contextual learner) $\mathcal{A}$ where if an optimizer plays T rounds of G against a learner running $\mathcal{A}$, the optimizer can get reward at least $V(G)T + \Omega(T)$.*

its On the other hand, we are unable to find any example of a contextual learning instance where a learner incurs high polytope swap regret that can be extended to a game where the optimizer *cannot* get more than their Stackelberg value. We conjecture that polytope swap regret is the "correct" notion of regret, both for Bayesian games and more generally for polytope games. Specifically, we pose the following open question:

Question 1 *Let $\mathcal{A}$ be a contextual learning algorithm which does not have low polytope swap regret. Is it possible for an optimizer to get at least $\mathsf{Val}(G)T + \Omega(T)$ reward when playing T rounds of a Bayesian game G against an optimizer running $\mathcal{A}$?*

Appendix D. Algorithmic considerations

D.1. A low polytope swap regret algorithm

Thus far, we have focused on characterizing relevant definitions of regret for the learner without actually discussing how to construct learning algorithms that minimize these forms of regret. We remedy this here by showing how to convert a traditional low-swap-regret algorithm to an algorithm for polytope learning which obtains $o(T)$ polytope swap regret (and hence $o(T)$ linear swap regret as well).

Let $\mathcal{A}$ be a low-swap-regret algorithm for online swap regret (e.g., the one presented in Blum and Mansour (2007)). The idea is simple: if $\mathcal{A}'$ is faced with a $\mathcal{P}$-learning instance, $\mathcal{A}'$ begins by initializing a copy of $\mathcal{A}$ with one action for each vertex $v \in \mathcal{V}(\mathcal{P})$. At the beginning of each round t, $\mathcal{A}'$ queries $\mathcal{A}$ for a mixed strategy $\beta^t \in \Delta(\mathcal{V}(\mathcal{P}))$ and then plays the corresponding convex combination of the vertices of $\mathcal{P}$, the point $x^t = \sum_{v \in \mathcal{V}(\mathcal{P})} \beta^t_v v \in \mathcal{P}$. Then $\mathcal{A}'$ will compute the reward $\langle r^t, v \rangle$ for each vertex v and pass this collection of rewards to $\mathcal{A}$.

We show that swap regret guarantees for $\mathcal{A}$ directly translate to polytope swap regret guarantees for $\mathcal{A}'$.

Theorem 10 *Let $\mathcal{A}$ be an online learning algorithm which incurs swap regret at most $R(N,T)$ on instances with N actions and T rounds. Then $\mathcal{A}'$ incurs polytope swap regret at most $R(|\mathcal{V}(\mathcal{P})|,T)$ on $\mathcal{P}$-learning instances over T rounds.*

Proof Let $\beta^t \in \Delta(\mathcal{V}(\mathcal{P}))$ be the mixed strategy output by $\mathcal{A}$ in round t, and let $x^t = \sum_{v\in\mathcal{V}(\mathcal{P})} \beta_v^t v$ be the point in $\mathcal{P}$ played by $\mathcal{A}'$ in round t. Note that β^t is a vertex partition of x^t. If we use these vertex partitions in the definition of polytope swap regret, we find that

$$\mathsf{PolySwapReg} \le \left(\max_{\pi:\mathcal{V}(\mathcal{P})\to\mathcal{V}(\mathcal{P})} \sum_{t=1}^{T} \left\langle r^t, \overline{\pi(\beta^t)} \right\rangle \right) - \sum_{t=1}^{T} \langle r^t, x^t \rangle.$$

If we rewrite the RHS of the above equation (decomposing by vertices in the same manner as in the proof of Theorem 6), we have that

$$\mathsf{PolySwapReg} \le \max_{\pi:\mathcal{V}(\mathcal{P})\to\mathcal{V}(\mathcal{P})} \sum_{v\in\mathcal{V}(\mathcal{P})} \sum_{t=1}^{T} \beta_v^t \left\langle r^t, \pi(v) - v \right\rangle.$$

But the right hand side of the above equation is exactly the value of $\mathsf{SwapReg}$ faced by $\mathcal{A}$. This regret in turn is at most $R(|\mathcal{V}(\mathcal{P})|,T)$ by the guarantees of $\mathcal{A}$. ■

D.2. Efficient learning algorithms for Bayesian games

The reduction in the previous section produces an algorithm incurs at most $O(\sqrt{TV\log V})$ polytope swap regret and runs in time $O(\mathrm{poly}(V))$ per round, where $V = |\mathcal{V}(\mathcal{P})|$ is the number of vertices of the polytope $\mathcal{P}$. While for some polytopes (e.g. the simplex $\Delta([N])$) these bounds are reasonable, some common polytopes have an exponentially large number of vertices (even those with a compact representation as the intersection of a small number of half-spaces). Most relevant to us, the Bayesian game polytope $\mathcal{P} = \Delta([N])^C$ has $V = N^C$ vertices, which is exponential in C. This dependence makes this algorithm unusable in settings with even a moderate number of contexts.

In this section we present two efficient contextual learning algorithms: one that simply runs a low-swap-regret algorithm independently for each context, and a more complex algorithm which incorporates the idea of swapping "between" different contexts. While we do not show these algorithms are low-polytope regret (the first one provably is not), we do show that these algorithms are somewhat robust to strategic behavior by the optimizer, in that we can still upper bound the maximum reward of optimizer as a function of the game G.

D.2.1. RUNNING A LOW-SWAP-REGRET ALGORITHM PER CONTEXT

We begin by analyzing the simple algorithm that runs an independent low-swap-regret algorithm for each context (Algorithm 1).

Since running a low external-regret algorithm per context results in a contextual learning algorithm with low external-regret, one may naturally suspect that running a low internal-regret algorithm per context should result in a contextual learning algorithm with low (polytope) swap-regret. Interestingly, this is not the case.

Algorithm 1: Running a low-swap-regret algorithm per context.

Learner initializes C copies $(\mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_C)$ of a low-swap-regret algorithm over N arms and T rounds.
for $t \leftarrow 1$ **to** T **do**
  for $c \leftarrow 1$ **to** C **do**
    Learner receives mixed strategy $\beta_c^t \in \Delta([N])$ from algorithm $\mathcal{A}_c$.
  end
  Learner plays $\beta^t(c) : [C] \to \Delta([N])$ given by $\beta^t(c) = \beta_c^t$.
  Optimizer plays mixed strategy $\alpha^t \in \Delta([M])$.
  for $c \leftarrow 1$ **to** C **do**
    Learner updates algorithm $\mathcal{A}_c$ with the rewards $u_L(\alpha^t, j, c)$ for $j \in [N]$.
  end
end

Theorem 11 *There exist Bayesian games G where if the optimizer plays G for T rounds against a learner running Algorithm 1 (for some choice of swap-regret algorithm), the optimizer can get at least* $\mathsf{Val}(G)T + \Omega(T)$ *reward.*

Although Theorem 11 shows that it is possible for an optimizer to gain significantly (over their Stackelberg value) by strategizing, we can upper bound the extent to which this occurs in terms of a different Stackelberg value-like benchmark.

Given a Bayesian game G, let $G_1, G_2, \ldots, G_C$ be the standard games induced by the C different contexts. We define the per-context Stackelberg value $\mathsf{PerConVal}(G)$ as the weighted average of the Stackelberg values of the games G_c:

$$\mathsf{PerConVal}(G) = \sum_{c=1}^{C} p_c \mathsf{Val}(G_c).$$

Alternatively, one can interpret $\mathsf{PerConVal}(G)$ as the maximum amount the optimizer can receive by playing a fixed strategy, if the optimizer's strategy too is allowed to depend on contexts. We show that $\mathsf{PerConVal}(G)$ upper bounds the optimizer's per-round reward when playing against Algorithm 1.

Theorem 12 *Let G be a Bayesian game. If an optimizer plays G for T rounds against a learner running Algorithm 1, the optimizer will receive reward at most* $\mathsf{PerConVal}(G)T + o(T)$.

Proof Since each sub-algorithm $\mathcal{A}_c$ of Algorithm 1 is low-swap regret, we know (via the results of Deng et al. (2019)) or Theorem 6 applied to standard games) that for each context c we must have that:

$$\sum_{t=1}^{T} u_O(\alpha^t, \beta^t(c), c) \leq \mathsf{Val}(G_c)T + o(T).$$

Summing this over c (weighting by p_c) we have that

$$\sum_{t=1}^{T} u_O(\alpha^t, \beta^t) \leq \mathsf{PerConVal}(G)T + o(T).$$

■

D.2.2. A SWAP-REGRET REDUCTION FOR CONTEXTUAL LEARNING

We now present a second algorithm (Algorithm 2) that avoids some of the pitfalls of Algorithm 1 (notably, it works on the example of Theorem 11 and achieves a stronger analogue of Theorem 12). Unlike Algorithm 1, which employs a low-swap-regret algorithm as a blackbox, in Algorithm 2 we do something more akin to the reduction of Blum and Mansour (2007) by running several external regret algorithms in parallel and computing a "steady state" distribution.

However, in our setting this notion of "steady state" is significantly more involved than in the classical external to internal regret reduction. Whereas the steady state of the reduction of Blum and Mansour (2007) can be expressed as the stationary distribution of a Markov chain, our "steady state" distribution is the fixed point of a system of degree 2 polynomials. This poses several challenges, both for showing this steady state exists and for computing it (indeed, we do not currently have a polynomial-time algorithm for finding this steady state, but we do have several heuristic approaches that appear to work well unlike in the case of the inefficient algorithm of Section D.1).

Algorithm 2: Swapping between contexts.

For each $c \in [C]$ and $j \in [N]$, the learner initializes an instance $\mathcal{A}_{c,j}$ of a low external-regret algorithm over $N + C$ arms and T rounds.

for $t \leftarrow 1$ **to** T **do**

  for $c \leftarrow 1$ **to** C **do**

    Learner receives a distribution $\gamma_{c,j}^t \in \Delta([N + C])$ from algorithm $\mathcal{A}_{c,j}$.

  end

  Learner plays a $\beta^t(c) : [C] \to \Delta([N])$ satisfying

$$\beta^t(c)_j = \sum_{j'=1}^{N} \beta^t(c)_{j'} \left(\gamma_{c,j',j}^t + \sum_{c'=1}^{C} \gamma_{c,j',N+c'}^t \beta^t(c')_j \right). \tag{3}$$

  Optimizer plays mixed strategy $\alpha^t \in \Delta([M])$.

  for $c \leftarrow 1$ **to** C **do**

    Learner updates algorithm $\mathcal{A}_{c,j}$ with the reward $r_{c,j}^t \in [-1, 1]^{N+C}$, where for $1 \leq j' \leq N$,

$$r_{c,j,j'}^t = \beta^t(c)_j \cdot u_L(\alpha, j', c),$$

    and where for $1 \leq c' \leq C$,

$$r_{c,j,N+c'}^t = \beta^t(c)_j \cdot u_L(\alpha, \beta^t(c'), c).$$

  end

end

We attempt to give some motivation behind the workings of Algorithm 2 and where this polynomial system arises from. Intuitively, what we would like to do is run C copies of a low-swap-regret algorithm, one for responsible for learning a good distribution over actions for each context. Each copy runs over $N + C$ arms. N of these arms correspond to pure actions for the learner. Where this gets tricky is the other C actions correspond to other contexts – specifically, they correspond to the distributions over arms output by the *original C low-swap-regret algorithms*.

Such an algorithm has the property that we do not only get low-swap-regret between actions in the same context (as with Algorithm 1), but also low-swap-regret "between" different contexts (in particular, this will allow us to avoid the bad example in Theorem 11). The problem is that this algorithm as described is inherently circular; the output of these low regret algorithms are also arms. We resolve this by constructing an appropriate fixed point problem.

Before we analyze the guarantees of our algorithm, we show that it is well-defined. In particular, we show that we can always find a solution to the system of equations (3) defining $\beta^t(c)$ in Algorithm 2.

Lemma 13 *For any choice of $\gamma_{c,j} \in \Delta([N + C])$, there always exists a $\beta : [C] \to \Delta([N])$ that satisfies*

$$\beta(c)_j = \sum_{j'=1}^{N} \beta(c)_{j'} \left(\gamma_{c,j',j} + \sum_{c'=1}^{C} \gamma_{c,j',N+c'} \beta(c')_j \right).$$

Lemma 13 shows that a solution β^t always exists to the above system of polynomial equations, but does not describe how to find it efficiently. Indeed, this is a problem – unlike in the linear case, solving systems of quadratic equations over multiple variables can be NP-hard.

Nonetheless, specific quadratic programs and systems can be solved efficiently, and we conjecture that it is possible to approximately solve this system in polynomial time. Specifically, taking inspiration from the power method for computing the stationary distribution for Markov chains, we suspect that starting from a uniform β and iterating the map

$$\beta(c)_j \to \sum_{j'=1}^{N} \beta(c)_{j'} \left(\gamma_{c,j',j} + \sum_{c'=1}^{C} \gamma_{c,j',N+c'} \beta(c')_j \right) \tag{4}$$

quickly converges to the fixed point shown to exist in Lemma 13 (this is supported by some computer simulations). We pose this as an open question.

Question 2 *Let $\beta^{(\tau)}$ be the element of $\Delta([N])^C$ obtained by starting with the $\beta^{(0)}$ that maps each $c \in [C]$ to the uniform distribution over $[N]$ and iterating the map (4) τ times. Then if β is the fixed point in Lemma 13, is it the case that*

$$\|\beta - \beta^{\tau}\|_\infty \leq \varepsilon$$

for some value of $\tau = \text{poly}(N, C, \log 1/\varepsilon)$?

As with Algorithm 1, we can upper bound the extent to which an optimizer can gain by strategizing against Algorithm 2. To do so, we first need to define a version of correlated equilibria in Bayesian games.

A *correlated equilibrium* of a Bayesian game G is specified by a set of C distributions $\mathcal{F}_c$ over $[M] \times [N]$, where the strategy profile $(i, j) \in [M] \times [N]$ occurs with probability $p_{i,j}(c)$ in $\mathcal{F}_c$. Now, imagine a variant of the game G (analogous to how correlated equilibria in standard games are defined) where the learner begins by communicating the context c to a third-party "correlator". The correlator then draws a strategy profile (i, j) from $\mathcal{F}_c$ and tells the optimizer to play i and the learner to play j. In order for this set of distributions to be a correlated equilibrium, they must satisfy the following properties[6]:

- The learner must have no incentive to misreport their type. That is, for all $c, c' \in [C]$, we must have:

$$\mathop{\mathbb{E}}_{(i,j)\sim\mathcal{F}_c}[u_L(i, j, c)] \geq \mathop{\mathbb{E}}_{(i,j)\sim\mathcal{F}_{c'}}[u_L(i, j, c)].$$

- The learner must have no incentive to "swap" their action. That is, for every swap rule $\pi : [N] \to [N]$, we must have:

$$\mathop{\mathbb{E}}_{(i,j)\sim\mathcal{F}_c}[u_L(i, j, c)] \geq \mathop{\mathbb{E}}_{(i,j)\sim\mathcal{F}_c}[u_L(i, \pi(j), c)].$$

We define the *correlated Stackelberg value* $\mathsf{CorrVal}(G)$ of G to be the maximum expected value for the optimizer over all correlated equilibria of G; i.e.,

$$\mathsf{CorrVal}(G) = \max \mathop{\mathbb{E}}_{c\sim\mathcal{D}} \mathop{\mathbb{E}}_{(i,j)\sim\mathcal{F}_c}[u_O(i, j, c)],$$

where the maximum is over all collections $\{\mathcal{F}_c\}$ are correlated equilibria of G. It turns out that for all Bayesian games G, $\mathsf{CorrVal}(G) \leq \mathsf{PerConVal}(G)$; in particular, this follows from the known fact that the Stackelberg value of a standard game is equal to the maximum utility of the Stackelberg player in a correlated equilibrium of the game (see Conitzer (2016) or Von Stengel and Zamir (2010) for a proof). In particular, if we remove the constraint that the learner has no incentive to misreport their type from the above criteria, we recover an alternate definition for $\mathsf{PerConVal}(G)$.

We now show that $\mathsf{CorrVal}(G)$ upper bounds the optimizer's per-round reward when playing against Algorithm 2 (thus providing a stronger bound than Theorem 12).

Theorem 14 *Let G be a Bayesian game. If an optimizer plays G for T rounds against a learner running Algorithm 2, the optimizer will receive reward at most $\mathsf{CorrVal}(G)T + o(T)$.*

Unlike Algorithm 1, we have no example of a Bayesian game G where the optimizer can do better than $\mathsf{Val}(G)T$ when playing against Algorithm 2. It is an interesting open question whether Algorithm 2 is low polytope swap regret (or otherwise prevents the optimizer from beating their Stackelberg value).

Question 3 *Is there a game G where the optimizer can get $\mathsf{Val}(G)T + \Omega(T)$ reward when playing against a learner running Algorithm 2? Is Algorithm 2 a low polytope regret contextual learning algorithm?*

6. Note that we only include constraints for the learner, since we later choose the equilibrium which is most favorable to the optimizer. This one-sidedness causes this definition to be slightly different than the traditional definition for correlated equilibria.

We suspect the answer to the above answer is no: in particular, we suspect that the exponential regret and complexity of the generic algorithm in Section D.1 is necessary, and that there is no contextual learning algorithm that achieves polytope swap regret $O(\mathrm{poly}(C,K)\sqrt{T})$ and no efficient (running in $\mathrm{poly}(C,K)$ time per iteration) contextual learning algorithm that achieves polytope swap regret $o(T)$.

D.3. Computing Bayesian Stackelberg equilibria is hard

One reason to suspect that minimizing polytope swap regret is hard for a contextual learner is that, surprisingly, the optimizer faces a provably hard optimization problem when playing against a low polytope swap regret learner in a Bayesian game G. Specifically, we show (via closely following a proof of a similar result for typed principal-agent problems in Guruganesh et al. (2021)) that it is APX-hard to compute the Stackelberg value $\mathsf{Val}(G)$ for a Bayesian game (and thus hard to compute the corresponding Stackelberg strategy).

Theorem 15 *It is APX-hard to compute the Stackelberg value for the optimizer in a Bayesian game. That is, there exists a constant $\varepsilon > 0$ such that given a Bayesian game G and a value $\mathsf{Val}' > 0$ it is NP-hard to distinguish between the cases $\mathsf{Val}(G) \le (1-\varepsilon)\mathsf{Val}'$ and $\mathsf{Val}(G) \ge \mathsf{Val}'$.*

Appendix E. Achieving the Stackelberg value in Bayesian games and polytope games

We show (as mentioned in Section 2.3) that an optimizer can always achieve the Stackelberg value in Bayesian game G, under mild conditions on G (essentially, every pure strategy for the learner is a strict best response for some strategy for the optimizer). In fact, we will show this for an arbitrary polytope game G, from which the conclusion for Bayesian games will immediately follow. Our proof largely follows the analogous proof in Deng et al. (2019) for standard games.

Let $\mathcal{P}$ be a polytope and let G be a $\mathcal{P}$-game. We say the game G is non-degenerate if, for each vertex v, there exists a strategy for the optimizer α_v where the learner's strict best response is v (i.e., $\mathsf{BR}(\alpha_v) = \{v\}$).

Lemma 16 *Let G be a non-degenerate game. Then if the optimizer plays T rounds of G against a learner running a low (external) regret algorithm $\mathcal{A}$, the optimizer can guarantee a reward of at least $\mathsf{Val}(G)T - o(T)$.*

Proof Let $\alpha = (r,s) \in \mathcal{Q}$ be the optimizer's strategy and let $v \in \mathcal{V}(\mathcal{P})$ be the learner's best-response in the Stackelberg equilibrium of G. Because G is non-degenerate, there exists a strategy $\alpha_v = (r_v, s_v) \in \mathcal{Q}$ for the optimizer where v is the strict best response. Let us define $\delta = \min_{v' \neq v} \langle r_v, v\rangle - \langle r_v, v'\rangle$; intuitively, δ represents the margin by which v is a strict best-response.

Assume the algorithm $\mathcal{A}$ has the guarantee that it incurs at most $R(T) = o(T)$ external regret on any $\mathcal{P}$-learning instance for T rounds. Set $\varepsilon = \sqrt{R(T)/T}$, and consider what happens when the optimizer plays $\alpha' = (1-\varepsilon)\alpha + \varepsilon\alpha_v$ every round for T rounds against $\mathcal{A}$.

Let $x^t \in \mathcal{P}$ (for $1 \le t \le T$) be the response of the learner in round t. Let $\rho^t \in \Delta(\mathcal{V}(\mathcal{P}))$ be an arbitrary vertex decomposition of x^t. Note that the total utility of the learner can be written as

$$\sum_{v' \in \mathcal{V}(\mathcal{P})} \sum_{t=1}^{T} \rho^t_{v'} \langle r', v'\rangle.$$

Since the learner has low external regret, we must have that

$$\sum_{v' \in \mathcal{V}(\mathcal{P})} \sum_{t=1}^{T} \rho_{v'}^t \langle r', v' - v \rangle \leq R(T).$$

Now, note that if $v' \neq v$, our guarantee on α_v implies that $\langle r', v' - v \rangle \geq \varepsilon\delta$. It therefore follows that

$$\sum_{v' \neq v} \sum_{t=1}^{T} \rho_{v'}^t = \sum_{t=1}^{T} (1 - \rho_v^t) \leq \frac{R(T)}{\varepsilon\delta} = \frac{\sqrt{R(T)T}}{\delta}.$$

On the other hand, the total utility of the optimizer can be written as

$$\sum_{v' \in \mathcal{V}(\mathcal{P})} \sum_{t=1}^{T} \rho_{v'}^t \langle s', v' \rangle \geq \sum_{t=1}^{T} \rho_v^t \langle s', v \rangle.$$

Since α and v form a Stackelberg equilibrium for G, $\langle s', v \rangle \geq (1 - \varepsilon)\langle s, v \rangle = (1 - \varepsilon)\mathsf{Val}(G)$. It follows that the optimizer's utility is at least

$$\sum_{t=1}^{T} \rho_v^t \langle s', v \rangle \geq \left(\sum_{t=1}^{T} \rho_v^t \right) (1 - \varepsilon)\mathsf{Val}(G) \geq T \left(1 - \frac{\varepsilon}{\delta} \right) (1 - \varepsilon)\mathsf{Val}(G) \geq \mathsf{Val}(G)T - o(T).$$

■

Appendix F. Omitted proofs

F.1. Proof of Theorem 3

Proof [Proof of Theorem 3] Since $\mathcal{A}$ is not a low-swap-regret algorithm, there exists some $\gamma > 0$ and positive integer N such that for infinitely many values of T, there exists an online learning instance with N actions and T rounds where $\mathcal{A}$ incurs at least γT swap regret.

If we let the game G depend on T, then this theorem directly follows from Lemma 1. But also note that for a fixed value of N, the ultimate game G constructed in Lemma 1 depends solely on the optimal swap function π. Since there are infinitely many values of T for which there exists a bad online-learning instance but only finitely many (N^N) different swap functions, infinitely many of these values of T will have the same swap function π and hence the same game G. This game satisfies the theorem statement. ■

F.2. Proof of Lemma 4

Proof [Proof of Lemma 4] Let $r^t \in [-1, 1]^d$ be the rewards of the bad instance of the $\mathcal{P}$-learning problem, and let $x^t \in \mathcal{P}$ be the corresponding actions played by $\mathcal{A}$. Since the linear swap regret on this instance equals R, we know that there exists an $M \in \mathcal{M}(\mathcal{P})$ such that

$$\sum_{t=1}^{T} \langle r^t, Mx^t \rangle - \sum_{t=1}^{T} \langle r^t, x^t \rangle = R. \tag{5}$$

We rewrite this as

$$\sum_{t=1}^{T}\langle r^t, (M-I)x^t\rangle = R. \tag{6}$$

To define the $\mathcal{P}$-game G we simply need to specify the polytope $\mathcal{Q}$. We will define $\mathcal{Q}$ to be the polytope

$$\mathcal{Q} = \left\{\left(y, \frac{1}{\lambda+1}(M-I)^\top y\right)\middle| y \in [-1,1]^d\right\}.$$

Note that scaling by $1/(\lambda+1)$ guarantees that $\mathcal{Q} \subseteq [-1,1]^{2d}$. In particular, since $\|M\|_1 \le \lambda$, $\|M^\top\|_\infty \le \lambda$, so $\|(M-I)^\top\|_\infty \le \lambda+1$, and therefore $(M-I)^\top y \in [-(\lambda+1),(\lambda+1)]^d$.

Now, consider an optimizer who in round t plays the action $q^t = (r^t, (M-I)^\top r^t)$. The learner, running $\mathcal{A}$, will see exactly the sequence of rewards r^t and therefore on each round t play action $x^t \in \mathcal{P}$. The total reward of the optimizer is then given by

$$\sum_{t=1}^{T}\langle (M-I)^\top r^t, x^t\rangle = \sum_{t=1}^{T}\langle r^t, (M-I)x^t\rangle = R.$$

On the other hand, we claim the Stackelberg value $\mathsf{Val}(G)$ of the game is at most zero. To see this, note that if the optimizer plays $q = (r, (M-I)^\top r) \in \mathcal{Q}$ and the learner best responds by playing $x \in \mathcal{P}$, then the optimizer's payoff is $\langle (M-I)^\top r, x\rangle = \langle r, (M-I)x\rangle = \langle r, Mx\rangle - \langle r, x\rangle$. But since $Mx \in \mathcal{P}$ and x is the learner's best response to q, we must have that $\langle r, Mx\rangle \le \langle r, x\rangle$. It follows that $\mathsf{Val}(G) \le 0$. ■

F.3. Proof of Theorem 5

Proof [Proof of Theorem 5] If $\mathcal{A}$ does not have low linear swap regret, then there exists a $\gamma > 0$ such that for infinitely many values of T, there exists a $\mathcal{P}$-learning instance where $\mathcal{A}$ incurs at least γT linear swap regret.

As with standard games, if the game G could depend on T, then we would be done by Lemma 4. We argue that for a fixed polytope $\mathcal{P}$, the procedure in Lemma 4 can generate only finitely many different games. In particular, note that for a fixed problem instance, the game G is defined entirely by a matrix M in $\mathcal{M}(\mathcal{P})$. Now, while there may be infinitely many matrices in $\mathcal{M}(\mathcal{P})$, we will show that $\mathcal{M}(\mathcal{P})$ is actually a convex polytope in $\mathbb{R}^{d^2}$, and we can always choose M to be a vertex of $\mathcal{M}(\mathcal{P})$ (of which there are finitely many).

To see that $\mathcal{M}(\mathcal{P})$ is a polytope, note that in order to guarantee the constraint that $x \in \mathcal{P}$ implies $Mx \in \mathcal{P}$ it is necessary and sufficient that M map each vertex of $\mathcal{P}$ to a point within $\mathcal{P}$. Therefore for each vertex v (of the finitely many vertices) of $\mathcal{P}$ and each halfspace $Ax \le b$ (of the finitely many halfspaces) defining $\mathcal{P}$, we obtain the linear constraint $AMv \le b$ on the matrix M. These linear constraints define $\mathcal{M}(\mathcal{P})$.

Now, note that for any fixed rewards r^t and actions x^t, the sum $\sum_{t=1}^{T}\langle r^t, Mx^t\rangle$ is linear in the matrix M. This means that over all matrices $M \in \mathcal{M}(\mathcal{P})$, it is maximized at one of the vertices of $\mathcal{M}(\mathcal{P})$. In particular, in Lemma 4, we can always ensure that the M we choose to define G is one of these finitely many vertices (and therefore there are only finitely many possible $\mathcal{P}$-games we can

generate). One of these games G must occur for infinitely many values of T, and that game satisfies the theorem statement. ■

F.4. Proof of Theorem 6

Proof [Proof of Theorem 6] Consider the transcript of this game when played for T rounds. Let $q^t = (r^t, s^t) \in \mathcal{Q}$ be the optimizer's action in round t, and let $x^t \in \mathcal{P}$ be the learner's action in round t. Since $\mathcal{A}$ is low swap regret, we know there exist vertex decompositions ρ^t of x^t such that for any swap function $\pi : \mathcal{V}(\mathcal{P}) \to \mathcal{V}(\mathcal{P})$,

$$\sum_{t=1}^{T} \left\langle r^t, \overline{\pi(\rho^t)} \right\rangle - \sum_{t=1}^{T} \langle r^t, x^t \rangle = o(T). \tag{7}$$

Using the fact that $x^t = \overline{\rho^t}$, we can rewrite (7) as

$$\sum_{t=1}^{T} \left\langle r^t, \overline{\pi(\rho^t)} - \overline{\rho^t} \right\rangle = o(T). \tag{8}$$

Decomposing (8) over vertices in $\mathcal{V}(\mathcal{P})$, this becomes

$$\sum_{v \in \mathcal{V}(\mathcal{P})} \sum_{t=1}^{T} \rho_v^t \left\langle r^t, \pi(v) - v \right\rangle = o(T). \tag{9}$$

Now, let $\sigma_v = \sum_{t=1}^{T} \rho_v^t$, let $\tilde{r}_v = (\sum_{t=1}^{T} \rho_v^t r^t)/\sigma_v$, and let $\tilde{s}_v = (\sum_{t=1}^{T} \rho_v^t s^t)/\sigma_v$. Note that $(\tilde{r}_v, \tilde{s}_v)$ is a convex combination of the optimizer's actions (r^t, s^t) and therefore belongs to $\mathcal{Q}$.

We can once again rewrite (9) as

$$\sum_{v \in \mathcal{V}(\mathcal{P})} \sigma_v \langle \tilde{r}_v, \pi(v) - v \rangle = o(T). \tag{10}$$

Now, note that we can write the optimizer's utility in the form $\sum_{v \in \mathcal{V}(\mathcal{P})} \sigma_v \langle \tilde{s}_v, v \rangle$. Assume that the statement of the theorem is not true, namely that for infinitely many T, we have that

$$\sum_{v \in \mathcal{V}(\mathcal{P})} \sigma_v \langle \tilde{s}_v, v \rangle \geq (\mathsf{Val}(G) + \gamma) T \tag{11}$$

for some $\gamma > 0$. Let $\mathsf{BR}(\tilde{s}_v) \in \mathcal{V}(\mathcal{P})$ be the learner's best response to the optimizer's action $(\tilde{r}_v, \tilde{s}_v)$ in G. By the definition of the Stackelberg value of G, we have that

$$\sum_{v \in \mathcal{V}(\mathcal{P})} \sigma_v \langle \tilde{s}_v, \mathsf{BR}(\tilde{s}_v) \rangle \leq \mathsf{Val}(G) T \tag{12}$$

Subtracting (12) from (11), we obtain

$$\sum_{v \in \mathcal{V}(\mathcal{P})} \sigma_v \langle \tilde{s}_v, v - \mathsf{BR}(\tilde{s}_v) \rangle \geq \gamma T \tag{13}$$

But this contradicts (10) for the swap function $\pi(v) = \mathsf{BR}(\tilde{s}_v)$. The theorem follows. ■

F.5. Proof of Theorem 8

Proof [Proof of Theorem 8] Note that when $K = C = 2$, there are four vertices in $\mathcal{V}(\mathcal{P})$: $v_{11} = (1, 0; 1, 0)$, $v_{12} = (1, 0; 0, 1)$, $v_{21} = (0, 1; 1, 0)$ and $v_{22} = (0, 1; 0, 1)$. The learner will play v_{11} for the first $T/4$ rounds, v_{12} for the second $T/4$ rounds, v_{21} for the third $T/4$ rounds, and v_{22} for the last $T/4$ rounds. Simultaneously, the rewards will be $r_{11} = v_{11}$ for the first $T/4$ rounds, $r_{12} = v_{12}$ for the next $T/4$ rounds, $r_{21} = v_{21}$ for the next $T/4$ rounds, and finally, $r_{22} = v_{11}$ for the last $T/4$ rounds. **Note that for the last $T/4$ rounds, $r_{22} \neq v_{22}$, but instead $r_{22} = v_{11}$.**

Let us begin by considering the polytope swap regret of this instance. Note that each action of the learner is already a vertex belonging to $V(\mathcal{P})$. This means that we have no freedom in choosing the vertex partition; each vertex partition must put all of its weight on the action itself. Then, the swap function which maps $(v_{11}, v_{12}, v_{21}, v_{22}) \to (v_{11}, v_{12}, v_{21}, v_{11})$ increases the learner's utility by 2 points per round for the last $T/4$ rounds, and therefore $\mathsf{PolySwapReg} \geq T/2$.

On the other hand, there is no linear function which maps vertices v_{11}, v_{12}, v_{21} to themselves but which maps the vertex v_{22} to the vertex v_{11}. To upper bound the linear swap regret, we can find the best linear swap function by checking each of the extremal points of $\mathcal{M}(\mathcal{P})$. It turns out there are 64 such extremal swap functions, and a straightforward computation shows that none perform better than the identity function. It follows that $\mathsf{LinSwapReg} = 0$. ■

F.6. Proof of Theorem 9

Proof [Proof of Theorem 9] We extend the example separating linear swap regret and polytope swap regret in Theorem 8. We generalize this to a game G where the optimizer has four actions, each corresponding to one of the four quarters of the game (so the optimizer plays action 1 for the first $T/4$ rounds, action 2 for the next $T/4$ rounds, etc.). For each action i, the optimizer has a reward vector $s_i \in \mathbb{R}^4$, denoting that if the learner plays a mixed action $\alpha \in \mathcal{P}$, then the optimizer gets reward $\langle \alpha, s_i \rangle$. (In the language of polytope games, the action set for the optimizer is $\mathcal{Q} = \mathrm{conv}(\{(v_i, s_i)\}_{i=1}^4)$. If we set:

$$\begin{aligned} s_1 &= (0.00, 0.26; 0.60, 0.21) \\ s_2 &= (0.05, 0.17; 0.45, 0.68) \\ s_3 &= (0.16, 0.25; 0.33, 0.20) \\ s_4 &= (0.16, 0.68; 0.22, 0.44), \end{aligned}$$

then we can check that while $\mathsf{Val}(G) = 0.74$, the optimizer gets $0.7575T$ reward from playing the mentioned trajectory of actions (action 1, then 2, then 3, then 4). ■

F.7. Proof of Theorem 11

Proof [Proof of Theorem 11] We adapt an example of Braverman et al. (2018). We describe a Bayesian game G with $M = N = C = 2$. We will interpret this game as a game where the optimizer is trying to sell an item to a learner whose value is specified by the context c. The optimizer can set one of two prices for the item (0 or 1), the learner's value for the item is either $1/4$ or $1/2$ (depending

on the value of C), and the learner must choose whether to buy or not buy the item (without seeing the price). Formally, we have

$$\begin{aligned} u_L(i,j,c) &= \left(\frac{c}{4} - i\right) j \\ u_O(i,1,c) &= ij, \end{aligned}$$

where $i \in \{0,1\}, j \in \{0,1\}, c \in [2]$, and let $\mathcal{D}$ be the uniform distribution over the two contexts. It is straightforward to verify that $\mathsf{Val}(G) = 1/4$, and this is achievable if the optimizer plays $\alpha = (3/4, 1/4)$ (i.e., the optimizer sets a price of $1/4$ for the item).

We now show that an optimizer can get $T/4 + \Omega(T)$ when playing against a learner running Algorithm 1 for T steps. As in the description of the algorithm, let $\beta^t : [C] \to \Delta([N])$ be the strategy the learner plays in round t.

Note that since $N = 2$, low-swap-regret over N actions is equivalent to low-regret over N actions, so we can just assume that both of the sub-algorithms $\mathcal{A}$ are a low-regret algorithm such as Hedge. The only property of the learning algorithm that we will need is the following: assume $R^t(c)_0$ is the total reward of action 0 up until round t in context c and that $R^t(c)_1$ is the total reward of action 1 up until round t in context c. Then there is a sublinear function $r(T) = o(T)$ such that if $R^t(c)_1 - R^t(c)_0 > r(t)$, then $\beta^t(c)_0 < o(T)$.

Now consider the following strategy for the optimizer: the optimizer will play 0 for the first $T/2$ rounds and 1 for the last $T/2$ rounds. Let us consider the values of $R^t(c)_0$ and $R^t(c)_1$. When $j = 0$, the learner does not buy the item and $u_L = u_O = 0$, so $R^t(c)_0 = 0$. On the other hand $R^t(c)_1 = ct/4$ for $t \in [1, T/2]$ and $R^t(c)_1 = ct/4 - (t - T/2)$ for $t \in [1, T/4]$. Note that when $c = 2$, $R^t(c) \geq 0$ for all t (and in fact $R^t(c) \geq o(T)$ for almost all t), so $\beta^t(c)_1 \geq 1 - o(T)$ for almost all t. It follows that the optimizer receives utility $(1/2) \cdot (T/2) \cdot 1 - o(T) = T/4 - o(T)$ from the context $c = 2$.

On the other hand, when $c = 1$, $R^t(c) \geq 0$ for all $1 \leq t \leq 3T/4$. A similar argument shows that in this case, the optimizer receives utility $(1/2) \cdot (T/4) = T/8 - o(T)$ from this context. Overall, the optimizer receives utility $3T/8 - o(T) = T/4 + \Omega(T)$. ■

F.8. Proof of Lemma 13

Proof [Proof of Lemma 13] Fix a $\beta : [C] \to \Delta([N])$, and consider the function $\beta' : [C] \to \mathbb{R}^n$ defined via:

$$\beta'(c)_j = \sum_{j'=1}^{N} \beta(c)_{j'} \left(\gamma_{c,j',j} + \sum_{c'=1}^{C} \gamma_{c,j,N+c'} \beta(c')_j \right).$$

We will show that $\beta'(c)$ is also an element of $[C] \to \Delta([N])$. Note that since both $\beta(c)$ and $\gamma_{c,j}$ are distributions, from the above equation we can observe that $\beta'(c)_j \in [0,1]$ for all j. It thus suffices to check that $\sum_j \beta'(c)_j = 1$ for each $c \in [C]$. We perform this computation:

$$\begin{aligned}
\sum_{j=1}^{N} \beta'(c)_j &= \sum_{j=1}^{N}\sum_{j'=1}^{N} \beta(c)_{j'} \left(\gamma_{c,j',j} + \sum_{c'=1}^{C} \gamma_{c,j',N+c'} \beta(c')_j \right) \\
&= \sum_{j=1}^{N}\sum_{j'=1}^{N} \beta(c)_{j'}\gamma_{c,j',j} + \sum_{j=1}^{N}\sum_{j'=1}^{N}\sum_{c'=1}^{C} \beta(c)_{j'}\gamma_{c,j',N+c'}\beta(c')_j \\
&= \sum_{j=1}^{N}\sum_{j'=1}^{N} \beta(c)_{j'}\gamma_{c,j',j} + \sum_{j'=1}^{N}\sum_{c'=1}^{C} \beta(c)_{j'}\gamma_{c,j',N+c'} \sum_{j=1}^{N} \beta(c')_j \\
&= \sum_{j=1}^{N}\sum_{j'=1}^{N} \beta(c)_{j'}\gamma_{c,j',j} + \sum_{j'=1}^{N}\sum_{c'=1}^{C} \beta(c)_{j'}\gamma_{c,j',N+c'} \\
&= \sum_{j'=1}^{N} \beta(c)_{j'} \sum_{k=1}^{N+C} \gamma_{c,j',k} \\
&= \sum_{j'=1}^{N} \beta(c)_{j'} = 1.
\end{aligned}$$

Now, since this mapping from β to β' is a continuous mapping from $\Delta([N])^C$ to $\Delta([N])^C$, by Brouwer's fixed point theorem, there must exist an element $\beta \in \Delta([N])^C$ which is fixed by this mapping (and therefore satisfies the equation in the theorem statement). ■

F.9. Proof of Theorem 14

Proof [Proof of Theorem 14] We will show that the average strategy profile of the optimizer and learner (over all T rounds) is approximately a correlated equilibrium of G, from which the conclusion will follow.

We begin by showing that in this average strategy profile, the learner cannot benefit much by "misreporting" their type, i.e.

$$\sum_{t=1}^{T} u_L(\alpha^t, \beta^t(c), c) \geq \sum_{t=1}^{T} u_L(\alpha^t, \beta^t(c'), c) - o(T). \tag{14}$$

To show this, note that the external regret guarantees of algorithm $\mathcal{A}_{c,j}$ ensure that for any $k \in [N + C]$,

$$\sum_{t=1}^{T} \langle r_{c,j}^t, \gamma_{c,j}^t \rangle \geq \sum_{t=1}^{T} r_{c,j,k}^t - o(T). \tag{15}$$

Let us pick $k = N + c'$. Then the above inequality becomes

$$\sum_{t=1}^{T} \langle r_{c,j}^t, \gamma_{c,j}^t \rangle \geq \sum_{t=1}^{T} \beta^t(c)_j \cdot u_L(\alpha^t, \beta^t(c'), c) - o(T). \tag{16}$$

On the other hand, note that

$$\langle r^t_{c,j}, \gamma^t_{c,j} \rangle = \beta^t(c)_j \left(\sum_{j'=1}^{N} \gamma_{c,j,j'} u_L(\alpha, j', c) + \sum_{c'=1}^{C} \gamma_{c,j,N+c'} u_L(\alpha, \beta^t(c'), c) \right) = \beta^t(c)_j u_L(\alpha, \beta^t(c), c), \tag{17}$$

where the last equality follows as a consequence of (3). Substituting this into (16), we have that:

$$\sum_{t=1}^{T} \beta^t(c)_j \cdot u_L(\alpha^t, \beta^t(c), c) \geq \sum_{t=1}^{T} \beta^t(c)_j \cdot u_L(\alpha^t, \beta^t(c'), c) - o(T) \tag{18}$$

Summing (18) over all $j \in [N]$ (and using the fact that $\sum_j \beta^t(c)_j = 1$), we obtain our desired inequality (14).

We now show that in the average strategy profile, the learner cannot benefit much by applying a swap function to their action. Specifically, let $\pi : [N] \to [N]$ be an arbitrary swap function. We will show that

$$\sum_{t=1}^{T} u_L(\alpha^t, \beta^t(c), c) \geq \sum_{t=1}^{T} u_L(\alpha^t, \pi(\beta^t(c)), c) - o(T). \tag{19}$$

Here for $\beta \in \Delta([N])$, we write $\pi(\beta)$ to denote the element of $\Delta([N])$ which satisfies $\pi(\beta)_j = \sum_{j' \in \pi^{-1}(j)} \beta_{j'}$ (i.e., we can sample from $\pi(\beta)$ by first sampling j' from β and then playing $\pi(j')$).

As before, we will start from the external regret guarantee (15) of the individual algorithm $\mathcal{A}_{c,j}$. If we fix $k = \pi(j)$ this time, and apply the same logic as before, we obtain the inequality

$$\sum_{t=1}^{T} \beta^t(c)_j \cdot u_L(\alpha^t, \beta^t(c), c) \geq \sum_{t=1}^{T} \beta^t(c)_j \cdot u_L(\alpha^t, \pi(j), c) - o(T). \tag{20}$$

But now, note that

$$\sum_{j=1}^{N} \beta^t(c)_j \cdot u_L(\alpha^t, \pi(j), c) = u_L(\alpha^t, \pi(\beta^t(c)), c). \tag{21}$$

It follows that by summing (20) over all $j \in [N]$ that we obtain our desired inequality.

Now, consider the set of C distributions $\mathcal{F}'_c$ over $[M] \times [N]$ given by

$$\mathop{\mathbb{P}}_{\mathcal{F}'_c}(i, j) = \frac{1}{T} \sum_{i=1}^{M} \sum_{j=1}^{N} \alpha^t_i \beta^t(c)_j.$$

In words, the collection of distributions $\mathcal{F}'_c$ record the average strategy profile played by the optimizer and learner over the T rounds. Now, (14) and (19) imply that for each T, there exists a function $\varepsilon(T) = o(1)$ such that $\mathcal{F}'_c$ form an ε-approximate correlated equilibrium of G (where an ε-approximate equilibrium satisfies the inequalities in the definition of a correlated equilibrium up to a slack of ε). Note that if we let $\mathsf{CorrVal}(G, \varepsilon)$ be the maximum value for the optimizer over all ε-correlated equilibria, then the optimizer will receive a reward of at most $\mathsf{CorrVal}(G, \varepsilon(T))T$ against this optimizer.

But now, note that $\mathsf{CorrVal}(G, \varepsilon)$ converges to $\mathsf{CorrVal}(G)$ as $\varepsilon \to 0$. In particular, this means that $\mathsf{CorrVal}(G, \varepsilon(T)) - \mathsf{CorrVal}(G) = o(1)$, and therefore the optimizer will receive a reward of at most $\mathsf{CorrVal}(G)T + o(T)$. ■

F.10. Proof of Theorem 15

Proof [Proof of Theorem 15] We adapt a proof of Guruganesh et al. (2021) which shows hardness of computing optimal contracts in principal agent problems with types ("adverse selection"). We will reduce to bounded-degree dominating-set, which is known to be an APX-hard problem Chlebík and Chlebíková (2008). Let H be a graph with V vertices, each with maximum degree at most 3. We will construct from H a Bayesian game G with $M = V + 1$ actions for the optimizer, $N = 5$ actions for the learner, and $C = 2N$ contexts.

For each vertex $1 \le v \le V$, let $\mathrm{nbr}(v, 0) = v$, and let $\mathrm{nbr}(v, 1)$, $\mathrm{nbr}(v, 2)$, and $\mathrm{nbr}(v, 3)$ be the three neighbors of v in H, arbitrarily ordered (if a vertex v has fewer than 3 neighbors, let $\mathrm{nbr}(v, j) = -1$ if $j > \deg(v)$). We will label the $M = V + 1$ actions for the optimizer $1, 2, 3, \ldots, V$ and $\varnothing$. We will label the $N = 5$ actions for the learner 0, 1, 2, 3, and $\overline{0}$. Finally, we will label the $C = 2V$ types $1, 2, \ldots, V$, and $\overline{1}, \overline{2}, \ldots, \overline{V}$.

We now describe the payoffs of G. We first restrict our attention to contexts $c = v$ for some $v \in [V]$. Intuitively, for these contexts the game works as follows. If the learner plays $\overline{0}$, then both the learner and optimizer receives nothing. Otherwise, if the learner plays action $j \in \{0, 1, 2, 3\}$ and the optimizer plays action i, the learner loses $1/(2V)$ but receives a payment of 1 from the optimizer if $i = \mathrm{nbr}(v, j)$. On the other hand, the optimizer gains $1/V$ if $\mathrm{nbr}(v, j)$ exists, but loses 1 to the learner if $i = \mathrm{nbr}(c, j)$. Formally, we have (for $v \in V$)

$$\begin{aligned} u_L(i, j, v) &= \mathbb{I}(i = \mathrm{nbr}(v, j)) - \frac{1}{2V} \\ u_O(i, j, v) &= \frac{1}{V}\mathbb{I}(\mathrm{nbr}(v, j) \neq -1) - \mathbb{I}(i = \mathrm{nbr}(v, j)) \\ u_L(i, \overline{0}, v) &= u_O(i, \overline{0}, v) = 0 \end{aligned}$$

Now let us consider a context $c = \overline{v}$ for some $v \in [V]$. In this case, if the learner plays any action $j \in \{0, 1, 2, 3\}$, the learner receives $-1/(2V)$ and the optimizer receives 0. On the other hand, if the learner plays $\overline{0}$ then they receive a payment of 1 from the optimizer if $i = v$, and the optimizer receives a payoff of $1/V$ but must pay the learner 1 if $i = v$. Formally:

$$\begin{aligned} u_L(i, \overline{0}, \overline{v}) &= \mathbb{I}(i = v) \\ u_O(i, \overline{0}, \overline{v}) &= \frac{1}{V} - \mathbb{I}(i = v) \\ u_L(i, j, \overline{v}) &= -\frac{1}{2V} \\ u_O(i, j, \overline{v}) &= 0 \end{aligned}$$

Finally, we assume the distribution $\mathcal{D}$ is uniform over contexts, i.e. $p_c = 1/2V$ for all contexts c.

Let us analyze G as a Bayesian Stackelberg game. First note that regardless of what mixed strategy α the optimizer plays, when the context is $\overline{v}$ the learner's best response is to play $\overline{0}$: this guarantees them a non-negative utility, whereas any other action they play gives them a utility of $-1/2V$. These $\overline{v}$ types cost the optimizer whenever the optimizer allocates weight to an action

$i \in [V]$ instead of to $\varnothing$. In particular, the optimizer loses a total of $(1 - \alpha_{\varnothing})$ utility from these types combined.

On the other hand, assume the type $c = v$ for some $v \in [V]$. Then if there exists an action j such that the optimizer places weight at least $1/(2V)$ on $\mathrm{nbr}(v, j)$ (i.e., $\alpha_{\mathrm{nbr}(v,j)} \geq 1/(2V)$), some such action will be the learner's best response: this action guarantees the learner a positive payoff, whereas the payoff for all other actions is at most zero. In this case the optimizer can get a payoff of up to $1/V - 1/(2V) = 1/(2V)$. On the other hand, if no such action exists, the learner should just play $\overline{0}$; this gives the learner zero utility, whereas all other actions will earn the learn negative utility. This case leads to the optimizer earning 0 utility.

From these observations, the first thing we can notice is that for each $i \in [V]$, the optimizer should allocate either weight 0 or $1/(2V)$ to action i. In particular, the optimizer never loses utility by decreasing the weight of an action from above $1/(2V)$ down to $1/(2V)$, or from less than $1/(2V)$ all the way down to 0. From now on, let's assume that $\alpha_i \in \{0, 1/(2V)\}$ for all $i \in [V]$.

Let S be the subset of $[V]$ containing the vertices i where $\alpha_i = 1/(2V)$. Let $\mathrm{nbr}(S)$ equal the set of vertices $i' \in [V]$ such that $\mathrm{nbr}(i', j) \in S$ for some j; in other words, $\mathrm{nbr}(S)$ is the set of vertices dominated by S. Then we claim that the value of the optimizer when playing this mixed strategy is equal to:

$$\frac{|\mathrm{nbr}(S)| - |S|}{4V^2}.$$

This is since: i) the optimizer gains utility $1/(2V)$ from each type $i \in \mathrm{nbr}(S)$, ii) the optimizer loses utility $1/(2V)$ from each type $\bar{i}$ with $i \in S$, and iii) each type has an equal probability of $1/(2V)$ of occurring.

Now, we claim that $\max_{S \subseteq [V]} |\mathrm{nbr}(S)| - |S| = V - D$, where D is the size of the minimal dominating set. To see this, note that while $\mathrm{nbr}(S) \neq [V]$, one can weakly monotonically increase the value of $|\mathrm{nbr}(S)| - |S|$ by adding an element of $[V] \setminus \mathrm{nbr}(S)$ to S (this increases $|S|$ by 1 and $|\mathrm{nbr}(S)|$ by at least 1. On the other hand, if $\mathrm{nbr}(S) = [V]$, then S is a dominating set so $\min |S| = D$ by definition. It follows that

$$\mathsf{Val}(G) = \frac{V - D}{4V^2}.$$

By the results of Chlebík and Chlebíková (2008) (Theorem 6), there exist graphs with degree at most 3 where it is NP-hard to distinguish decide between the cases $D \geq 0.2879V$ and $D \leq 0.2872V$. For the games generated from these graphs, it is likewise hard to distinguish whether $\mathsf{Val}(G) \leq 0.1781/V$ or whether $\mathsf{Val}(G) \geq 0.1782/V$. It is therefore NP-hard to compute $\mathsf{Val}(G)$ (or the associated Stackelberg strategy) to within a factor of $(1 - \varepsilon)$ for $\varepsilon = 1/1782$.

■