Εθνικο Μετσοβιο Πολυτεχνειο

Τμημα Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Εργαστηριο Λογικης και Επιστημης Υπολογιστων

Η Μέθοδος Πολλαπλασιαστικών Συντελεστών στο Σχεδιασμό Μηχανισμών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΛΥΔΙΑΣ ΖΑΚΥΝΘΙΝΟΥ

Επιβλέπων Καθηγητής:
Δημήτρης Φωτάκης
Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιανουάριος 2015

Εθνικο Μετσοβιο Πολυτεχνειο
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΛΟΓΙΚΗΣ ΚΑΙ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Η Μέθοδος Πολλαπλασιαστικών Συντελεστών στο Σχεδιασμό Μηχανισμών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΛΥΔΙΑΣ ΖΑΚΥΝΘΙΝΟΥ

Επιβλέπων: Δημήτρης Φωτάκης
Επίκουρος Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 7η Ιανουαρίου 2015.

.....	.....	.....
Δημήτρης Φωτάκης Επίκουρος Καθηγητής Ε.Μ.Π.	Ευστάθιος Ζάχος Καθηγητής Ε.Μ.Π.	Αριστείδης Παγουρτζής Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Ιανουάριος 2015.

.....
ΛΥΔΙΑ ΖΑΚΥΝΘΙΝΟΥ
Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Καταρχάς, ευχαριστώ θερμά τους καθηγητές του Εργαστηρίου Λογικής και Επιστήμης Υπολογιστών της σχολής μου, κ.Ζάχο, κ.Παγουρτζή και ιδιαίτερα τον κ.Φωτάκη γιατί μέσα από την αγάπη τους για τη διδασκαλία και την έρευνά τους, με ενθουσίασαν, έκαναν τη σχέση μου με το αντικείμενο προσωπική και κυρίως με βοήθησαν να συνεχίσω να διευρύνω τις γνώσεις μου και μετά από την πρώτη εκείνη επαφή. Ακόμα, ευχαριστώ τους τρεις τους αλλά και όλους τους καθηγητές τη σχολής μου, γιατί ο τρόπος που δομώ τις σκέψεις μου έχει αλλάξει προς το καλύτερο και αυτό οφείλεται σε μεγάλο βαθμό στη συναναστροφή μου με εκείνους, την ακαδημαϊκή αλλά και προσωπική.

Ευχαριστώ, επίσης, όλα τα μέλη του Εργαστηρίου, για τις ακαδημαϊκές τους συμβουλές αλλά και γιατί η 'οικογένεια' αυτή που έχουμε σχηματίσει είναι υπεύθυνη για το χαμόγελο που έχω όταν πηγαίνω στο Εργαστήριο.

Τέλος, ευχαριστώ τη φίλη μου Ειρήνη με την οποία νιώθω ότι μπορώ να καταφέρω τα πάντα (ίσως επειδή μαζί αντιμετωπίσαμε όλες τις εργασίες της σχολής...!) και τον Δημήτρη που μου θυμίζει να πιστεύω σε μένα. Ευχαριστώ τους παιδικούς μου φίλους που με τόση χάρη υπέμειναν τους μονολόγους μου περί θεωρητικής πληροφορικής και αν δεν είχαν επιλέξει να γίνουν νηπιαγωγοί και οικονομολόγοι δε θα αναγκαζόμουν να κατανοήσω σε τέτοιο βάθος τις νέες μου γνώσεις ώστε να τους τις μεταφέρω. Και πάνω από όλα, ευχαριστώ τους γονείς μου, όχι μόνο γιατί μου δίνουν την ευκαιρία και το χρόνο να ασχοληθώ με όσα αγαπώ και να γίνομαι καλύτερη σε αυτά, αλλά γιατί διαμόρφωσαν τη σκέψη μου πολύ πριν τη σχολή και με έκαναν να αγαπήσω τη μάθηση.

Περίληψη

Η Μέθοδος Πολλαπλασιαστικών Συντελεστών αποσκοπεί στην τυποποίηση μίας πολύ χρήσιμης ιδέας η οποία συναντάται σε μία ποικιλία ερευνητικών αντικειμένων και εφαρμόζεται πιο συχνά σε προβλήματα βελτιστοποίησης ή μηχανικής μάθησης. Οι αλγόριθμοι που λειτουργούν με βάση αυτή την ιδέα, διατηρούν μία κατανομή πιθανότητας (την οποία μπορούμε να φανταστούμε με τη μορφή βαρών) επάνω στα στοιχεία ενός συγκεκριμένου συνόλου, η οποία ανανεώνεται με ένα πολλαπλασιαστικό κανόνα. Παράλληλα, η ανάλυσή τους βασίζεται σε μία 'συνάρτηση δυναμικού', που αντικατοπτρίζει τις αλλαγές στα βάρη. Σε αυτή τη διπλωματική εργασία, θα παρουσιάσουμε το πλαίσιο της Μεθόδου, καθώς και πολλές από τις εφαρμογές της, επικεντρώνοντας το ενδιαφέρον μας σε αυτές που εμφανίζονται στο πεδίο της Αλγοριθμικής Θεωρίας Παιγνίων, και πιο συγκεκριμένα του Σχεδιασμού Μηχανισμών. Τα κριτήρια επιλογής των εφαρμογών ήταν τόσο η σημαντικότητά τους στο αντίστοιχο ερευνητικό πεδίο στο οποίο ανήκουν, όσο και η δομή τους που κάνει εύκολη την παρουσίασή τους ως υπο-περιπτώσεις της γενικής Μεθόδου.

Λέξεις Κλειδιά: Μέθοδος Πολλαπλασιαστικών Συντελεστών, Αλγοριθμική Θεωρία Παιγνίων, Άμεσος Σχεδιασμός Μηχανισμών, Φιλαλήθεις Συνδυαστικές Δημοπρασίες

Abstract

The Multiplicative Weights Update Method is the formulation of a framework capturing a very useful idea encountered in many diverse fields and applicable to a wide variety of optimization and learning problems. The algorithms that fit in this framework maintain a probability distribution (i.e. weights) over a certain set that is updated iteratively by a multiplicative rule, while the analysis of these algorithms relies on quantifying the change in an exponential potential function. In this thesis, we are going to introduce the framework and present a variety of its applications, concentrating our interest in the field of Algorithmic Game Theory, and even more specifically in Mechanism Design. The applications we consider, are chosen having in mind their importance in their respective fields and their structure which helps us point out that they are all instances of our (more general) meta-algorithm.

Keywords: Mupltiplicative Weights Update Method, Algorithmic Game Theory, Online Mechanism Design, Truthful Combinatorial Auctions

Contents

Chapter 1

Introduction

1.1 The Multiplicative Weights Update Method

It is very often that people need to make a decision having only limited or even worse no knowledge on the matter. If this decision is to be made once, all we can do is to choose what is best for us based on our knowledge at the time. But if this decision making procedure was to be repeated, our knowledge would grow in each round since we would receive the consequences of our decision, forcing us to re-evaluate our choices, and in this way increasing our chances of making a better decision in the future. This natural learning procedure is the root of an idea encountered in various algorithms. The idea is to maintain a distribution (i.e. weights) over a set and use a multiplicative rule to update the weights of each member of the set. In the decision problem, the set would be the possible choices we have and the multiplicative weights update rule would help us re-evaluate them.

Interestingly, all these algorithms belong to many diverse fields, they do not necessarily look similar when examined in detail, and most of them have discovered independently this main multiplicative idea. Even though the fact that the idea can be generalized had already been noticed ([30]), it was only recently ([1]) that the idea was formulated as a meta-algorithm, which treats the aformentioned algorithms as special cases. The significance of the meta-algorithm stems from its simplicity and the variety of its applications. Of course, in some cases its results are not optimal since information of the specific problem might be ignored for the sake of generality. In these cases, certain adjustments should be considered in order to yield even better results than those that occur by the straightforward application of the meta-algorithm.

In this thesis, we are interested in the impact of this method on Algorithmic Game Theory and more specifically on Mechanism Design, both of which are introduced in the next section.

1.2 Algorithmic Game Theory

In every possible model of a society, there is an interaction between its members. The decisions and actions of a member affect the others and vice versa. Unfortunatelly, more often than not, the incentives of the members are conflicting.

Algorithmic Game Theory considers these members as selfish players taking part in a game, meaning that they interact with each other in an effort to maximize their personal benefit. It is the study aiming to predict or even guide the players' behaviour in order to serve the benefits of the society as a whole or, at times, the benefits of a specific group.

1.2.1 Mechanism Design

The rules regulating the aformentioned game are the mechanism of the game. Usually, the players try to use those rules to manipulate the decision making procedure in order to achieve a better outcome for themselves. A very important research area studies exactly the impact of these manipulations to the greater good. The way to avoid them and to guide the players' behaviour is through Mechanism Design. More specifically, we try to align the goal of the mechanism with the individual goals of the players.

1.3 Notation and Preliminaries

1.3.1 Algorithmic Notions

Optimization problems seek a feasible solution of the problem that, in addition, maximizes (respectively, minimizes) a certain function. The value of this function on the solution is the *cost* of the solution. By OPT, we usually denote the optimal cost.

Definition 1. *An* approximation algorithm *is an algorithm that finds a feasible solution for the optimization problem whose cost is at least (respectively, at most)* $\frac{1}{\rho}OPT$ *(respectively,* ρOPT*) if it is a maximization (respectively, minimization) problem. The parameter* ρ *is called the* approximation ratio *and may depend on other parameters of the input instance.*

A useful way to represent optimization problems is via a *linear program* (LP), in which the set of all feasible solutions is represented by a polyhedron. Specifically, a maximization linear program in its canonical form is the following:

$$\begin{aligned} \max \quad & c^T x \\ & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where c and b are vectors, A is a matrix of coefficients and the desired solution is x. The most common problems that we face can be formulated into an Integer LP, meaning that the coefficients of x are integer. Since such a program is difficult to solve, we usually solve the LP with the last constraint relaxed and round the result into an integer solution via a randomized or deterministic rounding procedure.

Definition 2. *A* randomized algorithm *is one that uses uniformly random bits in addition to its input in order to make decisions, therefore the decisions the algorithm makes in each input can not be predicted from the start.*

Another variant of an algorithm depends on the way they access their input.

Definition 3. *As opposed to the commonly used algorithms,* online *algorithms access their input in a serial manner, in the sense that it is revealed to them while the algorithm is running and it is not available from the start. The comparison between the performance of the online and the offline algorithm of a problem gives us their* competitive ratio.

1.3.2 Game-Theoretic Notions

As we discussed, each player makes decisions based on his incentive. Those decisions, are called *strategies*, denoted s_i for each player $i \in \{1, \ldots, n\}$. There is also a set of outcomes O and each agent has a private valuation function $v_i : O \to \mathbb{R}$ from a set of possible valuations V_i, which represents how "satisfied" he is from this outcome. A valuation profile is the vector of the valuation functions of all players $v = (v_1, \ldots, v_n)$ and by v_{-i} we will denote the valuation profile of all players except i.

The Mechanism is therefore called to find the allocation $f : V_1 \times \ldots \times V_n \to O$, which returns the outcome of the mechanism depending on the valuations he receives as input, according to an *allocation rule*.

The problem is that the valuations of the players are private and they do not have to reveal them unless it helps them reach the desired outcome. So the actual input of f is a set of *bids* which may or may not be the true valuation functions of the players. This is why the players need to have benefit in reporting their true valuations.

Definition 4. *An allocation rule is* truthful *iff* $\forall i \in N,\ v_i \in V_i,\ v_{-i} \in V_{-i}, v_i' \in V_i$*:*

$$v_i(f(v_{-i}, v_i)) \geq v_i(f(v_{-i}, v_i'))$$

where (v_{-i}, v_i') *is the valuation profile where* i*'s valuation function is replaced by* v_i'*.*

If a strategy (i.e. the bids) of a player always leads to as good an outcome as any other strategy then this strategy is the best a player can do and so it is his *dominant strategy*. Formally:

Definition 5. *The valuation function* $t \in V_i$ *is a dominant strategy for agent* i *if* $\forall\ v_{-i} \in V_{-i}, t' \in V_i$*:*

$$v_i(f(v_{-i}, t)) \geq v_i(f(v_{-i}, t'))$$

We will restrict the types of mechanisms we study to those in which every player only reveals their bid (*direct mechanisms*), and we will do so without loss of generality, as the next theorem states.

Theorem 6 (Revelation Principle)**.** *Any mechanism that implements a certain allocation rule* f *in dominant strategies, can be transformed to a* direct truthful *mechanism implementing* f*.*

A negative result, shows that there is no non-trivial (non-dictatorial) allocation rule for games with more than 3 outcomes that is truthful. After this result, modifications had to be made in order to yield practical mechanisms. The mechanisms we will discuss, will give to the designer the ability to impose payments on the players, depending on the outcome of the mechanism. Therefore, a mechanism

in addition to providing an allocation, is also called to impose payments according to another function, namely the *payment rule* $p : V \to \mathbb{R}^n$. Now, the players aim to maximize the quantity $v_i(f(v_i, v_{-i})) - p(v_i, v_{-i})$, which we call *utility*. The new definition of truthfulness is the following:

Definition 7. *A mechamism $M = (f, p)$ is defined as truthful iff $\forall i \in N,\ v_i \in V_i,\ v_{-i} \in V_{-i},\ v_i' \in V_i$:*

$$v_i(f(v_{-i}, v_i)) - p(v_{-i}, v_i) \geq v_i(f(v_{-i}, v_i')) - p(v_{-i}, v_i')$$

Definition 8. *An allocation rule f is* impementable *if there exists a payment rule p such that the mechanism (f, p) is truthful.*

Most mechanisms belong into one of two categories: *Revenue* maximizing and *Social Welfare* Maximizing. The definitions of those two quantities are the following:

Definition 9. *For a mechanism $M = (f, p)$, we define as the* revenue *of the mechanism the quantity*

$$Rev(v) = \sum_i p_i(v)$$

Definition 10. *For an outcome $o \in O$ we define as* social welfare *(SW) the quantity*

$$SW(o) = \sum_i u_i(o)$$

Usually, we are concerned with Social Welfare, as it represents the "happiness" of the total of the players or as we mentioned before "the greater good".

Finally, a well-known mechanism that maximizes the Social Welfare and manages to be truthful is the VCG-Mechanism (by Vickrey, Clarke and Groves). This mechanism, requires from each agent to pay the loss of "happiness" he causes to all the other players by his presence in the game. In addition, in most cases it has the natural property which guarantees that no player loses by participating in the game (utilities≥ 0) and the payments are not negative (the mechanism does not give money to the players).

Definition 11 (VCG mechanism [29, 12, 20])**.** *The VCG mechanism is defined by (f, p) such that*

$$f(v) = \text{argmax}_o \sum_i v_i(o)$$

$$p_i(v) = \sum_{j \neq i} v_j(f(v_{-i})) - \sum_{j \neq i} v_j(f(v))$$

Chapter 2

The Multiplicative Weights Update Method

In this Chapter, we are going to present the Multiplicative Weights Update Method as shown in [1]. First, we will give an example in a simpler setting which makes it easier to understand the general meta-algorithm that follows it. Next, we will show how the method has been used to yield good results in an important field and how the use of specific information about the problem can help increase the approximation ratio as we discussed in Chapter 1. Finally, we will briefly describe other applications of the method that demonstrate its generality and importance.

2.1 The Weighted Majority Algorithm

We will first describe the *Weighted Majority Algorithm* introduced in [24] which is based on a very simple setting and leads naturally to the generalized meta-algorithm. To do so, we consider the *Prediction from Expert Advice Problem* where we want to predict the movement of a stock ("up" or "down") each day without any knowledge other than the predictions of n experts.

The algorithm maintains weights for the experts (initialized to 1), making a prediction in each round based on the *weighted majority* of the experts' predictions. In other words, if the total weight of all experts predicting "up"("down") is more than half the weight of all the experts then we predict "up"("down"), breaking ties arbitrarily. At the end of each round, the weight of each expert that predicted incorrectly is decreased, depicting the fact that we now trust him less than before and do not want his opinion to play such an important role in the next decision.

In particular, let $w_i^{(t)}$ be the weight of the ith expert for $i = 1, \ldots, n$, during round $t = 1, \ldots, T$. The algorithm is the following:

WEIGHTED MAJORITY ALGORITHM
Initialization: Fix an $\eta \leq \frac{1}{2}$. Set $w_i^{(1)} = 1 \ \forall i \in \{1, \ldots, n\}$.
At every step $t = 1, \ldots, T$**:**

1. If $\sum_{i \in U} w_i^{(t)}$, where U is the set of experts predicting "up"(resp., "down"), is greater than $\Phi^{(t)}/2$, where $\Phi^{(t)} = \sum_i w_i^{(t)}$ is the potential function, then predict "up"(resp., "down"). Break ties arbitrarily.

2. Depending on the movement of the stock, adjust the weight of each expert i that predicted incorrectly as follows:

$$w_i^{(t+1)} = w_i^{(t)}(1 - \eta)$$

The most interesting property of the *Weighted Majority Algorithm* is that it guarantees that our performance is roughly as good as it would be if we were to consult only the expert that made the fewest mistakes overall, i.e. the *best* expert. The following theorem quantifies this guarantee.

Theorem 12. *Let $M^{(T)}$ be the total number of mistakes our algorithm has made and $m_i^{(T)}$ the total number of mistakes expert i has made. Then the following bound holds for every i:*

$$M^{(T)} \leq 2(1 + \eta)m_i^{(T)} + \frac{2 \ln n}{\eta}$$

Clearly, the bound is tighter for the least $m_i^{(T)}$.

In order to understand the bound better, imagine the extreme case in which the mistakes made by the best expert are close to none. Then the quantity $\frac{2\ln n}{\eta}$ dominates the sum. This quantity depends only on the number of the experts and the fixed parameter η and not on the mistakes of any of the experts. It represents the mistakes the algorithm cannot avoid to make, since this is the only way to increase its knowledge and finally locate the best expert. As one would expect, increasing the number of experts n means that the algorithm can be fooled more easily so it needs to perform more experiments until it rejects the bad experts. On the contrary, increasing the parameter η means that each mistake decreases significantly the importance of an expert and helps the algorithm to distinguish the good ones faster.

On the other hand, if the mistakes of the best expert are too many then it is the quantity $2(1+\eta)m_i^{(T)}$ that dominates the sum. Imagine that instead of following our algorithm, we chose deterministicaly "up" or "down" in each round. Then we expect that with probability $\frac{1}{2}$ our guess would be the same as the best expert's, so our mistakes would be within a factor 2 of the expected mistakes of the best expert. Again, this quantity increases if we increase η and, intuitively, this is because a larger η helps in distinguishing the best expert and ignore the rest, making the best expert's mistakes very important to the decision and therefore able to cause more damage.

The proof of the theorem relies on the potential function and is very similar to the analyses of other algorithms based on this idea. Thus, it is presented now in detail:

Proof. By induction, it is easy to see that

$$w_i^{(t+1)} = (1-\eta)^{m_i^{(t)}} \tag{2.1}$$

Each time the algorithm makes a mistake, at least the weighted half of the experts must have made a mistake, therefore their weights decrease by a factor $(1-\eta)$. If $a \geq n/2$ is the number of experts that predicted incorrectly then

$$\Phi^{(t+1)} \leq \Phi^{(t)}\Big(\frac{n-a}{n} + \frac{a}{n}(1-\eta)\Big) = \Phi^{(t)}\Big(1 - \frac{a}{n}\eta\Big) \leq \Phi^{(t)}(1-\eta/2) \tag{2.2}$$

It is easily shown by induction from 2.2 that

$$\Phi^{(T+1)} \leq \Phi^{(1)}(1-\eta/2)^{M^{(T)}} = n(1-\eta/2)^{M^{(T)}} \tag{2.3}$$

Finally, for every i it holds:

$$\Phi^{(T+1)} \geq \; w_i^{(T+1)} \overset{2.3,2.1}{\Longrightarrow}$$

$$n(1-\eta/2)^{M^{(T)}} \geq (1-\eta)^{m_i^{(T)}} \overset{\ln(\cdot)}{\Longrightarrow}$$

$$M^{(T)} \ln(1-\eta/2) + \ln n \geq m_i^{(T)} \ln(1-\eta) \Longrightarrow$$

$$M^{(T)}(-\ln(1-\eta/2)) \leq \ln n + m_i^{(T)}(-\ln(1-\eta))$$

To obtain the result, we need to make some observations on the last inequality.

- LHS= $M^{(T)}(\frac{\eta}{2} + \frac{1}{2}(\frac{\eta}{2})^2 + \ldots) \geq M^{(T)}\frac{\eta}{2}$. This holds because all we did was to replace $\ln(1-x)$ with its Taylor series expansion which certainly converges since $|x| = |-\eta/2| \leq 1$.
- RHS$\leq \ln n + m_i^{(T)}(\eta + \eta^2)$. This is also true, being a well known inequality that holds for $\eta \leq 1/2$.

It follows that $M^{(T)}\frac{\eta}{2} \leq \ln n + m_i^{(T)}(\eta+\eta^2) \Longrightarrow M^{(T)} \leq 2(1+\eta)m_i^{(T)} + \frac{2\ln n}{\eta}$. $\square$

2.2 The Multiplicative Weights Update Algorithm

In the stock movement problem, the cost of an expert is binary in each round: if the outcome of the round is that the stock is "up" then the cost of every expert predicting "down" is 1, while the cost of the rest is 0. In the general setting, we still have n decisions (experts) but the costs of our decisions are not necessarily binary. In each round t, each decision i incurs a certain cost $m_i^{(t)}$, revealed to us after we have made our choice, which is based on the outcome of the round. These cost vectors $\mathbf{m}^{(t)}$ represent the penalties (or rewards in case of $m_i^{(t)} \leq 0$) that will affect the weight of decision i in the next round. The main difference between the *Weighted Majority Algorithm* and the generalized *Multiplicative Weights Algorithm* is the way we make our choice in each round. In this general framework, we pick a decision with a probability that depends on its weight. The algorithm is the following:

MULTIPLICATIVE WEIGHTS UPDATE ALGORITHM
Initialization: Fix an $\eta \leq \frac{1}{2}$. With each expert i, associate the weight $w_i^{(1)} = 1$.
At every step $t = 1, \ldots, T$**:**

1. Choose decision i with probability $p_i^{(t)} = w_i^{(t)}/\Phi^{(t)}$, where $\Phi^{(t)} = \sum_{i=1}^{n} w_i^{(t)}$.

2. Based on the costs $\mathbf{m}^{(t)}$, adjust the weight of each decision i as follows:

$$w_i^{(t+1)} = w_i^{(t)}(1 - \eta m_i^{(t)})$$

Note that the positive costs represent penalties and therefore decrease the decision's weight while the negative ones (rewards) increase it. In order to discuss the performance of this algorithm, we need to define the expected cost in round t as $\mathbf{E}_{i \in \mathbf{p}^{(t)}}[m_i^{(t)}] = \mathbf{m}^{(t)}\mathbf{p}^{(t)}$, so the total expected cost is $\sum_{t=1}^{T} \mathbf{m}^{(t)}\mathbf{p}^{(t)}$. Again, we compare this with the decision that performs best overall (i.e. has the least $\sum_{t=1}^{T} m_i^{(t)}$). The next theorem is equivalent to Theorem 12.

Theorem 13 (Main Theorem)**.** *Let all costs satisfy $m_i^{(t)} \in [-1, 1]$ and $\eta \leq 1/2$. The following bound holds for every i after T rounds:*

$$\sum_{t=1}^{T} \boldsymbol{m}^{(t)}\boldsymbol{p}^{(t)} \leq \sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t=1}^{T} |m_i^{(t)}| + \frac{\ln n}{\eta}$$

Clearly, the bound is tighter for the best decision.

The assumption that the costs' width is at most 1 is not unnatural, since we can assume that they are normalized. Furthermore, it can be proven via the probabilistic method that the bound is tight (up to constants in the additive error) for every online decision-making algorithm.

If we take the convex combination of the n inequalities guaranteed by Theorem 13 then we have the following corollary, which is very useful in applications.

Corollary 14. *After T rounds, for any distribution $\boldsymbol{p}$ it holds:*

$$\sum_{t=1}^{T} \boldsymbol{m}^{(t)}\boldsymbol{p}^{(t)} \leq \sum_{t=1}^{T}(\boldsymbol{m}^{(t)} + \eta|\boldsymbol{m}^{(t)}|)\boldsymbol{p}^{(t)} + \frac{\ln n}{\eta}$$

where $|\boldsymbol{m}^{(t)}| = (|m_1^{(t)}|, \ldots, |m_n^{(t)}|)$.

A dual result can be obtained when the vector $\mathbf{m}^{(t)}$ represents gains instead of losses, in which case we want a lower bound guarantee for the expected gain. The bound is the following, and it is a result of simply running the algorithm with cost vector $-\mathbf{m}^{(t)}$:

$$\sum_{t=1}^{T} \mathbf{m}^{(t)}\mathbf{p}^{(t)} \geq \sum_{t=1}^{T}(\mathbf{m}^{(t)} - \eta|\mathbf{m}^{(t)}|)\mathbf{p}^{(t)} - \frac{\ln n}{\eta} \tag{2.4}$$

2.3 The Plotkin-Shmoys-Tardos Framework

One of the main applications that demonstrates the power of the method is the PST Framework for packing/covering LPs [27]. The goal is to check the feasibility of the following convex program:

$$Ax \geq b, x \in \mathbb{P}$$
where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $\mathbb{P}$ is a convex set in $\mathbb{R}^n$.

Intuitively, $\mathbb{P}$ is a convex set that represents the "easy" constraints of a linear program such as non-negativity and $Ax \geq b$ represent the m constraints that are "hard" to satisfy. To solve the problem approximately, we design an algorithm that either returns an $x \in \mathbb{P}$ such that $A_i x \geq b_i - \epsilon$ holds for all i for an error parameter $\epsilon > 0$, or fails to return such an x, thus proving that the system is infeasible. By A_i we denote the ith row of the matrix A, i.e. the ith constraint.

In order to design such an algorithm we need to assume the existence of an Oracle which given a convex set $\mathbb{P} \subseteq \mathbb{R}^n$, and a probability vector $\mathbf{p}$ on the m constraints, returns a vector $x \in \mathbb{P}$ that satisfies the inequality $\mathbf{p}^T Ax \geq \mathbf{p}^T b$, if such an x exists.

We apply our method to the problem, corresponding:

- Each decision $i \in \{1, \ldots, m\}$ to the constraint $A_i x \geq b_i$.
- Each outcome with the $x^{(t)} \in \mathbb{P}$ presented by the Oracle in round t.
- Each coordinate of the penalty vector $m_i^{(t)} = (A_i x^{(t)} - b_i)/\rho$.

The parameter ρ is equal to the largest value $A_i x^{(t)} - b_i$ for every i and t and it is called the *width*. The width plays an important role in both the convergence time and the approximation result.

Our method implies that we call the Oracle in each round and update the weights and consequently the distribution over the constraints. It is somewhat counter-intuitive, though, for the more satisfied constraints to receive a larger penalty. To understand this better, one should think of the Oracle's procedure. It searches for an x that satisfies a single inequality which is produced as a convex combination of the constraints, with coefficients analogous to the weights. So if a constraint is far from being violated, its coefficient should be decreased in order to make the Oracle focus on other, more "difficult" constraints. By application of Theorem 13, the next result occurs:

Corollary 15. *After $T = O(\frac{\rho^2}{\epsilon^2} \ln m)$ rounds, the algorithm finds a solution $\overline{x}$ that satisfies the constraints within an additive error, i.e. $A_i \overline{x} \geq b_i - \epsilon$, if one exists, otherwise it concludes that the system is infeasible.*

Proof. First, notice that $\mathbf{m}^{(t)} \mathbf{p}^{(t)} \geq 0$ holds by the Oracle's guarantee. Thus, for the LHS of the inequality of Theorem 13 it holds that $\sum_{t=1}^{T} \mathbf{m}^{(t)} \mathbf{p}^{(t)} \geq 0$. For the RHS:

$$
\begin{aligned}
&\sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t=1}^{T} |m_i^{(t)}| + \frac{\ln m}{\eta} \\
&= (1+\eta) \sum_{\geq 0} m_i^{(t)} + \Big[(1-\eta) \sum_{<0} m_i^{(t)} \Big] + \frac{\ln m}{\eta} \\
&= (1+\eta) \sum_{\geq 0} m_i^{(t)} + \Big[(1+\eta) \sum_{<0} m_i^{(t)} + 2\eta \sum_{<0} |m_i^{(t)}| \Big] + \frac{\ln m}{\eta} && (|m_i^{(t)}| \leq 1) \\
&\leq (1+\eta) \sum_{t=1}^{T} m_i^{(t)} + 2\eta T + \frac{\ln m}{\eta} && (m_i^{(t)} \leq 1) \\
&\leq \sum_{t=1}^{T} m_i^{(t)} + 3\eta T + \frac{\ln m}{\eta}
\end{aligned}
$$

Multiplying by $\frac{\rho}{T}$ and substituting the cost variables with their values, we get the following inequality:

$$
\frac{1}{T} \sum_{t=1}^{T} [A_i x^{(t)} - b_i] + 3\eta\rho + \frac{\rho \ln m}{\eta T} \geq 0 \Leftrightarrow A_i \overline{x} \geq b_i - \epsilon \tag{2.5}
$$

Where the error parameter is $\epsilon = 6\rho\eta$, the number of rounds is $T = \lceil \frac{12\rho^2 \ln m}{\epsilon^2} \rceil$ and the solution $\overline{x}$ is the average of the outcomes that the ORACLE returns.

As for the case of infeasibility, if at some round t the ORACLE fails to return a solution then by linear programming theory (Farkas' lemma), it is proven that the system is infeasible, and the distribution $\mathbf{p}^{(t)}$ is the witness.

□

This result is very intuitive because it shows the tradeoff between the rounds that the algorithm runs and the error we obtain. In order to get a solution very close to the optimal, the algorithm must run for a long time, as one would expect.

Furthermore, if we set $\mathbb{P} = \{x : x \geq 0, c^T x = \text{cost(OPT)}\}$ where c^T is the vector of the objective function of the covering LP, then our solution $\overline{x}$ will also have optimum cost, as all $x^t \in \mathbb{P}$ and $\overline{x}$ is the average of those outcomes. In the next section, we present an application of the PST Framework regarding a problem with LP formulation that uses this remark.

2.3.1 The ORACLE

Let us define an (ℓ, ρ)-*bounded* ORACLE to be an algorithm which given a probability vector $\mathbf{p}$, solves the previous feasibility problem but also satisfies the inequalities: $A_i x - b_i \in [-\ell, \rho]\ \forall i \in I$ and $A_i x - b_i \in [-\rho, \ell]\ \forall i \in I$ (for a fixed subset $I \subseteq [m]$).

In the previous analysis, we assumed that the absolute values of the costs corresponding to the ORACLE's outcomes are bounded by ρ. If we allow (ℓ, ρ)-*bounded* ORACLEs, then our results would be tighter. For the sake of simplicity, though, the previous analysis is restricted to (ρ, ρ)-*bounded* ORACLEs.

Another point left unspecified in the previous analysis is the procedure of the ORACLE. It is not the subject of this thesis to describe the algorithm used to find a point in a convex set satisfying a single inequality, but the running time of such an algorithm matters to our framework. Fortunatelly, we can assume a running time of $O(m)$ per call.

Last but not least, the results hold even for ϵ/c-approximate ORACLEs, for a constant c. In this case, the number of rounds T is multiplied with a constant factor, therefore the asympotic result remains the same.

2.4 Width Reduction: Approximating Maximum Multicommodity Flow

The Maximum Multicommodity Flow Problem is represented by a LP and therefore gives a good example of the PST Framework. Unfortunatelly, straitforward application of the framework yields an algorithm with non-polynomial running time, which is caused by the fact that the width ρ depends on the capacity of the edges of the graph (and the number of the iterations T of the algorithm depends on ρ, as we can see from Corollary 15). Garg and Könemann [19] managed to reduce the width parameter ρ and derive a better algorithm which gives an $(1-\epsilon)^2$ approximation factor in $O(T_{sp} m \log m/\epsilon^2)$ running time, where T_{sp} is the time needed to find the shortest paths between the k source-sink pairs of the commodities.

First, let P be the set of all paths between the sounce-sink pairs and f_p the flow on path $p \in P$. We need to maximize the flow on all the source-sink pair paths, while the total flow of all paths using edge e cannot exceed the capacity of e. The LP formulation of the Maximum Multicommodity Flow Problem is the following:

$$\begin{aligned} \max \quad & \sum_{p \in P} f_p \\ & \sum_{p \ni e} f_p \leq c_e \ : \ \forall e \in E \\ & f_p \geq 0 \qquad \ : \ \forall p \in P \end{aligned}$$

Equivalently, we need to find a flow f such that:

$$\sum_{p \ni e} f_p \leq c_e \ : \ \forall e \in E, x \in \mathbb{P}$$

$$\text{where } \mathbb{P} := \Big\{ f : \sum_{p \in P} f_p = F^{OPT}, f_p \geq 0 : \ \forall p \in P \Big\}.$$

Note that we can easily find (via binary search on the lower bound of the objective) the value F^{OPT} that guarantees that our solution is of optimal cost.

A careful application of our method, would correspond:

- Each decision $e \in \{1, \ldots, |E|\}$ to the constraint $\sum_{p \ni e} \frac{f_p}{c_e} \leq 1$.
- Each outcome with the flow through path $p^{(t)}$ that satisfies $\sum_e w_e^{(t)} \sum_{p \ni e} \frac{f_p}{c_e} \leq 1$, presented by the ORACLE in round t.
- Each coordinate of the penalty vector $m_e^{(t)} = (\sum_{p \ni e} \frac{f_p}{c_e} - 1)/\rho$.

where the width parameter is $\rho = \max_{f \in \mathbb{P}} \max_e (\sum_{p \ni e} \frac{f_p}{c_e} - 1) = \frac{F^{OPT}}{\min_e c_e} - 1$, which unfortunatelly may be exponential on the size of the input graph, as we discussed.

Note that it suffices for the ORACLE to minimize the quantity $\sum_e w_e^{(t)} \sum_{p \ni e} \frac{f_p}{c_e}$ since the RHS of the inequality is 1. Minimizing $\sum_e w_e^{(t)} \sum_{p \ni e} \frac{f_p}{c_e} = \sum_p f_p \sum_{e \in p} \frac{w_e^{(t)}}{c_e}$, would mean to pass a flow of F^{OPT} cost through a path $p^{(t)} \in P$ of shortest length, where the length of each edge is $w_e^{(t)}/c_e$.

Garg and Könemann managed to modify the method in a way that reduces the width parameter. They did so by removing the optimality constraint of the convex set P, meaning that the ORACLE no longer needs to pass a flow of optimal cost through the paths, but passes as much flow as it can, provided that it does not exceed the minimum capacity of the edges of the path. More specifically, the ORACLE finds the shortest path $p^{(t)}$ with respect to edge lengths $w_e^{(t)}/c_e$ and routes as much flow as is allowed by the minimum capacity edge of the path, $c^{(t)}$. The coordinates of the cost vector are now $m_e^{(t)} = c^{(t)}/c_e$ for every edge on the chosen path and $m_e^{(t)} = 0$ for all other edges. Therefore the width parameter is reduced to 1. The algorithm is the following:

MAXIMUM MULTICOMMODITY FLOW ALGORITHM

Initialization: Fix $\eta \leq \frac{1}{2}$. With each edge e, associate the weight $w_e^{(1)} = \delta$ and set $f_p = 0$

At every step t, until some $w_e^{(t)} \geq 1$, do:

1. Choose shortest source-sink path $p^{(t)}$ using lengths $w_e^{(t)}/c_e$.
2. Increase flow $f_p = f_p + c^{(t)}$, where $c^{(t)}$ is the bottleneck capacity of $p^{(t)}$.
3. Adjust the weight of each edge e as follows:

$$w_i^{(t+1)} = w_i^{(t)}(1 + \eta \frac{c^{(t)}}{c_e})$$

Return: f scaled to be primal feasible

One would notice that the initialization is different, and this is because the variable δ is chosen during the approximation analysis in a way that enhances the result. Also, the final flow needs to be scaled down, since, as in the original framework, the outcomes of each round are not guaranteed to be a feasible solution for the system of inequalities. Finally, the termination rule is slightly different and it helps us infer the number of iterations T.

At first the Garg-Könemann algorithm seems to have more modifications than similarities with the algorithm implied by the PST Framework. But a closer look in those modifications helps us realize that they are ideas that are natural to the problem. From there on, the analysis uses the Corollary 14, with gains instead of losses, since our problem is a maximization problem and the edge weights are increased every time an edge is used so that it is less likely to be a part of the

shortest path in the next round. So even if the straightforward application of the framework does not yield good results, a few natural modifications of it give very good approximation and running time results, using the framework's analysis.

For illustrative purposes, we present next the analysis of the approximation ratio and the running time but if the reader understood the previous section's analysis, then it can be ignored since it does not provide any further intuition.

Proof. The weight of each edge is increased by a factor $(1+\eta)^{1/c_e}$ for each unit of flow routed through e. Let f_e be the total amount of flow passing through e and e^* the first edge for which: $w_{e^*}^{(T)} \geq 1$. Then

$$\delta(1+\eta\frac{f_{e^*}}{c_{e^*}}) \geq 1 \Rightarrow \frac{f_{e^*}}{c^{e^*}} \geq \frac{\ln\frac{1}{\delta}}{\ln(1+\eta)} \tag{2.6}$$

where the latter follows from the fact that $(1-\eta)^{|x|} \leq (1-\eta|x|)$ for $|x| \leq 1$.

From Theorem 13, the total expected cost (gain) and the total cost (gain) incurred by e^*, f_{e^*}/c_{e^*}, are related as follows:

$$\sum_{t=1}^{T}\frac{\sum_{e\in p^{(t)}}\frac{c_{(t)}}{c_e}w_e^{(t)}}{\sum_e w_e^{(t)}} \geq \frac{\ln(1+\eta)}{\eta}\frac{f_{e^*}}{c_{e^*}} - \frac{\ln m}{\eta} \overset{2.6}{\Rightarrow} \sum_{t=1}^{T}\frac{\sum_{e\in p^{(t)}}\frac{c_{(t)}}{c_e}w_e^{(t)}}{\sum_e w_e^{(t)}} \geq \frac{\ln\frac{1}{\delta}}{\eta} - \frac{\ln m}{\eta} = \frac{\ln\frac{1}{m\delta}}{\eta} \tag{2.7}$$

Now we claim that $\sum_{t=1}^{T}\frac{\sum_{e\in p^{(t)}}\frac{c_{(t)}}{c_e}w_e^{(t)}}{\sum_e w_e^{(t)}} \leq F/F^{OPT}$ where $F = \sum_{t=1}^{T} c^{(t)}$ is the total flow routed in all rounds. Suppose f_p^{OPT} is the flow that the optimal solution assigns to path p. For any set of lenghts w_e/c_e the shortest path satisfies:

$$\frac{\sum_e w_e}{\sum_{e\in p}\frac{w_e}{c_e}} \geq \sum_{p'} f_{p'}^{OPT} = F^{OPT} \tag{2.8}$$

Therefore by 2.7: $\frac{F}{F^{OPT}} \geq \frac{\ln\frac{1}{m\delta}}{\eta}$

In order to get a feasible flow we scale down F by $\log_{1+\eta}\frac{1+\eta}{\delta}$ so choosing $\delta = (1+\eta)((1+\eta)m)^{-1/\eta}$ the approximation ratio becomes $(1-\eta)^2$. If we set $\eta = \epsilon$ then we have $(1-\epsilon)^2$.

Regarding the running time: Any edge can be the minimum capacity edge at most $log_{1+\eta}\frac{1}{\delta}$ times. So within $mlog_{1+\eta}\frac{1}{\delta} = O(\frac{mlogm}{\eta^2})$ iterations, the algorithm must terminate. Each iteration involves k shortest path computations thus $O(mk)$ time. The overall running time is, therefore, $O(\frac{km^2logm}{\epsilon^2})$ (for $\eta = \epsilon$).

□

2.5 Other applications

As we stated earlier, important applications of the method are met in many diverse fields such as Machine Learning (The Boosting Algorithm), Semi-Definite Programming (0.878 approximation for Max Cut), Graph Theory (Sparsest Cuts), and the proof of Yao's XOR lemma. We will list some applications in this section that illustrate the usefullness of the idea and are easier to understand without demanding additional knowledge of their respective fields.

2.5.1 Approximating NP-hard problems: Set Cover

NP-hard problems which can be formulated in integer packing/covering linear programs can be approximated within $O(\log n)$ of the optimal solution. The randomized algorithm that gives this result uses randomized rounding to find an integer solution based on the solution of the LP relaxation of the problem. The deterministic algorithm simulates the randomized algorithm using Raghavan's derandomization method of *pessimistic estimators* ([28]).

Combining the ideas above, N.Young introduced in [30] a technique called *oblivious rounding*, which yields an algorithm that uses the multiplicative update rule. In the paper, N.Young uses his technique on the Set Cover problem, which is a representative of the class of **NP**-hard problems and proves that it gives the well-known $O(\log n)$-approximation greedy algorithm. We will now show how the Multiplicative Weights Update Method gives us the $O(\log n)$ approximation of Set Cover and finally we will try to give the intuition behind Young's framework and its connection to our method.

The Set Cover problem is defined as follows: We are given as input a set of elements $U = \{1, \ldots, n\}$ and a collection $\mathcal{C}$ of sets whose union equals U. The required output is the minimum number of sets from $\mathcal{C}$ covering U.

In the Multiplicative Weights Update Method we correspond:

- Each decision to an element $i \in \{1, \ldots, n\}$ that has to be covered.
- Each outcome to the set $C_{j^{(t)}} \in \mathcal{C}$, returned by the ORACLE, which maximizes the expected cost under $\mathbf{p}^{(t)}$ and as a result, the number of uncovered elements that it covers.
- Each coordinate of the penalty vector $m_i^{(t)} = \begin{cases} 1 & \text{if } i \in C_{j^{(t)}} \\ 0 & \text{otherwise.} \end{cases}$

The algorithm for $\eta = 1$ corresponds to the degenerate case in which the weight of an element is reduced to zero when it is covered and therefore the weights of the uncovered (at the time) elements form a uniform distribution. This is why maximizing the expected cost $\sum_{i \in C} p_i = \frac{|C \cap S_{\text{uncovered}}|}{|S_{\text{uncovered}}|}$ is the same as choosing C to maximize the number of uncovered elements that it covers. There is a slight

technical requirement for $\eta \leq 1/2$ but we can adjust the weights in a way so that the analysis is the same.

The algorithm described above yields a $\lceil \ln n \rceil$ approximation since in $T = \lceil \ln n \rceil \text{OPT}$ rounds it covers all the elements while only one set is chosen in each round, making the cost of our solution equal to the number of rounds.

The idea behind Young's oblivious rounding was Chernoff's bounds. The most known such bound shows that the sum of independent random variables $X_1, \ldots, X_n$ is highly concentrated around its mean and the way to prove this is by using Markov's inequality on the quantity $e^{\eta \sum_{i=1}^{n} X_i}$. Notice that if we have a partial sum with a "potential function" $e^{\eta \sum_{i} X_i}$ then adding a new X_j to the sum would mean to multiply the potential function with $e^{\eta X_j}$ (which is equal to $(1 - \eta X_j)$ for small ηX_j). Therefore, one can bound the probability of failure by $\mathbf{E}[e^{\eta \sum_{i} X_i}]$, so the latter is a pessimistic estimator of this probability and by choosing in each round the X_i that decreases the pessimistic estimator, we achieve the desired approximation ratio.

2.5.2 Algorithmic Game Theory: Zero-Sum Games

The Multiplicative Weights Update Method has many applications in Algorithmic Game Theory, which we will discuss on Chapter 3. But, zero-sum games provide a very intuitive and easy example of the method's application.

A zero-sum game is a situation in which a player's gain (respectively, loss) is exactly balanced by the losses (respectively, gains) of the other players. Thus, the total gains and losses sum to zero. In a two player zero-sum game, there is a $n \times m$ size matrix M, each entry $M(i, j)$ of which is w.l.o.g. in $[0, 1]$ and represents the payoff of the column player when he plays strategy j and the row player plays strategy i. Furthermore, if the row (respectively, column) player chooses his strategy from a distribution $\mathbf{p}$ (respectively, $\mathbf{q}$) over the rows (respectively, columns) then the expected loss (respectively, payoff) that he receives if the column (respectively, row) player chooses strategy j (i) is $M(\mathbf{p}, j) = \mathbf{E}_{i \in \mathbf{p}}[M(i, j)]$ (respectively, $M(i, \mathbf{q}) = \mathbf{E}_{j \in \mathbf{q}}[M(i, j)]$). According to John von Neumann's min-max theorem, if every player chooses a distribution (which is exactly a mixed strategy) in order to optimize their utility then they obtain the same value, also known as the game value:

$$\lambda^* = \min_{\mathbf{p}} \max_{j} M(\mathbf{p}, j) = \max_{\mathbf{q}} \min_{i} M(i, \mathbf{q}) \tag{2.9}$$

Suppose that we wish to solve the zero-sum game approximatelly, meaning to find row and column strategies ($\tilde{\mathbf{p}}$ and $\tilde{\mathbf{q}}$ respectively) such that, for an error parameter ϵ:

$$\min_i M(i, \tilde{\mathbf{q}}) \geq \lambda^* - \epsilon \tag{2.10}$$

and

$$\max_j M(\tilde{\mathbf{p}}, j) \leq \lambda^* + \epsilon \tag{2.11}$$

The problem already uses a payoff (loss) matrix so the Multiplicative Weights Update Method would seem a natural idea for a solution, as is originally proved in [18]. We correspond:

- Each decision to a pure strategy of the row player $i \in \{1, \ldots, n\}$.
- Each outcome to the pure column strategy $j^{(t)}$, returned by the ORACLE, which maximizes $M(\mathbf{p}^{(t)}, j^{(t)})$ in round t (the ORACLE operates in $O(n)$ time).
- Each coordinate of the penalty vector $m_i^{(t)} = M(i, j^{(t)})$, as $\rho = 1$ (note that this is exactly i's loss).

After $T = O(\log n/\epsilon^2)$ rounds, using the method's guarantee, we conclude that $\tilde{\mathbf{p}} = \mathbf{p}^{(\tilde{t})}$, where $\tilde{t}$ is the round with the minimum value of $M(\mathbf{p}^{(t)}, j^{(t)})$, is a distribution that corresponds to a value ϵ close to λ^* and so is the distribution $\tilde{\mathbf{q}}$ which assigns to each column j the probability $\frac{|\{t \in \{1,\ldots,T\} | j^{(t)} = j\}|}{T}$.

The general saddle point problem

The above problem can be considered as a special case of the following problem. Suppose we have two bounded convex sets X and Y in $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively and we are given a MEMBERSHIP ORACLE for each set, that is an algorithm which determines in polynomial time in the dimension of the space whether a point belongs in the set or not. We are also given a function $F : (X, Y) \to \mathbb{R}$ which is continuous and convex-concave, meaning that the function $F(\cdot, y) : X \to \mathbb{R}$ is convex $\forall y \in Y$ and $F(x, \cdot) : Y \to \mathbb{R}$ is concave $\forall x \in X$. The goal is to find a *saddle* point, i.e. (x^*, y^*) such that $\inf_{x \in X} \sup_{y \in Y} F(x, y) = \sup_{y \in Y} \inf_{x \in X} F(x, y)$ or at least approximate it. An ϵ-approximate saddle point would be a point (x^*, y^*) which satisfies the inequality $\sup_{y \in Y} F(x^*, y) \leq \inf_{x \in X} F(x, y^*) + \epsilon$.

The reason why this problem can be considered as a generalization of the previously described two player zero-sum game problem is the following: imagine the row player who needs to minimize his loss having to choose a strategy from a convex set X and the column player who needs to maximize his payoff having to choose a strategy from another convex set Y. For given strategies $x \in X$ and $y \in Y$, the loss (respectively, payoff) of the row (respectively, column) player would be given by the value of the function at this point, $F(x, y)$. Therefore, by the definition of saddle points, we can see that the game value we defined in zero-sum games, namely λ^*, is exaclty a saddle point.

This general problem can not be solved by direct application of the Multiplicative Weights Update Method because the sets X, Y are not necessarily bounded and the oracle we have assumed is weaker than the ones assumed previously. But an extension of the method, combined with sampling techniques will yield an efficient solution, whose running time depends on the width parameter as expected. The algorithm we are going to present is randomized and it returns an ϵ-approximate saddle point within an expected number of $O\Big(\frac{\rho^2(n+m)\ln\frac{R}{\epsilon}}{\epsilon^2}\Big)$ iterations, where R is a parameter we are going to introduce shortly. In each such iteration, the algorithm computes two approximate samples from *log-concave* distributions (for example, the normal distribution, the exponential distribution or the uniform distribution) which causes the overall running time to be $O^*\Big(\frac{\rho^2(n+m)^6\ln\frac{R}{\epsilon}}{\epsilon^2}\Big)$.

Definition 16. *A function is called* logarithmically concave *or* log-concave *for short, if its domain is a convex set and the logarithm of the function is a concave function.*

The above result only holds under an assumption, which is a very common assumption when dealing with Membership Oracles. We assume that we know two points $x^0 \in X$ and $y^0 \in Y$ and positive numbers r_X, r_Y, R_X, R_Y such that $B^m(x^0, r_X) \subseteq X \subseteq B^m(\mathbf{0}, R_X)$ and $B^n(y^0, r_Y) \subseteq Y \subseteq B^n(\mathbf{0}, R_Y)$ where $B^k(u^0, r_U)$ is the k-dimensional ball with center u_0 and radius r_U. The parameter $R = \max\{R_Y, R_X, \frac{1}{r_X}, \frac{1}{r_Y}\}$.

The algorithm is the following, introduced in [16]:

Randomized Fictitious Play

Initialization: Choose $x(0) \in X$ and $y(0) \in Y$ arbitrarily.

At every step $t = 1, \ldots, T$**, do:**

1. Choose $x' \in X$ and $y' \in Y$ independently with probability $\frac{p_{x'}(t)}{\|p(t)\|_1}$ and $\frac{q_{y'}(t)}{\|q(t)\|_1}$ respectively, where

$$p_{x'}(t) = e^{-\frac{\epsilon t F(x', y(t))}{2}}$$

$$q_{y'}(t) = e^{\frac{\epsilon t F(x(t), y')}{2}}$$

$$\|p(t)\|_1 = \int_{x' \in X} p_{x'}(t)dx'$$

$$\|q(t)\|_1 = \int_{y' \in Y} q_{y'}(t)dy'$$

2. Update the coordinates $x(t+1) = \frac{t}{t+1}x(t) + \frac{1}{t+1}x'$, $y(t+1) = \frac{t}{t+1}y(t) + \frac{1}{t+1}y'$

Return: $\Big(x(t), y(t)\Big)$

2.5.3 The Theory of Evolution: Coordination Games between genes

As impressive as it sounds, the Multiplicative Weights Update Method can be used to study how the frequency of appearence of a genotype (combination of two genes) changes from generation to generation in a sexual species, namely the population genetic dynamics. Each genotype ij has what is called its *fitness value* w_{ij}, which is the expected number of offspring this genotype produces by mating randomly and can be considered to be in $[1-s, 1+s]$, according to the weak selection assumption with strength s. These values form a matrix, a linear transformation of which is called the *differential fitness landscape*. It is proven in [11] that the population genetic dynamics is precisely the *multiplicative update dynamics* in a game where the players are the genes, all payoff functions are identical, and the payoff matrix is the differential fitness landscape. The multiplicative update dynamics imply that given a mixed strategy profile x^p of player p, this has to be updated in the following multiplicative way: The probability of player p playing strategy a is

$$\frac{x^p(a)(1+\eta q(a))}{1+\eta\bar{q}}$$

where $q(a)$ is the expected payoff of strategy a and $\bar{q}$ is the expected payoff to p (fixed for every player in a coordination game).

2.5.4 Network Congestion Control: TCP/IP Rule

The TCP/IP protocol, which is commonly used in communication networks today, uses a congestion control rule known as *multiplicative decrease, additive increase*. Even though it does not seem to fit perfectly into our framework, it does seem related. As one can easily prove, this rule converges to an allocation of $(1 \pm \epsilon)\frac{1}{n}$ of bandwidth to each user after $T \geq \log(\frac{n}{\epsilon})$ rounds.

Chapter 3

The Multiplicative Weights Update Method in Online Mechanism Design

In *Online Mechanism Design*, the players arrive dynamically and the mechanism has to make a decision irrevocably without any further information about the players who are "next in line". The Multiplicative Weights Update Method has many applications in Online Mechanism Design, as expected since it is a learning procedure applied in exaclty those situations where decisions are taken in each round. As we did in Chapter 2, we will make a selection of the applications that have had the most impact in the area. In this Chapter though, we will present them in a chronological order (for the most part), because most of the applications rely on previous work and we believe that this order will help illustrate the evolution of the ideas in a more natural way.

3.1 Online Routing

One of the first settings (in 1993), in which the idea of the method seemed to appear, was online routing problems.

In this setting ([3]), there is a network, modeled as a graph $G(V, E, u)$ with $|V| = n$ nodes and capacities $u(e)$ on its edges $e \in E$ which correspond to the bandwidth available in each connection. At any time, a request can be made. That is a request to make a connection of a certain bandwidth between two nodes in a certain period of time. Formally, a request i is a tuple $(s_i, t_i, r_i(\tau), T^s(i), T^f(i), \rho(i))$ where the first two coefficients are the source and the destination, r_i is the function of traffic rate required by the request over the period $\tau \in [T^s(i), T^f(i))$ (we will assume for simplicity that they are constant), and $\rho(i)$ is the profit of the request, which is assumed proportional to its rate and duration. The Mechanism then decides whether it will admit the request or reject it. In case it admits it, it also has to guarantee that the connection will be succesful, allocating the required bandwidth along a path P_i between the source and destination node, which is again chosen by the mechanism. The goal is of course to maximize the total profit, i.e. $\sum_{P_i \neq \emptyset} \rho(i)$ and it can be viewed as maximization of either revenue or social welfare.

Before we continue to present the algorithm, we should make some remarks. Firstly, this setting is a little peculiar because of the continuity of time. We can not use an algorithm that runs in rounds, since a request can be made at any time, so the algorithm must run each time we have a request. Moreover, there occurs the question of how we will evaluate such an algorithm. As it turns out, the algorithm can be proved competitive comparing the performance of the offline algorithm at any given interval and the performance of the online algorithm at the same interval, extended evenly by duration $2T = 2 \max \tau$.

In order to move on to the algorithm, we need to introduce one more quantity: the *relative load* λ. When we say that the competitive ratio of an online algorithm is λ with respect to relative load, it means that given any sequence of events (i.e. requests) that can be satisfied by the offline algorithm, the online algorithm would be able to satisfy them if the capacities of the edges were all increased by a factor λ. We define the relative load of an edge e, at time τ, right before the kth request by:

$$\lambda_e(\tau, k) = \sum_{i=1}^{k} -1 \sum_{e \in P_i} \frac{r_i(\tau)}{u(e)}$$

Obviously, in order for the capacity constraints to be satisfied, $\lambda_e(\tau, k) \leq 1$ needs to hold.

We are now ready to present the algorithm:

NEW CONNECTION ALGORITHM

Initialization: Fix a parameter $\mu = 2nT + 1$ and with all edges e at all times τ, associate the cost $c_e(\tau, j) = u(e)(\mu^{\lambda_e(\tau,j)} - 1)$.

If there is a path P_j from s to t such that:

$$\sum_{\tau} \frac{r_j(\tau)}{u(e)} c_e(\tau, j) \leq \rho(j) \tag{3.1}$$

Then route the connection on P_j and update the relative load of its edges:

$$\forall e \in P_j, \forall \tau \in [T^s(j), T^f(j)): \ \lambda_e(\tau, j+1) = \lambda_e(\tau, j) + \frac{r_j(\tau)}{u(e)} \tag{3.2}$$

Else block it.

The algorithm already reminds us of the Maximum Multicommodity Flow algorithm described in Section 2.4 of the previous chapter. We keep a cost for each edge at all times which is updated in a multiplicative way, increasing it for the period that it is used in order to avoid using it for future connections during that time. Of course, in this case, we can not let the bandwidth used in each edge exceed its capacity, and fortunatelly this is achieved by this algorithm if we make the assumption that $r_j(\tau) \leq \frac{\min_e\{u(e)\}}{\log \mu}$, which essentially translates to the fact that the rates of the requests have to be significantly smaller than the minimum capacity of the graph.

As one could possibly guess by now, the competitive ratio is a logarithmic function. In particular, the online algorithm is an $O(\log \mu) = O(\log nT)$ approximation of the offline. We will not prove this result since we will analyze later results in much more extent, but essentially, it occurs from the following obsevations: The sum of the costs is less than $\Theta(\log \mu)$ times the total profit of the online algorithm beacuse of the assumption we made and the sum of the offline profit caused by requests that we did not choose to route is less than the sum of the costs, because of the condition 3.1. It seems that the assumption helps us achieve feasibility and a good approximation ratio. What's more, it is proven that without this assumption a logarithmic approximation ratio can not be achieved by *any* online algorithm.

An important drawback of this algorithm is that the competitive ratio and the parameters of the algorithm itself depend on the time T which is the maximum duration of a request. This could be arbitrarily big with respect to the input n or even unknown. In order to fix this, the authors of [4] introduced one year later another algorithm which uses relative loads and the multiplicative technique and achives a $O(\log n)$ competitive ratio allowing $O(\log n)$ reroutes of requests, using a stability condition which is more easily checked, guarantees the desired approximation ratio and is weak enough so that it is not very often violated (at least not more than $O(\log n)$ times).

3.2 Combinatorial Auctions

Combinatorial Auctions (CAs) are a very interesting area of Mechanism Design, mainly because of their applications in real life. In a combinatorial auction, m non-identical items are auctioned simultaneously to n players, i.e. bidders. In the general case, each player has a valuation function v_j, which corresponds a non-negative value to each *bundle* of items, and the problem is to find a feasible allocation of the items, which maximizes the social welfare and impose paymens to the players. A *bundle* is a subset of the auctioned items. In the case of *single-minded* bidders the sets are of size 1. The classic case in which there are no more than a single copy of each item is called single-unit case. More interesting phenomena can be modeled using the multi-unit case, in which every item $i \in \{1, \ldots, m\}$ has B_i copies of itself, and in most settings, it suffices to allow B copies of each item.

As it has been described up until now, this problem seems to be just an instance of the VCG Mechanism given in Chapter 1. What we have ommited from the introduction, is the fact that VCG has certain drawbacks that can't be ignored.

The most important among them is that the mechanism requires choosing the outcome that maximizes the social welfare. In the combinatorial auction's case, the set of possible outcomes is exponential in size, (m^n even in the single-unit case) therefore computing the optimal outcome is impossible in polynomial time. When valuations are arbitrary even describing the valuation function to the mechanism is impossible in practice, since there are exponentially many points of the valuation function to be accessed (2^m bundles even in the single-unit case). Even the idea of approximation algorithms is not helpful in this scenario, because an approximation allocation rule does not guarantee truthfulness.

3.2.1 Online Truthful Multi-Unit Auctions for known bidders

The first breakthrough in this area, in 2003 ([5]), introduces a truthful mechanism for non single-minded bidders, auctioning B copies of each item with a good approximation ratio for both online and offline scenarios. The best result until then dealt with known single-minded bidders and achieved a $O(\sqrt{m})$ approximation ratio, which is the best possible in this case. The new result is based on an assumption, which however is very natural to the setting: the bidders desire no more than a single copy of each item. In fact, the assumption is that the demand of each player for each item is bounded by a lower and upper fraction of the copies. But it is easier to think of this percentage as $\frac{1}{B}$. Finally, we consider the items to have no multiplicity and allow the bundles to have fractional demands.

The way this algorithm overcomes the obstacle of choosing the optimal allocation is by transfering this decision on the players. Specifically, it announces the prices of each item and in each round asks the player himself (we can imagine it as a Demand Oracle) to choose the fraction of each item he wants to buy, in exchange for the same fraction of the item's price. Notice that it may still be **NP**-hard to choose the best bundle but this is now the players' problem!

The connection to the multiplicative weights update idea comes from the way the prices are updated and was inspired by the cost update rule in the online routing algorithms of the previous section. The price of each item $P^{(i)}$ is initialized at a certain value P_0 and it is multiplied by a parameter r per unit allocation. We denote by $l_j^{(i)}$ the fraction of item i that has already been sold when the jth player arrives. The general structure of the algorithm is the following:

General Multi-Unit CA Algorithm

Initialization: Fix the parameters P_0 and r. With all items $i \in \{1, \ldots, m\}$, associate the sold fraction $l_1^{(i)} = 0$.

For each bidder $j = 1, \ldots, n$

1. Update each item's i price: $P_j^{(i)} = P_0 r^{l_j^{(i)}}$.
2. Query j's Demand Oracle on the current prices and allocate the fractions it returns, $x_j^{(1)}, \ldots, x_j^{(m)}$.
3. Determine bidder's j total payment as $\sum_{i=1}^{m} x_j^{(i)} P_j^{(i)}$.
4. Update the sold fraction of every item: $l_{j+1}^{(i)} = l_j^{(i)} + x_j^{(i)}$.

The truthfulness of the Mechanism derives from the obsevation that if the price parameters do not depend on the valuations of the players then the player has no

incentive to lie since the choice of the bundles is his and because of the "take-it-or-leave-it" structure of the algorithm he cannot manipulate it. Of course, even then, the case of manipulation from coalitions remains possible. However, in the online algorithm these parameters depend on the maximum and minimum valuation $v_{\max}$ and $v_{\min}$ and they have to be considered known a priori for the mechanism to be truthful. More specifically, the parameters that satisfy certain assumptions that guarantee truthfulness are $P_0 = \frac{v_{\min}}{2n}$ and $r = B(2m\rho)^{1/(B-1)}$, where $\rho = \frac{v_{\max}}{v_{\min}}$. The approximation ratio is $O(B(m\rho)^{1/(B-1)})$, so it is close to being an optimal approximation for settings in which the range of the valuations is relatively small and $B = \Omega(\log m)$. Intuitively, large values of B give better performance because it allows the algorithm to run longer and to "learn" the right value for each item, depending on its "popularity".

The performance of the offline algorithm is better because we can choose the parameters to depend only on $v_{\max}$ and by modifying the general algorithm, using a "2nd" price (VCG-like) mechanism for the player with the maximum valuation, the mechanism remains truthful with no further assumptions. In addition, the approximation ratio is the same except that it does no depend on ρ. However, it is the online case we are mostly concerned with, since it fits the Multiplicative Weights Update Method better.

Using Randomness

Another work regarding the same problems was published at the same year ([2]) introducing a solution of *randomized* truthful mechanisms which have a good approximation ratio *in expectation*. This was a result of applying the general technique of the paper used to convert an online algorithm $\mathcal{A}$ to a truthful mechanism which uses $\mathcal{A}$ as a *black box*.

The approximation ratio is $O(\log \rho + \alpha)$, where α is the competitive ratio of $\mathcal{A}$ and ρ is the same valuation ratio as before. The randomization lies in the acceptance condition: Bidder j takes the set which maximizes the difference between his bid $b_j(s)$ and the minimum bid $c_j(s)$ (i.e. *the cost*) for which $\mathcal{A}$ accepts. But this is only if his bid is at least $c_j(s) + m$, where $m \in [0, \rho]$ is a random quantity, which follows the distribution $m = \begin{cases} 0 & \text{with probability } 1/2 \\ 2^i & \text{with probability } \frac{1}{2\log \rho} \end{cases}$.

Although the approximation ratio is good only *in expectation*, this algorithm is truthful even for coalitions (but not for players who merge their requests into a single one). There are, as usual, some properties that the original online algorithm has to satisfy but they are natural to almost all common settings. Algorithm $\mathcal{A}$, however, still has to provide a good approximation guarantee and the best known is the one from the previous section. More randomization algorithms for these problems appeared later, which we will analyze in subsequent sections.

3.2.2 Truthful Multi-Unit Auctions for single-minded bidders

In the previous section, one of the problems we had to face was that additional information were needed about the range of the bidders' valuations and the approximation ratio depended on them. If we allow single-minded bidders then there exists a truthful greedy combinatorial auctions algorithm which does not need any additional information on the bidders and gives a better approximation result that depends only on m and B. This algorithm, introduced among other results in [7], has approximation ratio $O(m^{1/B})$, using similar multiplication rules.

3.2.3 Online Truthful Multi-Unit Auctions

Finally, in an effort to rid the algorithms of previous assumptions and find an online algorithm that is efficient even for $B = o(\log m)$ and holds for the case of non single-minded bidders, the authors of [23], introduced two new algorithms, one deterministic and one randomized. These are the algorithms that we are going to analyze in extent, mainly because they yield the best results.

The initial idea is very similar to the one from [5], since it is the multiplicate weights update idea applied on the prices of the items and calls each player's i Demand Oracle D_i instead of computing the utility maximization bundle. The new idea, is that the mechanism uses the concept of oblivious randomized rounding, in the sense that there is a probability that each bundle returned by the Demand Oracle is not assigned to the corresponding player. More specifically, the first (deterministic) algorithm, does not return a feasible solution, meaning that it sells more than B copies of each item but has constant approximation ratio. The overselling is fixed in the second (randomized) algorithm.

The algorithm is given two parameters:

- $0 \leq L \leq u_{\max}$
- $\mu \geq 1$

such that there is at most one bidder whose valuation exceeds μL.

OVERSELLING MULTIPLICATIVE PRICE UPDATE ALGORITHM
Initialization: For each item $e \in U$: $p_e^1 = p_0 = \frac{L}{4Bm}$.
For each bidder $i = 1, \ldots, n$

1. Give bidder i bundle
$$S_i = D_i(U, p^i) \tag{3.3}$$
2. Update all sold items $e \in S_i$: $p_e^{i+1} = p_e^i \cdot 2^{1/B}$

By overselling, this algorithm achieves a constant factor approximation, since, as we stated earlier, letting the algorithm sell more gives us more information about the players' valuation of the items and helps us find the "right" price for them. Technically, the first parameter of the DEMAND ORACLE in 3.3 may differ between rounds (it could be any $U_i \subseteq U$). The analysis holds for the most part for any sequence $U_1, \ldots, U_n$ but the last lemma regarding the approximation ratio uses *unrestricted* oracles, meaning that it has to be $U_1 = \ldots = U_n = U$.

Lemma 17. *The allocation S maps at most sB copies of each item to the bidders, where $s = \log(4\mu Bm) + \frac{2}{B}$ (s is called the* overselling factor*).*

This is an easy result to prove, given the assumption that there is only one who could buy a copy after its price is higher than μL. We note here that μ can be erased from the factor if we assume that the bidders come in a random order (and not arbitrarily-in an adversarial way).

Regarding the approximation ratio:

Lemma 18. *The following two inequalities bounding the total valuation hold:*

- $v(S) \geq B \sum_{e \in U} p_e^* - Bmp_0$.
- $v(S) \geq v(OPT) - B \sum_{e \in U} p_e^*$.

The first inequality follows from the assumption that the utilities of the bidders are not negative and the second from the fact that every bundle T_i assigned by the optimal solution would have less or equal utility to the chosen bundle S_i at the time.

Summarizing the results:

Theorem 19. *The Overselling Algorithm produces an infeasible allocation that assigns at most $B \log(4\mu Bm) + 2$ copies to the bidders and achieves social welfare $v(S) \geq \frac{1}{4} v(OPT)$.*

Combining the above lemmata and the inequality $Bmp_0 = \frac{L}{4} \leq \frac{v(OPT)}{4}$, it is easy to prove the theorem.

Before we continue to fix the overselling problem attempting to preserve the constant approximation ratio, we will give the main idea behind it. If we want to preserve the ratio, we have to still "use" some of the bidders to find out how popular each item is. But this time, we will not sell it, since we are limited by the number of copies B. To manage to do both, we introduce a constant probability q. With probability q we actually sell the item and with probability $1-q$ we don't. We "sacrifice" it to gain information. It is already obvious that it is very important that this probability is chosen correctly.

MPU Algorithm with Oblivious Randomized Rounding
Initialization: For each item $e \in U$: $p_e^1 = p_0 = \frac{L}{4Bm}$, $b_e^1 = B$.
For each bidder $i = 1, \ldots, n$

1. Set
$$S_i = D_i(U_i, p^i) \text{ ,for } U_i = \{e \in U \mid b_e^i > 0\} \tag{3.4}$$
2. Update each item $e \in S_i$: $p_e^{i+1} = p_e^i \cdot 2^{1/B}$.
3. With probability q set $R_i = S_i$, else set $R_i = \emptyset$.
4. Update each item $e \in R_i$: $b_e^{i+1} = b_e^i - 1$.

Step 4 and the definition of U_i in 3.4 guarantee that the algorithm does not oversell. Let us now discuss the approximation ratio. Since this is now a randomized algorithm, the competitive ratio is the maximum ratio of the optimal offline algorithm to the expectation of the online algorithm, where the latter is denoted $\mathbf{E}[v(R)]$.

Theorem 20. *If q is sufficiently small such that for any $i \in [n]$ and any bundle $T \subseteq U$,*
$$\boldsymbol{E}[v_i(T \cap U)] \geq \frac{1}{2} v_i(T) \tag{3.5}$$
then $\boldsymbol{E}[v(S)] \geq \frac{1}{8} v(OPT)$ and $\boldsymbol{E}[v(R)] \geq \frac{q}{8} v(OPT)$. Therefore, we have an $O(\frac{1}{q})$ approximation of the optimal social welfare.

The proof uses similar arguments as the previous, except from the obvious implication that $\mathbf{E}[v(S)] \geq \frac{1}{8} v(OPT) \Rightarrow \mathbf{E}[v(R)] \geq \frac{q}{8} v(OPT)$ which is due to the linearity of expectation.

The conclusion of this theorem is that we need to find a way to choose probability q to be as large as possible but to satisfy 3.5 at the same time. This is contradictory because in order for the last inequality to hold, U_i should be as large as possible, therefore we should not sell items very often, meaning that q has to be small. Imagine that for $q = 0$ we sell no items so $\mathbf{E}[v_i(T \cap U_i)] = \mathbf{E}[v_i(T \cap U)] = \mathbf{E}[v_i(T)] = v_i(T) \geq \frac{1}{2} v_i(T)$.

There is no known way to decide q at every setting. In the case of CAs with bundles with maximum cardinality d, it is proven that condition 3.5 holds for $\frac{1}{q} = O(d^{1/B} \log m)$.

Finally, we note that the randomized mechanism is universally truthful since every determinstc mechanism it uses is truthful and probability q does not depend on the bids.

3.3 The Resource Allocation Framework on inputs with stochastic properties

Up to this point, we have compared the performance of the online algorithms to the best offline solution in the worst case of the input. This approach has known inapproximability bounds for many interesting problems and it also may in fact be too strict. If instead we assumed certain stochastic properties of the input, for example that the input comes from a known or unknown distribution, we would possibly be more just to the performance of the algorithms. N. Devanur, K. Jain et al.([13, 14, 8]) explored this direction on a class of problems that can be considered as a generalization of both CAs and many advertizing problems, called the *Resource Allocation Framework* while at the same time introducing fast approximation algorithms for mixed packing-covering LPs using the Multiplicative Weights Update Method. This particular application of the method is the subject of this subsection.

Firstly, we will desribe the problems we are going to use. A very well known canonical problem from the Resource Allocation Framework is the Adwords problem, where there are n advertizers and m queries (we can imagine them as keywords in a search engine) arriving online. Each advertizer i has a bid for keyword j, denoted b_{ij}, and a total budget B_i. The goal is to allocate the keywords to the advertizers (bidders) so that no one exceeds their budget and the revenue is maximized. A relaxation of the LP representing this problem is the following:

$$\begin{aligned}
\max \quad & \sum_{i,j} b_{ij}x_{ij} & \\
& \sum_{j} b_{ij}x_{ij} \leq B_i : \ \forall i \in \{1,\dots,n\} \\
& \sum_{i} x_{ij} \leq 1 \quad : \ \forall j \in \{1,\dots,m\} \\
& x_{ij} \geq 0 \quad : \ \forall i \in \{1,\dots,n\}, \forall j \in \{1,\dots,m\}
\end{aligned}$$

Given the Adwords problem, we can understand more easily the Resource Allocation Framework. In this framework, there are requests arriving online and the algorithm has to decide the way to serve these requests by choosing among some options $k \in K$ (may be exponential on m and n) which define the amount of the resources the algorithm will provide. By serving a request j with option k of course, there is an amount a_{ijk} of resource i that is consumed and also a profit w_{jk}. Denoting by C_i the total capacity of resource i, the corresponding LP for the Resource Allocation Framework is the following:

$$\begin{aligned} \max \quad & \sum_{j,k} w_{jk} x_{jk} \\ & \sum_{j,k} a_{ijk} x_{jk} \leq C_i : \ \forall i \in \{1, \ldots, n\} \\ & \sum_{k} x_{jk} \leq 1 \quad : \ \forall j \in \{1, \ldots, m\} \\ & x_{jk} \geq 0 \quad : \ \forall j \in \{1, \ldots, m\}, \forall k \in \{1, \ldots, K\} \end{aligned}$$

Finally, the mixed packing-covering problem consists of deciding whether the following mixed packing and covering system of inequalities has a feasible solution.

$$\begin{aligned} & \sum_{j,k} a_{ijk} x_{jk} \leq c_i : \ \forall i \in \{1, \ldots, n\} \\ & \sum_{j,k} b_{ijk} x_{jk} \geq d_i : \ \forall i \in \{1, \ldots, n\} \\ & \sum_{k} x_{jk} \leq 1 \quad : \ \forall j \in \{1, \ldots, m\} \\ & x_{jk} \geq 0 \quad : \ \forall j \in \{1, \ldots, m\}, \forall k \in \{1, \ldots, K\} \end{aligned}$$

The Gap Version of this problem is the problem of distinguishing with probability at least $1 - \delta$ between the cases where there is a feasible solution to the LP and where there is no feasible solution even if the packing and the covering constraints are relaxed by a multiplicative factor of $(1+\epsilon)$ and $(1-\epsilon)$ respectively.

Since the problems are now clear, we will describe the results which were based on an application of the Multiplicative Weights Update Method and then we will explain how it was essentially applied.

One of the results is that problems that belong to the general class of the Resource Allocation Framework (such as the Adwords problem) have an algorithm that yields an $1-\epsilon$ approximate solution given that there is a bound to the largest bid to budget ratio of the form $\gamma = O\Big(\frac{\epsilon^2}{\log(n/\epsilon)}\Big)$ and that the inputs are either drawn from an unknown independent identical distribution (*i.i.d.*) or follow the adversarial stochastic model. The latter is a model introduced in that same series of papers, in which an adversary picks a different distribution on each round but is constrained not to pick "too bad" distributions, meaning that the algorithm is given in each round T an OPT_T and the adversary is constrained to pick a distribution with $\mathbf{E}[OPT] \geq OPT_T$. Very similarly, in cases where γ is unbounded, a greedy algorithm gives an $1 - 1/e$ approximation.

The last result and the first to analyze is that there is an algorithm that solves the Gap Version of the mixed packing-covering problem. Moreover, this algorithm uses an ORACLE like the one introduced for the PST Framework, which given an input vector v and a request j returns an allocation x_j from the unary polytope $P_j = \{x_j \in \mathbb{R}^K \mid \sum_k x_{jk} \leq 1\}$ that maximizes the product $v \cdot x_j$. The ORACLE calls the algorithm makes are only $O\left(\frac{\gamma m \log(n/\delta)}{\epsilon^2}\right)$, where $\gamma = \max\left\{\left\{\frac{a_{ijk}}{c_i}\right\}_{i,j,k}, \left\{\frac{b_{ijk}}{d_i}\right\}_{i,j,k}\right\}$ is the parameter we defined for the Adwords problem generalized to the setting of the mixed packing-covering problem. Notice how this is an improvement of the PST Framework's results, since the number of calls depend linearly on γ and m, instead of quadratically.

We are now going to explain the technique that was used to prove these results and how the Multiplicative Weights Update Method was an important part of it. We are going to use a simpler problem to help focus on the technique itself. In this setting, each item i arrives online - drawn from an i.i.d. - and we either pay for an item its cost c_i or we put it in a bag which has capacity G. The goal is to minimize the total cost paid and we are going to assume again for simplicity that the optimal cost is G. The LP of the problem follows:

$$\begin{aligned} \min \quad & \sum_i c_i y_i \\ & \sum_i x_i \leq G \\ & x_i + y_i \geq 1 : \ \forall i \in \{1, \ldots, n\} \\ & x_i, y_i \geq 0 \quad : \ \forall i \in \{1, \ldots, n\} \end{aligned}$$

The solution comes in two stages. First, we assume that the distribution on the inputs and the optimal offline solution is known and we develop an algorithm that achieves the desired competitive ratio but also its analysis has certain properties. It is these properties of the analysis that lead to the second stage and to developing an algorithm that is unaware of the knowledge of the previous one but still achieves the promised ratio.

If we knew the distribution on the input in our problem, then we could choose a threshold a to help us decide whether the item will be put in the bag (if $c_i \geq a$) or we are going to buy it (if $c_i < a$). In this way, a would be chosen so that the probability $p = \mathbf{Pr}[c_i \geq a] = \frac{G}{m}$. We define the random variable X_t which is 1 iff in step t the item is put in the bag and the random variable Y_t which is c_i iff it is bought. In the dual case, the random variables are 0. Their means are the same quantity, namely $\frac{G}{m}$. If we also define the sums $S_t = \sum_{i=1}^{t} X_i$ and $V_t = \sum_{i=1}^{t} Y_i$ then the probability of exceeding the capacity of the bag is $\mathbf{Pr}[S_m \geq G(1+\epsilon)]$ and the probability of returning a sub-optimal solution is $\mathbf{Pr}[V_m \geq G(1+\epsilon)]$. These probabilities together are the probability of failure of the algorithm, and the analysis basically bounds this using Chernoff bounds as used in the multiplicative

weights update idea in [30].

It is easy to prove that the (scaled) failure probability after t steps, denoted by FP^t is given by $FP_X^t + FP_Y^t = \Pi_{i=1}^t(1+\epsilon X_i) + \Pi_{i=1}^t(1+\epsilon Y_i)$ and the expected failure probability after $t+1$ steps is less than $FP_X^t(1+\epsilon\mathbf{E}[X_i]) + FP_Y^t(1+\epsilon\mathbf{E}[Y_i]) = (FP_X^t + FP_Y^t)(1+\epsilon\frac{G}{m})$, by our observation about the means of the random variables. Notice that this expected value grows once again in a multiplicative way as the algorithm proceeds and it is this quantity that we need to minimize. Therefore, we need to choose whether or not to put the item in the bag depending on whether the quantity FP_X^t is bigger than $c_i FP_Y^t$ or not. As a result, the desired "unaware" algorithm using the same threshold idea, but this threshold can vary between the different steps of the algorithm and it is updated in a multiplicative way. To summarize, the root of the Multiplicative Weights Update Method in this technique lies in the fact that the performance of the algorithm again depends on a random variable (the expectation of which is the potential function), and by using a pessimistic estimator of this variable's changes over the course of the algorithm, we can achieve the desired tradeoff of the method (since minimizing this pessimistic estimator does not require any further knowledge of the distribution or the optimal solution's value).

In the CAs setting, the algorithm is very naturally translated into a posted-prices algorithm we have already discussed and yields a guarantee of $1 - O(\epsilon)$ for $\gamma = \frac{1}{\min_i c_i} = O\Big(\frac{\epsilon^2}{\log(n/\epsilon)}\Big)$ and given a desired social welfare W^* can also avoid updating the prices at each step maintaining the guarantee for an i.i.d. input.

3.4 Auctions with budget constraints

In a lot of marketing settings, the auction model assumes bidders with budget constraints. It is a very natural assumption to make, since in some very common auctions, such as sponsored search auctions, the values are too big for the bidders' liquid assets. In these models, the objective function that needs to be maximized can not be one of the common ones, such as social welfare or revenue, as it has been proven. For example, there is an impossibility result that states that there is no truthful mechanism that achieves more than n-approximation of the optimal social welfare, where n is the number of bidders, even if the budgets are known to the auctioneer. Another very popular measure of efficiency for budget constrained auctions, is *Pareto efficiency*. An outcome is considered Pareto efficient if there is no other outcome for which at least one bidder is better-off and all the others are not worse-off. Unfortunatelly, this measure is not quantifiable and has already given some impossibility results. A new measure proposed in [15] is *liquid welfare* which is, intuitively, the minimum between how much a player is willing to pay for an outcome and how much he is able to pay (i.e. his budget). More formally:

Definition 21. *Liquid Welfare of an outcome x, denoted $\bar{W}(x)$, is the sum over all bidders of the minimum value between their valuation for the outcome and their budget, i.e. $\bar{W}(x) = \sum_{i=1}^{n} \bar{v}_i(x_i)$, where $\bar{v}_i(x_i) = \min\{v_i(x_i), B_i\}$.*

Based on the Multiplicative Weights Update Method and the technique presented before from [5], the authors of [15] gave an algorithm that is a $O(\log n)$-approximation for the liquid welfare in the case where the bidders are submodular, meaning that their valuation functions are concave, and their budgets are private. We must note that in this setting there is only one divisible good, therefore the valuation functions are $v_i(x_i) : [0,1] \to \mathbb{R}_+$. The result can be translated into the case of m identical indivisible goods (which is closer to the combinatorial auctions' usual setting) adjusting the approximation ratio to $O(\log m)$.

Firstly, we are going to describe a subroutine used in the algorithm, the *Sell-without-r* subroutine: In this subroutine, we are given a player r and his modified valuation $\bar{v}_i(\frac{1}{2})$ for half the good and we need to decide on the prices in which we are going to sell to each bidder arriving, except bidder r.

1. Divide the remaining segment $\left[0, \frac{1}{2}\right]$ of the good into $k = 8 \log n$ equal parts (each of size $\frac{1}{2k}$).
2. Associate with each part i a price per unit $p_i = \frac{2^i}{8} \bar{v}_i(\frac{1}{2})$.
3. Each bidder chooses the available segment that maximizes his utility, given of course that the price he pays does not exceed his budget. Formally, player i gets $x_i = \text{argmax}\{v_i(x_i) - \int_{z_i}^{z_i+x_i} p(t)dt\}$, where $z_i = \sum_{j<i} x_j$ and the payment

function $p(t)$ has value p_i for the interval $t \in \left[\frac{1}{2k}(i-1), \frac{1}{2k}i\right]$. We denote the payment of bidder i by $\pi_i = \int_{z_i}^{z_i+x_i} p(t)dt$.

The mechanism defines the first two players, r_1 and r_2 with the maximum $\bar{v}_i(\frac{1}{2})$. It then runs the subroutines *Sell-without-*r_1 returning outcome $(x^{(1)}, \pi^{(1)})$ and *Sell-without-*r_2 returning outcome $(x^{(2)}, \pi^{(2)})$. For every bidder except r_1 return the allocation and payments implied by outcome (1). Finally, for bidder r_1 choose either to allocate $x_1^{(2)}$ with the corresponding payment or half the good with payment $2\bar{v}_i(\frac{1}{2})$ depending on the utility of those outcomes for bidder r_1.

It is easy to see with exaustive analysis that the mechanism described above is indeed truthful and also that the final allocation is feasible. More importantly, by case analysis it is proven to be a $O(\log n)$-approximation for the liquid welfare, when it is one of the hardest impossibility cases with respect to Pareto efficiency. In a slightly more general setting, where the bidders are subadditive, the result can be extended to a $O(\log^2 n)$-approximation.

Chapter 4

The Multiplicative Weights Update Method in Regret Theory

As it was described in Chapter 1, Algorithmic Game Theory models society as a game where players interact with each other strategically, aiming to maximize their personal gain. A very natural question that occurs is whether this game reaches a point of balance, where players do not need to change their strategy anymore, referred to as an equilibrium. And if it does reach such a point, how long does it take to get there? As one can imagine, players reach this point by learning from the consequences of their strategies and re-evaluating their choices. Therefore, it doesn't come as a surprise that players tend to follow decision-making algorithms which are special cases of the Multiplicative Weights Update Method. In order to demonstrate the relation between those, we are going to introduce the concept of *regret*.

4.1 The Model

We consider again a game which is played in T rounds. In this game, the player is given a set of strategies S, where $|S| = n$. In each round t, the player picks a mixed strategy, that is he decides on a probability distribution $\mathbf{p}^{(t)}$ over his strategies. Afterwards, an adversary picks a cost vector over his strategies $\mathbf{c}^{(t)}$, where $|c_i^{(t)}| \leq 1$, for each $i \in \{1, \ldots, n\}$. Finally, the cost incurred in each round is $\sum_{t=1}^{T} \mathbf{p}^{(t)}\mathbf{c}^{(t)}$.

Notice that this is a single player game against an unknown adversary so at a first glance it doesn't seem to model the society very well. Fortunatelly, this is not accurate, since we can imagine the cost vector of the adversary to be a function of all the other players in the game. We will discuss this in more detail later in the chapter.

We have now set the rules of the game, but what we mainly want to know is what the goal should be. Of course, it is to minimize the total cost at the end of the rounds. But what should we compeete against? The first benchmark that comes to mind is the minimum total cost, given by the best decision sequence in hindsight, which would be possible if we knew the cost vector up front and decided on the probability distribution which gives 1 to the action s_i with the $\min c_i^{(t)}$ in each round. This quantity seems to be too much to ask, since we know nothing about the cost vector before we make each round's decision. An example which demonstrates this fact is the following simple setting:

Imagine that there are only two strategies s_1 and s_2. The adversary decides on the costs as follows: if $p_1^{(t)} \geq 1/2$ then the cost vector is $\mathbf{c}^{(t)} = (1, 0)$ and $\mathbf{c}^{(t)} = (0, 1)$ otherwise. In this case, the best decision sequence in hindsight has total cost 0 while the expected total cost of our player is $T/2$.

This seems too unfair a benchmark, therefore we must start limiting the goal we compare to in order to get better and more logical results. To this direction, we fix the best action in hindsight, that is the action s_{i^*} with the $\min \sum_{t=1}^{T} c_{i^*}^{(t)}$. This is the point where we should introduce *regret*.

Definition 22. *The* (time-averaged) external regret *of the probability distribution sequence* $\mathbf{p}^{(1)}, \ldots, \mathbf{p}^{(T)}$ *with respect to strategy* s_{i^*} *is*

$$\frac{1}{T}\Big|\sum_{t=1}^{T} \mathbf{p}^{(t)}\mathbf{c}^{(t)} - \sum_{t=1}^{T} c_{i^*}^{(t)}\Big|$$

From now on we will refer to this quantity simply as *regret*.

We seek online decision-making algorithms that satisfy the so called *no-regret* property.

Definition 23. *An online decision-making algorithm is called* no-regret *if for every adversary, the expected regret with respect to every fixed strategy in S is $o(1)$ while $T \to \infty$.*

We have to note that in this chapter we assume that the player learns the whole cost vector after each round and that the adversary is *adaptive*, that is he knows the mixed strategies decided by the player in all the previous rounds. There is also a special case where the adversary's knowledge is limited to the mixed strategy decided by the player in the current round. In this case, we call the adversary *oblivious*. Another variation is to allow the player to learn only the realized cost of the current round instead of the cost vector, and this is called the *bandit model*. Similar results like the ones we are going to discuss later on exist for these variations as well.

Impossibility Results

Firstly, we can not expect to find a deterministic no-regret algorithm. To understand this one should simply think of an example where at each round, the algorithm commits to a single strategy. The obvious move for the adversary is to set the cost of this strategy at 1, and the cost of every other at 0. Then, the total cost of the algorithm is T while the total cost of the best strategy in hindsight is at most $\frac{T}{n}$ (otherwise, the sum of the costs that every strategy would have gathered at the end of the rounds would be more that T). Thus, the algorithm would have regret $\frac{n-1}{n}T$, which is not $o(1)$ as $T \to \infty$.

We are now convinced that no deterministic algorithm would work. But there are limitations to randomized algorithms, too.

There is no randomized algorithm that vanishes external regret faster than $\Theta(\sqrt{(\ln n)/T})$. For $n = 2$ actions this translates to the following: there is no randomized algorithm that vanishes regret faster than $\Theta(1/\sqrt{T})$. We will show the latter via an example, which is essentially the reason that the general case (for n actions) holds.

Consider an adversary that chooses uniformly at random between the cost vectors $(1, 0)$ and $(0, 1)$, independently on each day t. In this case, no matter what the decision-making algorithm chooses on each day, the total expected cost at the end of T rounds is $T/2$. However, in hindsight, one of the two actions has total cost $\frac{T}{2} - \Theta(\sqrt{T})$ with constant probability (where the $\Theta(\sqrt{T})$ emerges from the deviation). This implies that there is a distribution over all the oblivious

adversaries such that every algorithm (randomized or not) has expected regret $\Omega(1/\sqrt{T})$, where the expectation is taken over the algorithm's coin flips and the choice of the adversary (which are 2^T in total). Therefore, for every algorithm, there exists an adversary that causes expected regret at least $\Omega(1/\sqrt{T})$ after T rounds.

4.2 Regret and Coarse-Correlated Equilibria

4.2.1 No-Regret algorithms

Even though the previously stated impossibility results may seem discouraging, the good news is that no-regret algorithms exist. They in fact depict a very natural human learning behaviour and what's more, they match those impossibility results.

Theorem 24. *There exist very simple no-regret algorithms, which guarantee expected regret $O(\sqrt{(\ln n)/T})$ with respect to every fixed action.*
As a result, there exists an online decision-making algorithm that for every $\varepsilon > 0$ has expected regret at most ε with respect to every fixed action, after $O(\ln n/\varepsilon^2)$ rounds.

A very popular no-regret algorithm is *Hedge* which fits clearly the Multiplicative Weights Update Method. Formally, *Hedge* is given an action set A and it is the following algorithm:

HEDGE ALGORITHM
Initialization: Fix a $\varepsilon \leq \frac{1}{2}$. With each action a, associate the weight $w^{(1)}(a) = 1$.
At every step $t = 1, \ldots, T$**:**

1. Choose action a with probability $p^{(t)}(a) = w^{(t)}(a)/\Phi^{(t)}$, where $\Phi^{(t)} = \sum_{a \in A} w^{(t)}(a)$.
2. Based on the cost vector returned by the adversary $\mathbf{c}^{(t)}$, adjust the weight of each action a as follows:

$$w^{(t+1)}(a) = w^{(t)}(a)(1-\varepsilon)^{c^{(t)}(a)}$$

The analysis of the previous algorithm gives the optimal regret bound presented in Theorem 24 and is very easy to derive as it follows step by step the analysis of the Multiplicative Weights Update Method from Chapter 2. It is important to note that, as in the original method, the parameter ε is determined based on the total number of rounds T. The good news is that if T is not known a priori, the bounds still stand if we set each day t a different $\varepsilon = \sqrt{(\ln n)/\bar{t}}$, where $\bar{t}$ is the smallest power of 2 that is larger than t.

4.2.2 No-Regret Dynamics

It is now time to pass from single-player to multi-player. The new setting is no-regret dynamics and the rules are the following:

In each day $t \in \{1, \ldots, T\}$:

- Each player i simultaniously and independently chooses a mixed strategy $\mathbf{p}_i^{(t)}$ using a no-regret algorithm (not necessarily *Hedge*).
- At the end of each day, player i receives a cost vector $\mathbf{c}_i^{(t)}$ where $c^{(t)}(s_i)$ is the expected cost of strategy s_i and it is computed by $\mathbf{E}_{\mathbf{s}_{-i} \sim \sigma_{-i}}\Big[C_i(s_i, \mathbf{s}_{-i})\Big]$.

The time-averaged history of joint play under no-regret dynamics converges to the set of coarse correlated equilibrium (CCE), which is the biggest set in our hierarchy of equilibria. This forges a fundamental connection between a static equilibrium concept and outcomes generated by natural learning dynamics. It means that these equilibria, of the more general and less restrictive form, are essentially easy to find via natural learning procedures. More formally:

Suppose after T rounds of no-regret dynamics, every player has regret at most ε for each of his strategies. Let $\sigma^{(t)} = \Pi_{i=1}^{k} \mathbf{p}_i^{(t)}$ denote the outcome distribution at time t and $\sigma = \frac{1}{T} \sum_{t=1}^{T} \sigma^{(t)}$ the time-averaged history of these distributions. Then σ is a ε-approximate equilibrium in the sense that

$$\mathbf{E}_{\mathbf{s} \sim \sigma}\Big[C_i(\mathbf{s})\Big] \leq \mathbf{E}_{\mathbf{s}_{-i} \sim \sigma_{-i}}\Big[C_i(s_i', \mathbf{s}_{-i})\Big] + \varepsilon$$

for every player i and unilateral deviation s_i'.

The proof uses only the definition of no-regret algorithms and coarse-correlated equilibria.

4.3 Swap-Regret and Correlated Equilibria

After proving that the biggest set of equilibria is tractable (a set of players can actually approximately reach such a point in not too many rounds of playing the game), we are moving to the next biggest set, the set of correlated equilibria, and ask the same question. Can regret theory still help us in this query?

Firstly, we introduce correlated equilibria.

Definition 25. *A correlated equilibrium of a game is a distribution σ over the outcomes such that, for every player i with strategy set S_i, and every switching function $\delta : S_i \to S_i$*

$$\mathbf{E}_{\mathbf{s}\sim\sigma}\Big[C_i(\mathbf{s})\Big] \leq \mathbf{E}_{\mathbf{s}_{-i}\sim\sigma_{-i}}\Big[C_i(\delta(s_i), \mathbf{s}_{-i})\Big].$$

An ε-approximate correlated equilibrium is accordingly the one which satisfies the latter inequality with an additive error ε. Notice that if we set δ to be a constant function, that is $\delta(s_i) = s_i'$ for all $s_i \in S_i$, then we get a coarse-correlated equilibrium. Correlated equilibria are harder to reach because they give us more to compare to.

In order to search for algorithms that the players can follow to reach such an equilibrium, we introduce a new concept of regret, *swap-regret* and the corresponding *no-swap-regret* algorithms.

Definition 26. *An online decision-making algorithm is called* no-swap-regret *if for every adversary, the expected swap-regret*

$$\frac{1}{T}\Big[\sum_{t=1}^{T} c(a^{(t)})^{(t)} - \sum_{t=1}^{T} c(\delta(a^{(t)}))^{(t)}\Big]$$

with respect to every switching function $\delta : A \to A$ is $o(1)$ as $T \to \infty$.

Suppose after T rounds of no-swap-regret dynamics, every player has swap-regret at most ε for each of his strategies. Let $\sigma^{(t)} = \Pi_{i=1}^{k} \mathbf{p}_i^{(t)}$ denote the outcome distribution at time t and $\sigma = \frac{1}{T}\sum_{t=1}^{T} \sigma^{(t)}$ the time-averaged history of these distributions. Then σ is a ε-approximate equilibrium in the sense that

$$\mathbf{E}_{\mathbf{s}\sim\sigma}\Big[C_i(\mathbf{s})\Big] \leq \mathbf{E}_{\mathbf{s}_{-i}\sim\sigma_{-i}}\Big[C_i(\delta(s_i), \mathbf{s}_{-i})\Big] + \varepsilon$$

for every player i and switching function $\delta : S_i \to S_i$.
The proof is very simple, using again only the definitions.

What remains is to see whether no-swap-regret algorithms exist or not. If they do and they are easy to follow, we can conclude that games do reach correlated equilibria approximately. Fortunatelly, there is a proof given in [6] that if no-regret algorithms exists (which they do), then no-swap-regret algorithms also exist. Therefore, the answer in the initial question is positive.

The proof is a black box reduction, using no-regret algorithms as the black boxes. Let n denote the number of actions. Then we assume $M_1, \ldots, M_n$ to be no-regret algorithms. As we have already discussed, this means that they receive cost vectors as input and have probability distributions as output. The main algorithm (the no-swap-regret algorithm) works as follows:

In each day $t = 1, \ldots, T$:

- Receive distributions $\mathbf{q}_i^{(t)}$, $i = 1, \ldots, n$ over actions from the corresponding algorithms M_i.
- Compute a distribution $\mathbf{p}^{(t)}$ (we will choose this later on).
- Receive cost vector $\mathbf{c}^{(t)}$.
- Return to each algorithm M_i the cost vector $p^{(t)}(i) \cdot \mathbf{c}^{(t)}$.

The time-averaged expected cost of the main algorithm is

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{n} p^{(t)}(i) \cdot c^{(t)}(i).$$

We need to prove that the latter quantity is at most $\frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{n} p^{(t)}(i) \cdot c^{(t)}(\delta(i))$ plus a term that is $o(1)$ when $T \to \infty$, for every switching function δ.

We know that M_i s are no-regret algorithms. Therefore, if R_i is their regret, then for every fixed action k:

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{j=1}^{n} q_i^{(t)}(j) p^{(t)}(i) \cdot c^{(t)}(j) \leq \frac{1}{T} \sum_{t=1}^{T} p^{(t)}(i) \cdot c^{(t)}(k) + R_i.$$

If we instantiate $k = \delta(i)$ and add all those inequalities for all $i = 1, \ldots, n$ we get:

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{j=1}^{n} \sum_{i=1}^{n} q_i^{(t)}(j) p^{(t)}(i) \cdot c^{(t)}(j) \leq \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{n} p^{(t)}(i) \cdot c^{(t)}(k) + \sum_{i=1}^{n} R_i.$$

If we choose $p^{(t)}(i) = \sum_{j=1}^{n} q_j^{(t)}(i) p^{(t)}(i)$ then the above inequality is exactly what we need to prove, since the last quantity tends to 0 when $T \to \infty$. Can we define $\mathbf{p}^{(t)}$ like this? Imagine a Markov chain where the set of states is A and we move from j to i with probability $q_j^{(t)}(i)$. The definition of the stationary distribution of this Markov chain is the equality we need to show for $\mathbf{p}^{(t)}$. But we know that at least one stationary distribution exists and we can find it via eigenvector computation in polynomial time. Therefore, the desired $\mathbf{p}^{(t)}$ distribution exists and can be computed easily. □

Bibliography

[1] S. Arora, E. Hazan, and S. Kale. The Multiplicative Weights Update Method: a Meta-Algorithm and Applications. *Theory of Computing*, 8(1):121–164, 2012.

[2] B. Awerbuch, Y. Azar, and A. Meyerson. Reducing truth-telling online mechanisms to online optimization. In L. L. Larmore and M. X. Goemans, editors, *STOC*, pages 503–510. ACM, 2003.

[3] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive on-line routing. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 32–40, Nov 1993.

[4] B. Awerbuch, Y. Azar, S. Plotkin, and O. Waarts. Competitive Routing of Virtual Circuits with Unknown Duration. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '94, pages 321–327, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[5] Y. Bartal, R. Gonen, and N. Nisan. Incentive compatible multi unit combinatorial auctions. In J. Y. Halpern and M. Tennenholtz, editors, *TARK*, pages 72–87. ACM, 2003.

[6] A. Blum and Y. Mansour. From External to Internal Regret. *Journal of Machine Learning Research*, 8:1307–1324, 2007.

[7] P. Briest, P. Krysta, and B. Vöcking. Approximation techniques for utilitarian mechanism design. In H. N. Gabow and R. Fagin, editors, *STOC*, pages 39–48. ACM, 2005.

[8] N. Buchbinder, K. Jain, and J. S. Naor. Online Primal-dual Algorithms for Maximizing Ad-auctions Revenue. In *Proceedings of the 15th Annual European Conference on Algorithms*, ESA'07, pages 253–264, Berlin, Heidelberg, 2007. Springer-Verlag.

[9] N. Buchbinder and J. Naor. Online Primal-dual Algorithms for Covering and Packing Problems. In *Proceedings of the 13th Annual European Conference*

on Algorithms, ESA'05, pages 689–701, Berlin, Heidelberg, 2005. Springer-Verlag.

[10] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games.* Cambridge University Press, 2006.

[11] E. Chastain, A. Livnat, C. Papadimitriou, and U. Vazirani. Multiplicative Updates in Coordination Games and the Theory of Evolution. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 57–58, New York, NY, USA, 2013. ACM.

[12] E. Clarke. Multipart pricing of public goods. *Public Choice*, 8:19–33, 1971.

[13] N. R. Devanur. Online Algorithms with Stochastic Input. *SIGecom Exch.*, 10(2):40–49, June 2011.

[14] N. R. Devanur, K. Jain, B. Sivan, and C. A. Wilkens. Near optimal online algorithms and fast approximation algorithms for resource allocation problems. In Y. Shoham, Y. Chen, and T. Roughgarden, editors, *ACM Conference on Electronic Commerce*, pages 29–38. ACM, 2011.

[15] S. Dobzinski and R. P. Leme. Efficiency Guarantees in Auctions with Budgets. *CoRR*, abs/1304.7048, 2013.

[16] K. M. Elbassioni, K. Makino, K. Mehlhorn, and F. Ramezani. On Randomized Fictitious Play for Approximating Saddle Points Over Convex Sets. *CoRR*, abs/1301.5290, 2013.

[17] D. Fotakis, P. Krysta, and C. Ventre. Combinatorial Auctions Without Money. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 1029–1036, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

[18] Y. Freund and R. E. Schapire. A decision-theoretic generalisation of on-line learning and application to boosting. *Journal of computer and system science*, 55:119–139, 1997.

[19] N. Garg and J. Könemann. Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems. In *FOCS*, pages 300–309. IEEE Computer Society, 1998.

[20] T. Groves. Incentives in Teams. *Econometrica*, 41:617–631, 1973.

[21] S. Hart and A. Mas-Colell. A Simple Adaptive Procedure Leading to Correlated Equilibrium. *Econometrica*, 68(5):1127–1150, Sept. 2000. doi: 10.1111/1468-0262.00153.

[22] T. Kesselheim, A. Tönnis, K. Radke, and B. Vöcking. Primal Beats Dual on Online Packing LPs in the Random-order Model. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC '14, pages 303–312, New York, NY, USA, 2014. ACM.

[23] P. Krysta and B. Vöcking. Online Mechanism Design (Randomized Rounding on the Fly). In A. Czumaj, K. Mehlhorn, A. M. Pitts, and R. Wattenhofer, editors, *ICALP (2)*, volume 7392 of *Lecture Notes in Computer Science*, pages 636–647. Springer, 2012.

[24] N. Littlestone and M. K. Warmuth. The Weighted Majority Algorithm. In *FOCS*, pages 256–261. IEEE Computer Society, 1989.

[25] G. Loomes and R. Sugden. Regret theory: An alternative approach to rational choice under uncertainty. *Economic Journal*, 92:805–824, 1982.

[26] C. H. Papadimitriou. Computing correlated equilibria in multi-player games. In H. N. Gabow and R. Fagin, editors, *STOC*, pages 49–56. ACM, 2005.

[27] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast Approximation Algorithms for Fractional Packing and Covering Problems. In *FOCS*, pages 495–504. IEEE Computer Society, 1991.

[28] P. Raghavan. Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs. *J. Comput. Syst. Sci.*, 37(2):130–143, 1988.

[29] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[30] N. E. Young. Randomized Rounding Without Solving the Linear Program. In K. L. Clarkson, editor, *SODA*, pages 170–178. ACM/SIAM, 1995.