Primal-Dual Combinatorial Algorithms

Qihui Zhu

May 8, 2009

ABSTRACT

Linear program and its duality have long been ubiquitous tools for analyzing NP-hard problems and designing fast approximation algorithms. Plotkin *et al* proposed a primal-dual combinatorial algorithm based on linear duality for fractional packing and covering, which achieves significant speedup on a wide range of problems including multicommodity flow. The key ideas there are: 1) design a primal oracle with partial constraints which can employ existing efficient combinatorial algorithms; 2) perform multiplicative updates on dual variables from the "feedback" of the oracle and drive the solution to optimal. Recently, this algorithm is extended to SDP based relaxation on max-cut, sparsest cut (Arora *et al*), and general convex programming (Khandekar *et al*). In this paper, we summarize previous primal-dual algorithms as a unified computational paradigm. We show how several important applications can be derived from this framework. We also summarize the characteristics of them and address some common issues.

Contents

Chapter 1

Introduction

Linear programming (LP) has been widely used for analyzing combinatorial problems and designing fast approximation algorithms. The LP formulation leads to principle approaches for the large class of packing and covering problems [18, 23], multicommodity flow [18, 19], travelling salesman problem (TSP) [13], faculty location [21], etc. The power of LP based algorithms largely attributes to the *duality* which simultaneously considers two different but coupled problems: the primal and the dual. Each one of them serves the guidance and bound on solving its counterpart, with a different perspective to the problem.

In a seminal work [18], Plotkin *et al* proposed a *primal-dual combinatorial algorithm* for fractional packing and covering, which greatly outperformed previous approaches on the a large set of problems such as minimal cost multicommodity flow, the Held-Karp bound for TSP, and cutting stock. The key idea is to feed the current estimate of the dual to improve the primal during iterations, and vice versa. On the primal side, one solves an oracle with partial constraints and a simplified cost function induced by dual variables. This provides a freedom of designing oracles adapted to different problems and can employ existing efficient combinatorial algorithms. On the dual side, dual variables are adjusted by a multiplicative update rule according to the "feedback" from the oracle. The updated dual variables thus give a tighter bound in the next iteration.

The primal-dual formulation provides more insight to the problem than just treating LP as a blackbox. Computationally, while solving LP using general purpose solutions [20, 16, 22] (*e.g.* interior point methods) has shown some degree of success, *combinatorial* algorithms built on the primal-dual formulation can exploit specific structures, generate much more efficient approximation solutions, and provide explicit manipulation to the computation routine.

The computation paradigm has been extended in many directions recently. Garg *et al* [8] developed width independent algorithms for multicommodity flow, maximum concurrent flow and other fractional packing problems by making adaptive advances. Semidefinite programming (SDP) based relaxation has proved to give tighter bounds

on a set of NP-hard problems such as MaxCut [10], Sparsest Cut [1, 4], and Balanced Separator [4]. Arora *et al* extended the primal-dual LP algorithm to the SDP relaxation setting, which achieved significant speedup [3, 2]. Also, Khandekar *et al* generalized the technique to convex programming [13, 9].

In this paper, we unify the previous primal-dual approaches into one computational framework and summarize into a common algorithm. We present the complexity analysis based on a exponential potential function in the online prediction setting. In additional to fractional packing and covering, we show how existing algorithms on multicommodity flow, SDP relaxation, boosting and matrix game can be derived from this common paradigm.

Chapter 2

Fractional Packing and Covering

In this chapter, we define the packing and covering problem, and introduce the fractional packing and covering which is the central problem throughout the paper. The linear programming formulation of these two problems are dual to each other and hence have the same optimum. We also give an equivalent min-max/max-min formulation of fractional packing/covering, which suggests connection to applications developed later.

2.1 Packing and Covering

The packing problem arises from choosing the most valuable objects to fill knapsacks. Suppose there are n objects whose prices are p_i $(i = 1, ..., n)$. One would like to choose a subset of these items maximizing their total price, subject to m capacity constraints such as weight, dimension, etc. Denote the maximum value of each capacity constraint as c_j and the contribution from item i as W_{ji}. The capacity constraint requires the total contribution $\sum_{i \text{ selected}} W_{ji}$ not to exceed c_j. Finding the optimal packing can be written as the following integer programming problem:

$$\begin{aligned} \max_{\mathbf{x}\in\{0,1\}^n} \quad & \sum_i p_i x_i \\ \text{s.t.} \quad & \sum_i W_{ji} x_i \le c_j, \quad j = 1, ..., m \end{aligned} \tag{2.1}$$

where x_i is the 0/1 indicator of whether object i is selected. By relaxing the integer constraint $\mathbf{x} \in \{0, 1\}^n$ to $0 \le \mathbf{x} \le 1$, we obtain a linear program which provides a lower bound to eq. (2.2). We call it *fractional packing* in the following form:

$$\begin{aligned} \max_{\mathbf{x}\in\mathbb{R}^n} \quad & \mathbf{p^T x} \\ \text{s.t.} \quad & \mathbf{A} \cdot \mathbf{x} \le \mathbf{c} \\ & \mathbf{x} \ge 0 \end{aligned} \tag{2.2}$$

Here $\mathbf{A} = [\mathbf{W}; \mathbf{I}]$ and $\mathbf{c} = [c_1, ..., c_m, \underbrace{1, ..., 1}_{n}]^T$. Hence the constraint $\mathbf{x} \le 1$ has been folded into the matrix constraint $\mathbf{A} \cdot \mathbf{x} \le \mathbf{c}$.

The covering problem is to find sets with minimal total cost to cover elements. Let c_j's be the costs of the n sets. Each set j covers element i for W_{ij} times. The multiplicity of each element i to be covered is required to be at least p_i. Let y_j be the number of copies of set j that are selected (choosing multiple copies are allowed). Similar to packing, the covering problem can be written as an integer program, and relaxed to *fractional covering*:

$$\begin{aligned} \min_{\mathbf{y}\in\mathbb{N}^n} \quad & \sum_j c_i y_j && (2.3)\\ \text{s.t.} \quad & \sum_j W_{ij} y_j \geq p_i, \quad i = 1, ..., n \\ \min_{\mathbf{y}\in\mathbb{R}^n} \quad & \mathbf{c^T y} && (2.4)\\ \text{s.t.} \quad & \mathbf{A^T} \cdot \mathbf{y} \geq \mathbf{p} \\ & \mathbf{y} \geq 0 \end{aligned}$$

We will focus on the fractional packing and covering eq. (2.2),(2.4). In the following chapters, we will see that they are the foundation of designing algorithms for numerous NP-hard problems.

2.2 Lagrangian Dual

The fractional packing problem eq. (2.2) and fractional covering eq. (2.4) are actually Lagrangian duals. By introducing nonnegative Lagrangian multipliers $(\mathbf{y}, \lambda)$ to the constraints $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ and $\mathbf{x} \geq 0$ respectively, the Lagrangian function $\mathcal{L}(\mathbf{x}, \mathbf{y}, \lambda) = \mathbf{p^T x} + \mathbf{y^T}(\mathbf{c} - \mathbf{Ax}) + \lambda^\mathbf{T}\mathbf{x}$ always serves a upper bound of the fractional packing cost function eq. (2.2) $\mathbf{p^T x}$, whenever $\mathbf{x}$ is feasible or not. Therefore, $\max_\mathbf{x} \mathcal{L}(\mathbf{x}, \mathbf{y}, \lambda)$ bounds the optimum of eq. (2.2). To obtain the best upper bound, one would like to solve:

$$\begin{aligned} \min_{\mathbf{y},\lambda} \max_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \mathbf{y}, \lambda) &= \min_{\mathbf{y},\lambda} \max_{\mathbf{x}} \mathbf{p^T x} + \mathbf{y^T}(\mathbf{c} - \mathbf{Ax}) + \lambda^\mathbf{T}\mathbf{x} \\ &= \min_{\mathbf{y},\lambda} \max_{\mathbf{x}} \mathbf{c^T y} + (\mathbf{p} - \mathbf{A^T y} + \lambda)^\mathbf{T}\mathbf{x} \\ &= \min_{\mathbf{p}-\mathbf{A^T y}+\lambda=\mathbf{0}} \mathbf{c^T y} \\ &= \min_{A^T y \geq p} \mathbf{c^T y} \end{aligned} \tag{2.5}$$

The third equality is due to the fact that if $\mathbf{p} - \mathbf{A^T y} + \lambda \neq \mathbf{0}$, one can always choose appropriate $\mathbf{x}$ such that $\mathcal{L}(\mathbf{x}, \mathbf{y}, \lambda)$ goes to infinity. Hence only when the condition $\mathbf{p} - \mathbf{A^T y} + \lambda = \mathbf{0}$ is satisfied, it gives a finite bound. Now eq. (2.5) is exactly the fractional covering problem eq. (2.4). By strong duality of linear program, the optimum of eq. (2.2) and eq. (2.4) coincides with [5]. Therefore packing and covering are essentially flipped sides of the same coin: solving one implies the other.

2.3 Min-Max Formulation

Interestingly, the packing problem has an alternative min-max formulation, which suggests its intimate connection to zero-sum game and Von Neumann's celebrated minimax theorem [17]. We will discuss this application in Chapter 4.

To simplify the formulation, we assume $\mathbf{c} > 0$ for now and for many packing problem the capacity $\mathbf{c}$ is indeed positive. Let P be a convex set defined on $\mathbf{x}$ and $f_j(\mathbf{x})$ ($j = 1, ..., m$) be a set of convex functions. The general *min-max problem* is defined as:

$$\min_{\mathbf{x}\in P} \max_{j\in[m]} f_i(\mathbf{x}) \tag{2.6}$$

where $[m] = \{1, ..., m\}$ denotes the set of indices.

For fractional packing problem, we set

$$P = \{\mathbf{x} : \ \mathbf{p^T x} = 1, \ \mathbf{x} \geq 0\} \tag{2.7}$$

$$f_j(\mathbf{x}) = \mathbf{A_j} \cdot \mathbf{x}/c_j \tag{2.8}$$

where $\mathbf{A_j}$ is the jth row of matrix $\mathbf{A}$. The optimal value of eq. (2.6) μ^*_{mm} and that of eq. (2.2) μ^*_p is related by $\mu^*_{mm} = 1/\mu^*_p$. For any feasible solution $\mathbf{x}_{mm}$ with objective value μ_{mm}, $\mathbf{x} = \mathbf{x}_{mm}/\mu_{mm}$ is feasible for the packing eq. (2.2), meaning $\mu^*_{mm} \geq 1/\mu^*_p$. Conversely, any feasible packing solution $\mathbf{x}_p$ produces a feasible solution $\mathbf{x}_{mm} = \mathbf{x}_p/\mu_p$ whose objective value is $1/\mu_p \geq \mu^*_{mm}$. Therefore, the min-max formulation eq. (2.6) is equivalent to the original packing formulation eq. (2.2).

Similarly, covering eq. (2.4) can be formulated as a max-min problem:

$$\max_{\mathbf{y}\in P} \min_{j\in[n]} g_j(\mathbf{y}) \tag{2.9}$$

In fact, Young *et al*[23] called eq. (2.6) *generalized packing problem*.

The extra condition of $\mathbf{b} \geq 0$ can be removed if the range of $\mathbf{a_j^T} \cdot \mathbf{x}$ is bounded.

The Lagrangian dual of eq. (2.6) provides a way to bound the cost function which is developed as an oracle in the rest of the paper. The min-max formulation can be rewritten as

$$\begin{aligned} \min_{\mathbf{x}\in P,\lambda} \quad & \lambda \\ \text{s.t.} \quad & f_j(\mathbf{x}) \leq \lambda, \quad j = 1, ..., m \end{aligned}$$

The Lagrangian dual function is $\mathcal{L}(\mathbf{x}, \mathbf{y}, \lambda) = \lambda + \sum_j y_j[f_j(\mathbf{x}) - \lambda] = \sum_j y_j f_j(\mathbf{x}) + \lambda(1 - \sum_j y_j)$. $\max_{\mathbf{y}} \min_{\mathbf{x},\lambda} \mathcal{L}(\mathbf{x}, \mathbf{y}, \lambda)$ gives the dual formulation:

$$\begin{aligned} \max_{\mathbf{y}} \quad & \min_{\mathbf{x}\in P} \sum_j y_j f_j(\mathbf{x}) \\ \text{s.t.} \quad & \sum_j y_j = 1, \ \mathbf{y} \geq 0 \end{aligned} \tag{2.10}$$

If there is an efficient routine computing $\min_{\mathbf{x}\in P} \sum_j y_j f_j(\mathbf{x})$ given a distribution $\mathbf{y}$, one can generate lower bounds on-the-fly from the current dual variables $\mathbf{y}$. As we will see later, this opens the door to efficient combinatorial algorithms.

Chapter 3

Primal Dual Algorithms for Fractional Packing and Covering

3.1 Feasibility Problem

First we start with a feasibility version of the fractional packing problem:

> **(Feasibility)** Given a convex set $P \subseteq \mathbb{R}^n$, an $m \times n$ constraint matrix $\mathbf{A}$ and an $n \times 1$ vector $\mathbf{c}$, determine whether there exists $\mathbf{x} \in P$ such that
>
> $$\mathbf{a_j^T x} - c_j \leq 0, \quad j = 1, ..., m \tag{3.1}$$

Here $\mathbf{a_j^T}$ is the jth row of matrix $\mathbf{A}$ as defined before.

In the packing problem, the convex set P is a simple polytope:

$$P = \{\mathbf{x} : \ \mathbf{p^T x} \leq \alpha, \ 0 \leq \mathbf{x} \leq 1\} \tag{3.2}$$

where α is a constant. If eq. (3.1) is feasible, then the optimal value μ_p^* of eq. (2.2) is at least α. Otherwise it is less than α. By a binary search on α, one can find a $(1+\beta)$ approximation to the optimization problem within $O(\log \beta)$. Our discussion will focus on eq. (3.1) in the subsequent sections.

3.2 Primal-Dual Framework

Primal and dual formulation provides different perspectives on the problem: for the feasibility version, primal solution serves as "yes" certificate while the dual solution serves as "no" certificate. Just as the divide-and-conquer strategy, one would like to generate a series of yes and no certificates to narrow down the search space. Therefore, primal and dual need to communicate, and use one to update the other.

3.2.1 Multiplicative Weight Update: from Primal to Dual

Suppose we are given a primal estimate and its corresponding cost as feedback, how can we update the current dual estimate? We start with considering an online prediction

problem.

Online Prediction. There are m experts who make predictions on uncertain events in the world. Our goal is to construct the best strategy over time from these experts. At time t, if the prediction from the jth expert is taken, the event (possibly adversarial) incurs a positive *reward* $\mathcal{R}_j^t$ and a negative *loss* $-\mathcal{L}_j^t$. Hence the net *value* gained is $\mathcal{V}_j^t = \mathcal{R}_j^t - \mathcal{L}_j^t$. One can construct a mixed strategy from these experts by linearly combining their predictions. A mixed strategy specifies positives weights $\mathbf{y}^t = (y_1, ..., y_m)^T$ on all the experts. The total net value of the strategy will be $\mathcal{V}^t = \sum_j \overline{y}_j^t \mathcal{V}_j^t$ where $\overline{\mathbf{y}}^t = \mathbf{y}^t / \sum_j y_j^t$ are the normalized weights. Consider the event sequence from time $t = 0$ to T. At time t, the strategy chooses weights $\mathbf{y}^t$ on the experts based on all previous observations $\mathcal{R}^k$ and $\mathcal{L}^k$ with $0 \leq k \leq t-1$, and gains a value $\mathcal{V}^t$. One would like to maximize the cumulative value over time $\mathcal{V} = \sum_{t=0}^{T} \mathcal{V}^t$.

Intuitively, experts making correct predictions previously should be up-weighted while experts predicting incorrectly should be down-weighted. In other words, the weights should be updated according to the "feedback" of the experts from the world $\mathcal{V}_j^t$. We introduce a multiplicative weight update scheme to guide the strategy from the feedback:

(Multiplicative Weight Update) Initialize weights $\mathbf{y}^{(0)} = (1, ..., 1)^T$. At time t, prediction from expert j produces a value of $\mathcal{V}_j^t \in [-1, 1]$. Given a constant $\epsilon \in (0, 1)$, update the weights $\mathbf{y}^{t+1}$ at time $t+1$ by

$$\mathbf{y}_j^{t+1} = \mathbf{y}_j^t \exp(\epsilon \mathcal{V}_j^t) \tag{3.3}$$

Theorem 1. *(Perturbed Value of the Strategy) Let* $\mathcal{R} = \sum_t \sum_j \overline{y}_j^t \mathcal{R}_j^t$ *and* $\mathcal{L} = \sum_t \sum_j \overline{y}_j^t \mathcal{L}_j^t$ *be the cumulative reward and loss of the strategy using eq. (3.3). The perturbed value of the strategy given by eq. (3.3) is worse than the performance of best pure strategy only by* $\frac{\log m}{\epsilon}$*, as stated in the following inequality:*

$$\max_j \mathcal{V}_j \leq \exp(\epsilon)\mathcal{R} - \exp(-\epsilon)\mathcal{L} + \frac{\log m}{\epsilon} \tag{3.4}$$

Proof. Consider the potential function $\Phi^t = \sum_j y_j^t$.

On the one hand, we can compute it using the update rule:

$$\begin{aligned}
\Phi^t &= \sum_j y_j^t \\
&= \sum_j y^{(0)} \prod_{k=1}^{t} \exp[\epsilon \mathcal{V}_j^k] && \text{(Update rule (3.3))} \\
&= \sum_j \exp[\epsilon \sum_{k=1}^{t} \mathcal{V}_j^k] && (y_j^{(0)} = 1) \\
&\geq \exp[\epsilon \cdot \sum_{k=1}^{t} \mathcal{V}_j^k] && (3.5)
\end{aligned}$$

Note the above inequality holds for any j. Therefore, Φ^t is bounded below by

$$\Phi^t \geq \exp[\epsilon \cdot \max_j \mathcal{V}_j] \tag{3.6}$$

On the other hand, we have

$$\begin{aligned}
y_j^{t+1} - y_j^t &= y^t[\exp(\epsilon \mathcal{V}_j^t) - 1] \\
&\leq y^t \cdot (\epsilon \mathcal{V}_j^t) \cdot \exp(\epsilon \mathcal{V}_j^t) \\
&= y^t[\epsilon \exp(\epsilon \mathcal{V}_j^t) \mathcal{R}_j^t - \epsilon \exp(\epsilon \mathcal{V}_j^t) \mathcal{L}_j^t] \\
&\leq y^t[\epsilon \exp(\epsilon) \mathcal{R}_j^t - \epsilon \exp(-\epsilon) \mathcal{L}_j^t] \\
&= y^t \epsilon \widetilde{\mathcal{V}}_j^t
\end{aligned}$$

Here $\widetilde{\mathcal{V}}_j^t = \exp(\epsilon)\mathcal{R}_j^t - \exp(-\epsilon)\mathcal{L}_j^t$ is the "perturbed" version of value $\mathcal{V}_j^t$. The first inequality holds because $\exp(x) - 1 \leq x \cdot \exp(x)$ for any x. The second inequality is due to the fact that $\mathcal{V}_j^t \in [-1, 1]$.

By summing up the above inequality over j, we have

$$\begin{aligned}
\Phi^{t+1} &= \sum_j (y_j^{t+1} - y_j^t) + \Phi^t \\
&\leq \sum_j y_j^t \epsilon \widetilde{\mathcal{V}}_j^t + \Phi^t \\
&= \epsilon \Phi^t \cdot \sum_j y_j^t \widetilde{\mathcal{V}}_j^t / \sum_j y_j^t + \Phi^t \\
&= \Phi^t (1 + \epsilon \widetilde{\mathcal{V}}^t) \\
&\leq \Phi^t \cdot \exp(\epsilon \widetilde{\mathcal{V}}^t) && (1 + x \leq \exp(x))
\end{aligned}$$

Using induction over t and $\Phi^0 = m$, we bound Φ^t above by

$$\Phi^t \leq m \cdot \exp(\sum_k \epsilon \widetilde{\mathcal{V}}^k) \tag{3.7}$$

Finally combining eq. (3.6), (3.7) yields

$$\epsilon \cdot \max_j \mathcal{V}_j \leq \log m + \sum_k \epsilon \widetilde{\mathcal{V}}^k \tag{3.8}$$

which is equivalent to eq. (3.4).

□

Theorem 1 is essential in all the analysis presented in this paper. It proves the quality of the multiplicative update rule (3.3). Since the average strategy given by the update rule cannot exceed the best strategy in the hindsight, we would like the gap between their values $\max_j \sum_t \mathcal{V}_j^t$ and $\sum_t \mathcal{V}^t$. This value is called *regret* of the strategy. The theorem proves the fact that the regret is as small as $\log m/\epsilon$ and constant to time T. We can bound the regret over time by the following corollary:

Corollary 1. *(Regret Over Time) If $\mathcal{V}_j^t \in [-\rho, \rho]$ for all j, then we have a bound on the average value $\mathcal{V}/T$:*

$$\max_j \frac{\mathcal{V}_j}{T} \leq \frac{\mathcal{V}}{T} + \frac{\rho \log m}{\epsilon T} + \rho\epsilon \exp(\epsilon) \tag{3.9}$$

Proof. Since $\mathcal{V}_j^t \in [-\rho, \rho]$, we can substitute $\mathcal{V}_j^t$ by $\mathcal{V}_j^t/\rho$ and prove the following inequality for $\mathcal{V}_j^t \in [-1, 1]$:

$$\max_j \mathcal{V}_j \leq \mathcal{V} + \frac{\log m}{\epsilon} + T\epsilon \exp(\epsilon)$$

We set $\mathcal{R}_j^t = \max(0, \mathcal{V}_j^t)$ and $\mathcal{L}_j^t = \max(0, -\mathcal{V}_j^t)$, which satisfies $\mathcal{V}_j^t = \mathcal{R}_j^t - \mathcal{L}_j^t$.

Under these simplifications, we can apply Theorem 1 on $\mathcal{V}$:

$$\begin{aligned}
\max_j \mathcal{V}_j &\leq \widetilde{\mathcal{V}} + \frac{\log m}{\epsilon} \\
&= \mathcal{V} + \frac{\log m}{\epsilon} + (\exp(\epsilon) - 1)\mathcal{R} - (\exp(-\epsilon) - 1)\mathcal{L} \\
&\leq \mathcal{V} + \frac{\log m}{\epsilon} + \epsilon \exp(\epsilon)|\mathcal{V}| \\
&\leq \mathcal{V} + \frac{\log m}{\epsilon} + \epsilon \exp(\epsilon) T
\end{aligned}$$

The first inequality uses the fact that $|\mathcal{V}| = \mathcal{R} + \mathcal{L}$, $\exp(\epsilon) - 1 \leq \epsilon \exp(\epsilon)$ and $1 - \exp(-\epsilon) \leq \epsilon < \epsilon \exp(\epsilon)$. □

The above bound shows that the regret over time consists of two terms: the term $\frac{\rho \log m}{\epsilon T}$ which can be "washed out" by time and the other term $\rho\epsilon \exp(\epsilon)$ which cannot. If we would like to diminish the regret over time, for example proportional to a small number δ, we can set $\epsilon \sim \delta/\rho$ and $T \sim \rho^2/\delta^2$. However, if $\mathcal{V}$ only contains reward or loss, the result can be strengthened as:

Corollary 2. *(Regret for Reward Only) If $\mathcal{V}_j^t \in [0, \rho]$ for all j, i.e. $\mathcal{L}_j^t = 0$ for all t and j, then we have a bound on the average value $\mathcal{V}/T$:*

$$\max_j \frac{\mathcal{V}_j}{T} \leq \exp(\epsilon) \cdot \frac{\mathcal{V}}{T} + \frac{\rho \log m}{\epsilon T} \tag{3.10}$$

Corollary 3. *(Regret for Loss Only) If $\mathcal{V}_j^t \in [-\rho, 0]$ for all j, i.e. $\mathcal{R}_j^t = 0$ for all t and j, then we have a bound on the average value $\mathcal{V}/T$:*

$$\max_j \frac{\mathcal{V}_j}{T} \leq \exp(-\epsilon) \cdot \frac{\mathcal{V}}{T} + \frac{\rho \log m}{\epsilon T} \tag{3.11}$$

These two corollaries are direct consequences of eq. (3.4). They are stronger claims than Corollary 1 because we only need to set $T \sim \rho/\delta$ to make the regret over time small, instead of $T \sim \rho/\delta$. As we will see later, this is the fundamental difference between packing/covering and general LP, in which the latter has higher complexity.

Width. We have introduced an important parameter ρ refered as *width* in the literature. Since the running time T is proportional to ρ or even ρ^2, reducing width is critical for almost all primal-dual algorithms. We will see the example on multicommodity flow in Chapter 4.

3.2.2 The Oracle: from Dual to Primal

From the dual formulation eq. (2.10), we would like to improve the current primal solution by minimizing $\sum_j y_j f_j(\mathbf{x})$.

> **(Oracle)** Given a convex **constraint set** $P \subseteq \mathbb{R}^n$, a dual variable $\mathbf{y} \in \mathbb{R}^m$ and a set of functions $\mathcal{V}_j(\mathbf{x})$ $(j = 1, ..., m)$. Optimize the linear combination of $\mathcal{V}_j(\mathbf{x})$ in the constraint set P:
>
> $$\min_{\mathbf{x} \in P} \sum_j y_j \mathcal{V}_j(\mathbf{x}) \tag{3.12}$$

The constraints in the original problem have been separated into two parts. Constraints easy to check and optimize are pushed into Constraint Set P, making the oracle efficient to compute. Hard constraints are left outside and are only approximated by the Lagrangian as in eq. (3.12). It is a design choice how to divide the two.

In the case of packing, P is given by eq. (3.2). Define $\mathcal{V}_j(\mathbf{x}) = \mathbf{a_j^T x} - c_j$ for $j = 1, ..., m$. Notice that $\sum_j y_j \mathcal{V}_j(\mathbf{x}) = (\mathbf{A^T y})^\mathbf{T} \mathbf{x} - \mathbf{c^T y}$, given $\mathbf{y}$, the oracle becomes

$$\begin{aligned} &\min_{\mathbf{x}} \ (\mathbf{A^T y})^\mathbf{T} \mathbf{x} \\ &\text{s.t. } \mathbf{c^T x} = \alpha, \ 0 \leq \mathbf{x} \leq 1 \end{aligned} \tag{3.13}$$

If $\mathbf{c} \geq 0$ and $\mathbf{A} \geq 0$, one can solve eq. (3.13) by simply sorting $(A^T y)_j / c_j$ in ascending order, and choose $x_j = 1$ until according to the order until $\mathbf{c^T x} = \alpha$ is satisfied. The oracle (3.12) simply reduces to sorting whose complexity is $O(n \log n)$.

3.2.3 Algorithm Summary

So far we have all the ingredients of primal dual combinatorial algorithms. We summarize the primal-dual algorithm for packing as follows:

Algorithm 1 Primal Dual Algorithm

Initialize $\mathbf{y}^0 = (1, ..., 1)^T$, $t = 0$, $S = 0$, $\epsilon = \delta/3\rho$.
Define $f_j(\mathbf{x}) = \mathbf{a_j^T x} - c_j$.
Do
    Call oracle (3.12) and obtain the optimum μ^t and optimal point $\mathbf{x}^t$.
    If $\mu^t > 0$ then return *infeasible*.
    Compute $w^t = 1/\max_j |f_j(\mathbf{x}^t)|$.
    Run multiplicative weight update (3.3):
        $y_j^{t+1} = y_j^t \exp(\epsilon w^t f_j(\mathbf{x}^t))$
    $S := S + w^t$.
    $t := t + 1$.
Until $S \geq 9\rho \log m/\delta^{-2}$.
Output *feasible solution* $\bar{\mathbf{x}} = \frac{\sum_t w^t \mathbf{x}^t}{\sum_t w^t}$.

3.3 Complexity Analysis

Theorem 2. *(Complexity of the Primal Dual Algorithm) Algorithm 1 either declare that the fractional packing eq. (2.2) is infeasible, or output an approximate feasible solution* $\bar{\mathbf{x}}$ *satisfying*

$$\mathbf{a_j^T}\bar{\mathbf{x}} - c_j \leq \delta \tag{3.14}$$

for all $j = 1, ..., m$. *The total number of calls to the oracle is* $O(\rho^2\delta^{-2} \log m)$ *with* $\rho = \max_j \max_{\mathbf{x}\in P} |f_j(\mathbf{x})|$.

Proof. We build our proof based on Corollary 1. First notice that if at some time $\mu^t > 0$, then the eq. (2.2) is indeed infeasible. Otherwise suppose there exists $\mathbf{x}^t$ such that $f_j(\mathbf{x}^t) = \mathbf{a_j^T x}^t - c_j \leq 0$ for all j. Because $\mathbf{y}^t \geq 0$ throughout the algorithm, $\mu^t \leq \sum_j y_j^t f_j(\mathbf{x}^t) \leq 0$, a contradiction.

Suppose the algorithm runs to the end and output $\bar{\mathbf{x}}$. Let $\mathcal{V}_j^t = w^t f_j(\mathbf{x}^t)$ be the value

incurred by the update. Notice that $\mathcal{V}_j^t \in [-1, 1]$. By applying Corollary 1, we have

$$\begin{aligned}
\max_j[\mathbf{a_j^T}\bar{\mathbf{x}} - c_j] &= \max_j \frac{\sum_t w^t(\mathbf{a_j^T}\mathbf{x}^t - c_j)}{\sum_t w^t} \\
&= \max_j \frac{\sum_t \mathcal{V}_j^t}{\sum_t w^t} \\
&\leq \frac{1}{\sum_t w^t}[\mathcal{V} + \frac{\log m}{\epsilon} + \epsilon T \exp(\epsilon)] \\
&\leq \frac{1}{\sum_t w^t}[\frac{\log m}{\epsilon} + \epsilon T \exp(\epsilon)] \\
&= \frac{1}{S}[\frac{\log m}{\epsilon} + \epsilon T \exp(\epsilon)] \\
&\leq \delta
\end{aligned} \tag{3.15}$$

The first inequality uses the fact that $\mathcal{V}^t = (w^t / \sum_j y_j^t) \sum_j y_j^t f_j(\mathbf{x}^t) = w^t \mu^t / \sum_j y_j^t \leq 0$ for every t since the oracle never fails. The last inequality is due to the termination condition $S \geq 9\rho \log m/\delta^{-2}$, $T/S = T/\sum_t w^t \leq \rho$ and $\epsilon = 3\delta/\rho$.

Therefore, $\mathbf{x}$ returned by the algorithm satisfies the approximate feasibility eq. (3.14). Finally, each time the algorithm collects $w^t \geq 1/\rho$ and it terminates when $S = \sum_t w_t \geq S \geq 9\rho \log m/\delta^{-2}$, the total number of iterations is at most $O(\rho^2 \delta^{-2} \log m)$. □

Variant 1 If $\mathbf{A} \geq 0$ and $\mathbf{c} \geq 0$, we can improve the running time of Algorithm 1 to $O(\rho\delta^{-1} \log m)$ by changing the termination condition to $S \geq \rho\delta^{-1}\epsilon^{-1} \log m$ and set $f_j(\mathbf{x}) = \mathbf{a_j^T}\mathbf{x}/c_j$.

Variant 2 If $f_j(\mathbf{x}) \geq 0$ for $\mathbf{x} \in P$, we can improve the running time of Algorithm 1 to $O(\rho\delta^{-1} \log m)$ by changing the termination condition to $S \geq \rho\delta^{-1}\epsilon^{-1} \log m$.

In both cases, we can use Corollary 2 instead. Eq. (3.15) has a tighter bound $\max_j[\mathbf{a_j^T}\bar{\mathbf{x}} - c_j] \leq \frac{\log m}{\epsilon S}$, the rest of the analysis falls through. The two variants subsume the results in [18].

Chapter 4

Applications and Extensions

The computational paradigm developed in the previous chapter is very flexible. CONSTRAINT SET and ORACLE can be tailored to specific problems. Efficient combinatorial algorithms can be incorporated as ORACLE and largely reduce the complexity of primal-dual algorithms. Therefore, it is used in designing fast approximation algorithms for numerous problems. We summarize several applications and extensions, and show how they are derived from the basic primal-dual algorithm in the following sections.

4.1 Multicommodity Flow

We first illustrate the primal-dual framework on the multicommodity flow problem.

Maximum Multicommodity Flow. Given a graph $G = (V, E)$, edge capacity c_e on every edge $e \in E$ and k pairs of sources and sinks (s_i, t_i), with each one transferring one commodity, one would like to maximize the sum of flows allowed by the graph. Let $\mathcal{P}$ be the set of all possible paths between pairs of sources and sinks. One can think of the multicommodity flow as "packing" these paths to saturate the edge capacities. The LP formulation can be written as

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{p \in \mathcal{P}} x_p \\ \text{s.t.} \quad & \sum_{p \ni e} x_p \leq c_e \ \forall e, \qquad \mathbf{x} \geq 0 \end{aligned}$$

Note that we never explicitly solve this LP, because the number of variables is just huge: there are exponentially many different paths. The dual problem is more manageable:

$$\begin{aligned} \min_{\mathbf{y}} \quad & \sum_{e} y_e c_e \\ \text{s.t.} \quad & \sum_{e \in p} y_e \geq 1 \ \ \forall p, \qquad \mathbf{y} \geq 0 \end{aligned}$$

The meaning of the dual is to minimize the cost $\sum_e y_e c_e$ given a "distance" function $\mathbf{y}$ on edges, subject to the shortest path between all pairs of terminals has length at least 1. It suggests that only the shortest paths matter in the final multicommodity flow.

One naive idea is to apply Algorithm 1 directly by taking $P = \{\mathbf{x} : \sum_p x_p = \alpha\}$. Consider $f_e(\mathbf{x}) = \sum \sum_{p \ni e} x_p / c_e$. The oracle minimizes $\sum_e y_e \sum_{p \ni e} x_p / c_e = \sum_p x_p \sum_{e \in p} y_e / c_e$ which amounts to find the shortest path with length y_e / c_e. Every time the oracle routes the flow following the shortest path. The final flow is the average flow over time. However, the width is as high as $\rho = \max_e \max_{\mathbf{x} \in P} |f_e(\mathbf{x})| \geq f^* / c_{min}$ where f^* is the optimal flow. This only gives a pseudo-polynomial algorithm because the running time depends on c_{min} instead of $\log(c_{min})$ (length of input).

The cause of the problem is that each step the oracle routes "too much" flow, without even respecting the capacity constraint. Another way to see it is that the constraint for P is too loose, making the width too high. Garg *et al* [9] fixed this problem by only routing $\min_{e \in p^*} c_e$ amount of flow, where p^* is the shortest path returned by the oracle. With this modification, we can map Algorithm 1 to the problem:

- Constraint Set. $P = \{\mathbf{x} : \sum_p x_p = \alpha,\ x_p \leq \min_{e \in p} c_e\}$.
- Oracle. Given dual variables $\mathbf{y}$, find $\mathbf{x} \in P$ minimizing $\sum_e y_e \sum_{p \ni e} x_p / c_e$ by computing the shortest path p^*.
- Multiplicative Weight Update. Route $f^t = \min_{e \in p^*} c_e$ units of flow, which is the minimal capacity on p^*. Update $\mathbf{y}$ by $y_e^{(t+1)} = y_e^{(t+1)} \cdot \exp(\epsilon f^t / c_e)$.

The width has been *automatically upper bounded by 1* since $f^t \leq c_e$. This generate the following combinatorial algorithm:

Algorithm 2 Maximum Multicommodity Flow

1: Initialize $y_e^0 = 1$ for all edges e. $f = 0$, $t = 0$.
2: Do
3: Call oracle to compute the shortest path p^t with length y_e^t / c_e.
4: Route $f^t = \min_{e \in p^t} c_e$ units of flow on p^t: $x_e^t = f^t$ if $e \in p^t$, otherwise $x_e^t = 0$.
5: Run multiplicative weight update:
$$y_e^{t+1} = y_e^t \exp(\epsilon f^t / c_e)$$
6: $f := f + f^t$.
7: $t := t + 1$.
8: Until $f \geq \frac{\log m}{\epsilon^2}$.
9: Output *feasible flow* $\mathbf{x}_{fea} = \mathbf{x}/C$ and $f_{fea} = f/C$ with $C = \max_e \frac{\sum_{p \ni e} x_e}{c_e}$.

Here the value C is called the *maximal congestion* which measures maximal relative load on edges.

Analysis. We will show that the final flow returned by the algorithm is feasible and close to the optimum.

Proposition 1. *The final flow $\mathbf{x}_{fea}$ in Algorithm 2 is feasible and $f_{fea} \geq (1 - 2\epsilon) f^*$ where f^* is the optimal flow.*

Proof. Set $\mathcal{V}_e^t = f^t/c_e$ as the value in Theorem 1. Since it is always positive, the loss can be ignored: $\mathcal{L} = 0$.

$$\max_e \sum_t \mathcal{V}_e^t \leq \exp(\epsilon) \sum_t \mathcal{V}^t + \frac{\log m}{\epsilon} \tag{4.1}$$

Note that $x_e^t = f^t$ if and only if e on path p^t selected by the oracle. Hence we have

$$\max_e \sum_t \mathcal{V}_e^t = \max_e \frac{\sum_t x_e^t}{c_e}$$

The left hand side of eq. (4.1) is the maximal congestion C.

Next we bound the value of $\sum_t \mathcal{V}^t$:

$$\sum_t \mathcal{V}^t = \sum_t \frac{\sum_{e \in p^t} y_e^t \cdot (f^t/c^e)}{\sum_e y_e^t} = \sum_t f^t \cdot \frac{\sum_{e \in p^t} y_e^t/c_e}{\sum_e y_e^t}$$

We will prove that the factor $(\sum_{e \in p^t} y_e^t/c_e)/(\sum_e y_e^t) \leq 1/f^*$ where $f^* = \sum_{p'} f_{p'}^*$ is the optimal flow. In fact

$$\begin{aligned}
\frac{\sum_{e \in p^t} y_e^t/c_e}{\sum_e y_e^t} &\leq \frac{\sum_{e \in p^t} y_e^t/c_e}{\sum_e y_e^t \cdot \sum_{p' \ni e} f_{p'}^*/c_e} \\
&= \frac{\sum_{e \in p^t} y_e^t/c_e}{\sum_{p'} f_{p'}^* \cdot \sum_{e \in p'} y_e^t/c_e} \\
&\leq \frac{\sum_{e \in p^t} y_e^t/c_e}{\sum_{p'} f_{p'}^* \cdot \sum_{e \in p^t} y_e^t/c_e} \\
&= \frac{1}{\sum_{p'} f_{p'}^*} = \frac{1}{f^*}
\end{aligned} \tag{4.2}$$

The first inequality is due to the fact that the optimal flow must satisfy the capacity constraint $\sum_{p' \ni e} f_{p'}^* \leq c_e$. The second inequality uses the property that p^t is the shortest path by length y_e^t/c_e, and $\sum_{e \in p^t} y_e^t/c_e \leq \sum_{e \in p'} y_e^t/c_e$ for any path p'.

Combining eq. (4.1), (4.2), we have

$$C \leq \exp(\epsilon) \frac{f}{f^*} + \frac{\log m}{\epsilon}$$

When the algorithm returns, Since the termination condition is $C \geq \frac{\log m}{\epsilon^2}$, the final flow is close to the optimum:

$$\frac{f}{f^*} \geq \exp(-\epsilon)(C - \frac{\log m}{\epsilon}) \geq (1 - 2\epsilon)C \tag{4.3}$$

$f_{fea} = f/C \geq (1 - 2\epsilon) f^*$ is therefore a $(1 - 2\epsilon)$ approximation.

□

Running Time. In each step t, p^t creates a congestion of 1 on the edge $e \in p^t$ with minimal capacity. The final congestion on any edge is at most $\lceil \frac{\log m}{\epsilon} \rceil$. Therefore, an

edge can be the minimal edge of capacity on shortest path at most $O(\frac{\log m}{\epsilon})$ times. The graph has m edges and each time we call shortest path k times for all terminal pairs. The total running time is $O(m\frac{\log m}{\epsilon} \cdot T_{sp}) = O(km(m + n \log n)\frac{\log m}{\epsilon})$ using Fibonacci heap implementation for shortest path.

4.2 Semidefinite Programming (SDP)

Semidefinite programming (SDP) has been applied for designing approximation algorithms for a wide range of combinatorial problems. The notable SDP relaxation for MAX CUT [10] provides a 0.878-approximation, outperforming LP based relaxation algorithms which only give approximation around 0.5. The method is extended to solve MAX2SAT and other satisfiability problems. Arora *et al* [4] utilizes SDP to generate an $O(\sqrt{\log n})$-approximation SPARSEST CUT. The procedures of these algorithms are very similar: one solves the SDP relaxation to produce a good geometric embedding of the nodes, and then some rounding scheme is applied to generate tight approximation solutions.

SDP also satisfies duality defined on the cone of semidefinite matrices. The general primal and dual SDP can be written as:

(Primal SDP)

$$\begin{aligned} \min_{\mathbf{X}} \quad & tr(\mathbf{CX}) \\ \text{s.t.} \quad & tr(\mathbf{A}_j\mathbf{X}) \leq b_j, \quad j = 1, ..., m \\ & \mathbf{X} \succeq 0 \end{aligned} \tag{4.4}$$

(Dual SDP)

$$\begin{aligned} \max_{\mathbf{y}} \quad & \mathbf{b}^T\mathbf{y} \\ \text{s.t.} \quad & \sum_{j=1}^{m} \mathbf{A}_j y_j \succeq \mathbf{C} \\ & \mathbf{y} \geq 0 \end{aligned} \tag{4.5}$$

Notice that the primal SDP has the same form as packing and the dual SDP has the same form as covering, except that positivity is replaced by semidefiniteness and vector-vector product is replaced by matrix-vector product. In fact, LP is a special case of SDP by restricting $\mathbf{X}$ to be diagonal.

There are two alternative paths extending the primal-dual algorithm to SDP: one on the primal problem eq. (4.4) [14, 2] and the other on the dual eq. (4.5) [3].

Extension to Primal

We consider MAXCUT for illustration purpose:

$$\begin{aligned} \min_{\mathbf{X} \in \mathbb{R}^{\mathbf{n} \times \mathbf{n}}} \quad & tr(\mathbf{CX}) \\ \text{s.t.} \quad & \text{diag}(\mathbf{X}) \leq 1 \\ & \mathbf{X} \succeq 0 \end{aligned} \tag{4.6}$$

For simplicity, the optimization problem is reduced to feasibility as in Chapter 3. First we bound the trace of the matrix $\mathbf{X}$ by $\sum_i X_i i \leq R$. For MAXCUT, R is equal to n. We map Algorithm 1 to primal SDP as follows.

- CONSTRAINT SET. $P = \{\mathbf{X} : \ \mathbf{X} \succeq 0, \ tr(\mathbf{X}) \leq R\}$ is the set of all semidefinite matrices whose trace is bounded by $R = n$.

- ORACLE. Optimize $\min_{\mathbf{X} \in P} y_0(tr(\mathbf{CX})/\alpha - 1) + \sum_j y_{j=1}^m [tr(\mathbf{A}_j\mathbf{X}) - b_j]$ given dual variables $y_0, y_1, ..., y_m$ corresponding to the $(m+1)$ constraints. Here the feasibility $tr(\mathbf{CX}) \leq \alpha$ is treated as a constraint. Solving the oracle amounts to computing the smallest eigenvector of matrix $y_0 \cdot \mathbf{C} + \sum_j y_j \mathbf{A}_j$.

- MULTIPLICATIVE WEIGHT UPDATE. $\mathbf{y}_j^{t+1} = \mathbf{y}_j^{t+1} \exp(\epsilon \mathcal{V}_j^t), \ (j = 0, ..., m)$ similar to line 7 in Algorithm 1.

The oracle can be implemented efficiently for large sparse matrices using Lanczos method [11]. The algorithm [2] uses the above reduction run in $\tilde{O}(n^{1.5} \cdot \min(N, n^{1.5}))$, where $\tilde{O}(f(n)) = O(f(n) \log^c f(n))$ for some c and N is the number of non-zero entries in $\mathbf{A}$.

Extension to Dual

The application of Algorithm 1 to the dual problem is even more powerful. CONSTRAINT SET and ORACLE become similar to packing and covering in the LP formulation, hence with much lower complexity. The difference is that MULTIPLICATIVE WEIGHT UPDATE needs to be generalize to matrix.

- CONSTRAINT SET. $P = \{\mathbf{y} : \ \mathbf{b^T y} \leq \alpha, \ \mathbf{y} \geq 0\}$.

- ORACLE. Optimize $\max_{\mathbf{y} \in P} \sum_j tr(\mathbf{A}_j\mathbf{X})y_j - tr(\mathbf{CX})$ given $\mathbf{X}$. This amounts to sorting analogous to eq. (3.12).

- MULTIPLICATIVE WEIGHT UPDATE. Compute $\mathbf{X}^{t+1} = \mathbf{X}^t \exp[\epsilon \sum_j (\mathbf{A}_j y_j^t - \mathbf{C})/\rho]$.

Now the complexity transfers to the step of multiplicative weight update on matrices. Exponential of a matrix $\mathbf{X}$ can be computed by

$$\exp(\mathbf{X}) = \sum_{k=0}^{\infty} \frac{\mathbf{X}^k}{k!}$$

where one can compute the series up to only a few term given the norm $\|\mathbf{X}\|$ is bounded. [3] utilizes random projection which enables computing $\exp(\mathbf{X})$ in $O^*(N)$. This also gives $O^*(N)$ for the overall algorithm, which is $O^*(n^{1.5})$ speedup compared to solving the primal problem for sparse matrices.

4.3 Boosting

Boosting, particularly AdaBoost [6, 7] has been a powerful learning technique in both theory and application. Given training examples, it trains a strong classifier by combining a set of weak classifiers in a weighted majority way [15]. To describe the setting, we first briefly review the PAC learning model [12].

PAC Learning. Let X be a set called the *domain*. A *concept* is a Boolean function $c: X \to \{0,1\}$. A learner have a sequence of training examples $(x, c(x))$ drawn from an arbitrary but unknown distribution $\mathcal{D}$ on X. The learner is asked to output a hypothesis $h: X \to \{0,1\}$, whose *error* is measured by $E_{x\mathcal{D}}(|h(x) - c(x)|)$. A *strong PAC learning algorithm* is one that given $\alpha, \gamma > 0$ and access to random examples, outputs with probability $(1-\gamma)$ a hypothesis whose error is at most α. A *weak PAC learning algorithm* is one that given $\beta, \gamma > 0$ outputs with probability $(1-\gamma)$ a hypothesis whose error is at most $1/2 - \beta$.

Boosting. If there exists weak learning algorithm for a collection of concepts, there exists a strong learning algorithm. Suppose there are m training examples (x_j, y_j) with labels $y_i \in \{0,1\}$. Boosting maintains weights $\mathbf{w}$ on $\mathbf{x}$. At each round t, it calls the weak learning algorithm on the distribution $\mathbf{w}^t$ of the training examples, and outputs a weak hypothesis $h^t(\mathbf{x})$ and error e_j^t on x_j. Weights are then increased when the examples are labeled incorrectly *i.e.* $e^t = 1$. The strong hypothesis equal to the majority voting from $h^1(\mathbf{x}), ..., h^T(\mathbf{x})$. It resembles to online prediction (3.3) with the following ingredients:

- Oracle A weak learning algorithm.
- Multiplicative Weight Update. $w_j^{t+1} = w_j^t \exp(\epsilon \cdot e_j^t)$.

Note that each example corresponds to "experts" in online prediction and the "events" are the weak hypotheses generated at each round.

We present a simple version of Boosting as follows.

Algorithm 3 Boosting

1: Initialize $w_j^0 = 1$ for samples $j = 1, ..., m$, $t = 0$.
2: Do
3: Set $p_j^t = w_j^t / \sum_j w_j^t$ for $j = 1, ..., m$.
4: Call oracle Weak Learning on the distribution $\mathbf{p}^t$. Output a hypothesis $h^t(\mathbf{x})$.
5: Run multiplicative weight update $w_j^{t+1} = w_j^t \exp[\epsilon \cdot e_j^t]$ with $e_j^t = |h^t(x_j) - y_j|$.
6: $t := t + 1$.
7: Until $t > T$.
8: Output final hypothesis $h_{final}(x_j) = \begin{cases} 1, & \sum_t h^t(x_j) \geq T/2; \\ 0, & \text{otherwise.} \end{cases}$

Proposition 2. *Algorithm 3 returns a final hypothesis h_{final} whose error is no greater than α with $\epsilon = \beta$, $T = \frac{4}{\beta^2} \log \frac{1}{\alpha}$.*

Proof. Let $\mathcal{V}_j^t = e_j^t$. By modifying the proof of Theorem 1, we derive the following inequality from eq. (3.7):

$$\Phi^T \leq m \cdot \exp[\epsilon \exp(\epsilon) \sum_t e^t] \leq m \cdot \exp[\epsilon(1+2\epsilon)T(\frac{1}{2} - \beta)] \tag{4.7}$$

because in each round the error e^t of the weak learning algorithm is at most $1/2 - \beta$. Note that $\exp(\epsilon) \leq 1 + \epsilon \exp \epsilon \leq 1 + 2\epsilon$ for $\epsilon \leq 1/\log 2$.

Let E be the set of examples that h_{final} misclassifies. $\sum_t e_j^t \geq T/2$ for $j \in E$ due to majority voting. From eq. (3.5), we have:

$$\Phi^T \geq \sum_j \exp(\epsilon \sum_t e_j^t) \geq \sum_{j \in E} \exp(\epsilon \sum_t e_j^t) \geq |E| \exp(\epsilon T/2) \tag{4.8}$$

Combining eq. (4.7) and (4.8) we obtain

$$|E| \exp(\epsilon T/2) \leq m \cdot \exp[\epsilon(1 + 2\epsilon)T(\frac{1}{2} - \beta)]$$

This means the error is no greater than α as desired:

$$\frac{|E|}{m} \leq \exp[\epsilon^2 T - \epsilon(1 + 2\epsilon)T\beta] \leq \exp(-\frac{\beta^2}{4}T) = \alpha$$

□

Algorithm 3 is called *hedge* in [6]. AdaBoost is an adaptive version where the step size ϵ in each round are adjusted according to the error of the weak hypothesis e^t (this is why it is called Adaptive Boosting). The underlying principle stays the same the proof here adapts to AdaBoost as well.

4.4 Zero-Sum Two Player Game

The online prediction strategy described in Theorem 1 as well as the min-max formulation eq. (2.6) has a very close connection to zero-sum two player matrix game.

Matrix Game. Consider a game defined by a playoff matrix $\mathbf{M} \in \mathbb{R}^{n \times m}$. A row player R and a column player C are competing in the game. The row player has n strategies represented by the rows of the matrix $\mathbf{M}$. Accordingly, the column player has m strategies from $\mathbf{M}$'s columns. When R chooses strategy i and C chooses strategy j to play the game, R obtains a playoff M_{ij} and C suffers a loss M_{ij}. The game is zero sum because the playoff of R exactly equals to the loss of C. Moreover, R can construct a mixed strategy specified by a distribution $\mathbf{x} \in \mathbb{R}^{n \times 1}$ on the rows ($\sum_i x_i = 1$, $\mathbf{x} \geq 0$). Similarly the mixed strategy of C is denoted as a distribution $\mathbf{y}$. When R and C play using the mixed strategies, the playoff is $\mathbf{x^T My}$. The goal of the row player R is to find the best $\mathbf{x}$, such that no matter how good the C's strategy is, R always obtain the biggest playoff. This worst case optimal playoff is computed by $\max_{\mathbf{x}} \min_{\mathbf{y}} \mathbf{x^T My}$. The column player C wants to optimize $\min_{\mathbf{y}} \max_{\mathbf{x}} \mathbf{x^T My}$, which is the minimum of the worst loss.

The min-max theorem [17] proves that these two optima are identical and the common value is call *the value of the game*:

$$v = \max_{\mathbf{x}} \min_{\mathbf{y}} \mathbf{x^T My} = \min_{\mathbf{y}} \max_{\mathbf{x}} \mathbf{x^T My} \tag{4.9}$$

We derive a procedure from Algorithm 1 to compute the value of a matrix game. Without loss of generality, we assume $M_{ij} \in [0, 1]$. Let strategies of R be experts and C be the adversarial world.

- Constraint Set. $P = \{\mathbf{y} : \sum_j y_j = 1\}$.
- Oracle. Given row strategy $\mathbf{x}^t$, find a column strategy $\mathbf{y}^t = \min_{\mathbf{y} \in P} (\mathbf{x^t})^\mathbf{T} \mathbf{M} \mathbf{y}$. This is simply the column $c^t = \arg\min_j (\mathbf{M^T x^t})_j$ with the minimal loss.
- Multiplicative Weight Update. R update the row strategies by $\mathbf{x}^{t+1} = \mathbf{x}^t \bullet \exp(\epsilon \cdot \mathbf{M y^T})$ ($\bullet$ is element-wise multiplication).

The game is played T rounds using the above procedure. We show that the average strategies $\bar{\mathbf{x}} = \frac{\sum_t \mathbf{x}^t}{T}$ and $\bar{\mathbf{y}} = \frac{\sum_t \mathbf{y}^t}{T}$ are near optimum.

Proposition 3. *The value of the game satisfies eq. (4.9). Following the above procedure, one can compute v within $(1 + \beta)$-approximation where $\beta = \epsilon \exp(\epsilon) + \frac{\log m}{\epsilon T}$.*

Proof. For clarity we define $v_{col} = \min_\mathbf{y} \max_\mathbf{x} \mathbf{x^T M y}$ and $v_{row} = \max_\mathbf{x} \min_\mathbf{y} \mathbf{x^T M y}$ and prove that $v_{row} = v_{col}$. From Corollary 2,

$$\begin{aligned}
v_{col} &= \min_\mathbf{y} \max_\mathbf{x} \mathbf{x^T M y} \\
&\leq \max_i (\mathbf{M}\bar{\mathbf{y}})_i \\
&\leq \exp(\epsilon) \cdot \frac{1}{T} \cdot \sum_t \min_\mathbf{y} (\mathbf{x^t})^\mathbf{T} \mathbf{M y} + \frac{\log m}{\epsilon T} \\
&\leq \exp(\epsilon) \cdot \frac{1}{T} \cdot \min_\mathbf{y} \sum_t (\mathbf{x^t})^\mathbf{T} \mathbf{M y} + \frac{\log m}{\epsilon T} \\
&= \exp(\epsilon) \cdot \min_\mathbf{y} \bar{\mathbf{x}}^\mathbf{T} \mathbf{M y} + \frac{\log m}{\epsilon T} \\
&\leq \exp(\epsilon) \cdot \max_\mathbf{x} \min_\mathbf{y} \mathbf{x^T M y} + \frac{\log m}{\epsilon T} \\
&\leq \exp(\epsilon) \cdot v_{row} + \frac{\log m}{\epsilon T}
\end{aligned}$$

The first inequality comes from the fact that given $\mathbf{y}$, the best playoff for the row strategy is $\max_\mathbf{x} \mathbf{x^T M}\bar{\mathbf{y}} = \max_i (\mathbf{M}\bar{\mathbf{y}})_i$. The second inequality is exactly Theorem **??**. The third inequality says the sum of minimum is not greater than the minimum of the sum. The fourth inequality is due to $\bar{\mathbf{x}}$ might not be optimal.

Similarly, we have

$$v_{row} \leq \exp(\epsilon) \cdot v_{col} + \frac{\log n}{\epsilon T}$$

By taking the limit $\epsilon \to 0$ and $\epsilon T \to \infty$, we conclude that $v_{row} = v_{col}$.

Finally, the procedure computes the game value with error not greater than

$$\exp(\epsilon) \cdot v + \frac{\log m}{\epsilon T} - v \leq \exp(\epsilon) - 1 + \frac{\log m}{\epsilon T} \leq \epsilon \exp(\epsilon) + \frac{\log m}{\epsilon T}$$

□

Running Time. To achieve $(1+\beta)$-approximation, we set $\epsilon = \beta/3$ and $T = \log m/(3\beta^2)$. The complexity of each iteration is dominated by matrix multiplication $\mathbf{M^T x}$ in the oracle denoted as T_{mult}. For a full playoff matrix $\mathbf{M}$, $T_{mult} = O(mn)$. The total running time is $O(mnT) = O(mn \log m/\beta^2)$. In [17], John Von Neumann gave a similar algorithm with complexity $O(m^2 n \log(mn)/\beta^2)$.

Chapter 5

Conclusion

We have surveyed several primal-dual combinatorial algorithms in a unified framework. The common characteristic of this class is that the primal and dual problems are tracked simultaneously, and one is used to bound and improve the other. The primal is improved by an oracle which enables fast approximate combinatorial subroutines. The dual evolves using multiplicative update from the feedback of the oracle. The overall complexity is very low in terms of problem size, and mostly dominated by the width and the approximation factor.

Originally designed for packing and covering, with simple elements or structured elements such as paths, primal-dual combinatorial algorithms have been applied extensively in many more problems beyond packing and covering. We show several representative examples in this paper and demonstrate how fast algorithms can be derived from the basic primal-dual framework. In many of the problems, the primal-dual algorithms achieve significant speedup in large-scale problems. We believe that the primal-dual framework is very promising as a general computational paradigm for many other problems.

Bibliography

[1] Sanjeev Arora, Elad Hazan, and Satyen Kale. $o(\sqrt{\log n})$ approximation to sparsest cut in $o(n^2)$ time. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 238–247, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *FOCS*, pages 339–348. IEEE Computer Society, 2005.

[3] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 227–236, New York, NY, USA, 2007. ACM.

[4] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *STOC*, pages 222–231, 2004.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, 2004.

[6] Freund and Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *JCSS: Journal of Computer and System Sciences*, 55, 1997.

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

[8] Garg and Konemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1998.

[9] Naveen Garg and Rohit Khandekar. Fractional covering with upper bounds on the variables: Solving lps with negative entries. In *ESA*, pages 371–382, 2004.

[10] M. X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[11] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[12] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT press, Cambridge, Massachusetts, 1994.

[13] Rohit Khandekar. *Lagrangian Relaxation Based Algorithms for Convex Programming Problems*. PhD thesis, IIT Delhi, 2004.

[14] Philip Klein and Hsueh i Lu. Efficient approximation algorithms for semidefinite programs arising from max cut and coloring. In *In Proceedings of the twenty-eighth annual ACM Symposium on the Theory of Computing*, pages 338–347, 1996.

[15] Littlestone and Warmuth. The weighted majority algorithm. In *FOCS: IEEE Symposium on Foundations of Computer Science (FOCS)*, 1989.

[16] Nesterov, Y. E. and Nemirovsky, A. S. *Interior Point Polynomial Methods in Convex Programming : Theory and Algorithms*. SIAM Publishing, 1993.

[17] John Von Neumann. Zur theorie der gesellshaftsspiele. *Mathematical Annals*, 100:295–320, 1928.

[18] Serge A. Plotkin, David B. Shmoys, and Eva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20:257–301, 1995.

[19] S. Plotkin C. Stein E. Tardos T. Leighton, F. Makedon and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *23rd Annual ACM STOC*, pages 101–111, 1991.

[20] Pravin M. Vaidya. A new algorithm for minimizing convex functions over convex sets. *Math. Program.*, 73(3):291–341, 1996.

[21] V. V. Vazirani. *Approximation Algorithms*. Springer, 2004.

[22] Stephen J. Wright. *Primal-dual interior-point methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[23] Neal E. Young. Randomized rounding without solving the linear program. In *SODA '95: Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 170–178, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.