CS 598CSC: Topics in Graph Algorithms	Lecture date: 10/22/2024
Instructor: Chandra Chekuri	Scribe: CC

1 Positive Linear Programs

Positive linear programs are a large and interesting class of linear programs that arise in combinatorial optimization. These have some useful properties that can be exploited algorithmically via iterative methods coming from first order optimization methods and this leads to fast approximation solutions. Although these methods are old in optimization the formal analysis of these methods with provable worst-case guarantees for concrete problems can be traced to the work of faster algorithms for multicommodity flows [SM90] which was later abstracted to general classes of linear programs [PST95, GK94]. Parallel algorithms [LN93] were also derived via these methods. There have been a large number of developments since then.

Positive linear programs, as the name suggests, are linear programs where the input to the linear program consists of positive numbers. This allows one to discuss relative approximations. There are three types of positive linear programs that we commonly work with.

- Packing LPs. A packing LP is of the form $\max c^T x$ subject to $Ax \leq b, x \geq 0$ where $c \in R^n_{\geq 0}$ and $A \in R^{m \times n}_{\geq 0}$ and $b \in R^m_{\geq 0}$. Notice that all coefficients are non-negative. There are n variables and m non-trivial constraints.
- Covering LPs are of the form $\min c^T x$ subjec to $Ax \geq b, x \geq 0$ where where $c \in R^n_{\geq 0}$ and $A \in R^{m \times n}_{\geq 0}$ and $b \in R^m_{\geq 0}$.
- Mixex Packing and Covering LPs are typically cast in tersm of feasibility of inequalities $Ax \leq b, Cx \leq d, x \geq 0$ where $A \in \mathbb{R}^{m_1 \times n}_{\geq 0}, b \in \mathbb{R}^{m_1}_{\geq 0}, C \in \mathbb{R}^{m_2 \times n}_{\geq 0}, d \in \mathbb{R}^{m_2}_{\geq 0}$. Objectives of max or min form can be converted to appropriate inequalities since packing and covering inequalities are allowed.

An *explicit* LP is one in which all the data is specified in terms of the variables, constraints and the non-negative entries in the inequalities. We will assume that the representation is given in the sparse form. We will usually let n denote the number of variables, m the number of constraints and N the number of non-zeroes in the matrices. Hence the input size of an explicitly specified positive LP is $\Theta(N + m + n)$.

An *implicit* LP is one in which we have an underlying data such as a graph or geometric object and an LP formulation that is specified implicitly based on that data. Many such LPs can have exponential number of variables or constraints and often they can be solved efficiently or approximately via the Ellipsoid method. In such settings we will not write down the explicit LP and we will seek iterative methods that have a running time that depends on the size of the original input that defines the LP. We give several examples below.

1.1 Examples

Max weight matching in bipartite graphs: Given a bipartite graph $G = (V, E, w)$ with non-negative edge weights the following LP gives an exact solution to the max weight bipartite matching

problem.

$$\begin{aligned} \max & \sum_{e \in E} w_e x_e \\ & \text{such that} \\ & \sum_{e \in \delta(u)} \leq 1 \quad u \in V \\ & x_e \geq 0 \quad e \in E \end{aligned}$$

One can see that this is a packing LP with $N = 2|E|$, $n = |E|$ and $m = |V|$. Note that the notation of n, m is confusing here from that of typical graph usage since we are using n for number of variables and m for number of constraints. Although this LP is mainly used for bipartite matching, it is sometimes useful even for non-bipartite graphs. One can show that its integrality gap is $2/3$ for general graphs.

Tree packing: We considered the tree packing LP previously. We want to pack spanning trees into the capacity of a given graph. This is an *implicit* packing LP with an exponential number of variables and $m = |E|$ constraints.

$$\text{maximize} \sum_{T \in \mathcal{T}} y_T \quad \text{subject to} \quad \sum_{T \ni e} y_T \leq c_e \quad \forall e \in E, \quad y_T \geq 0 \quad \forall T \in \mathcal{T}$$

Max multicommodity flow: We considered the maximum multicommodity flow problem as a relaxation for the Multicut problem. We are given graph $G = (V, E)$ with non-negative edge capacities $u : E \to \mathbb{Q}_+$ and k source-sink pairs $(s_1, t_1), \ldots, (s_k, t_k)$. We let $\mathcal{P}_i$ denote the set of all s_i-t_i paths in G. We have a variable x_p for each path $p \in \uplus_i \mathcal{P}_i$ to denote the amount of flow that is being routed on path p. We want to maximize the total flow sent between all pairs.

$$\begin{aligned} \max & \sum_{i=1}^{k} \sum_{p \in \mathcal{P}_i} x_p \\ & \text{such that} \\ & \sum_{i=1}^{k} \sum_{p \in \mathcal{P}_i : e \in p} x_p \leq u(e) \quad e \in E \\ & x_p \geq 0 \quad p \in \uplus_i \mathcal{P}_i. \end{aligned}$$

The above is also an *implicit* packing LP.

We can write this as an explicit LP with variables $f(e, i)$ which is the amont of flow on e for pair i. However, that LP will involve flow conservation constraints and will not be a positive LP. The disadvantage of the explicit LP is that it has a large number of variables and constraints and typical explicit LP solvers use memory that is qudratic in the number of variables to do matrix operations while solving the LP via the Simplex or interior point method, and this makes the memory a bottleneck.

Vertex Cover and Set Cover: We consider the well-known Set Cover problem. We are given a collection of n sets $S_1, S_2, \ldots, S_n$, each a subset of $\mathcal{U}$ consisting of m elements. Each set S_i has a cost c_i and the goal is to pick a min-cost sub-collection whose union is $\mathcal{U}$. We can define a $m \times n$ matrix A where $A_{ij} = 1$ if $j \in S_i$ and 0 otherwise. Then the LP relaxation for the Set Cover problem is of the form

$$\min c^T x \quad \text{subject to} \quad Ax \geq 1, x \geq 0$$

and it is known that its integrality gap is $O(\log m)$. The Vertex Cover problem is a further special case when we wish to cover edges of a graph with the vertices (thus each element in this system is in exactly two sets). The integrality gap of the LP for Vertex Cover is 2.

Covering Integer Programs with bound constraints: Covering Integer Programs (CIPs) generalize Set Cover. A CIP is essentially an integer version of a covering program.

$$\min c^T x \quad \text{subject to} \quad Ax \geq b, x \in \{0,1\}^n$$

We work with the LP relaxation which is a covering program. When A, b are arbitrary, the integrality gap of the natural LP for CIP can be as large as m. One needs to add Knapsack-Cover inequalities to strengthen the LP and then the LP becomes much more complex but nevertheless one can solve it fast using MWU methods. See [CQ19] and references to work on CIPs. One can also add bound constraint on CIPs where we can take a variable x_j only up to a bound d_i. Then we get packing constraints and the LP becomes a mixed packing and covering LP, albeit in a simple form.

$$\min c^T x \quad \text{subject to} \quad Ax \geq b, x \leq d, x \in \mathcal{Z}^n_+$$

Again the natural LP is not good in terms of integrality gap and one needs Knapsack cover inequalities.

Maximum concurrent flow: We saw the max concurrent flow LP in the context of the Sparsest Cut problem. See below. If we guess λ then it becomes a constant and we get a mixed packing and covering LP. We can do binary search for λ.

$$\begin{aligned}
& \max \lambda \\
\sum_{p \in \mathcal{P}_{s_i,t_i}} y_p & \geq \lambda D_i \quad i \in [k] \\
\sum_{i=1}^{k} \sum_{p \in \mathcal{P}_{s_i,t_i}, e \in p} y_p & \leq c_e \quad e \in E \\
y_p & \geq 0 \quad p \in \mathcal{P}_{s_i,t_i}, i \in [k]
\end{aligned}$$

Densest Subgraph: Another interesting mixed packing and cover LP comes up when solving the Densest Subgraph problem. We refer the reader to [BGM14].

1.2 Known approximation results for explicit packing LPs

There are several clean results known for explicit positive LPs. We refer the reader to [Qua19, Wan17] for a good overview. Here we state a few high-level results for sequential models.

- For mixed and packing covering LPs there is an MWU based algorithm that runs in deterministic $O(N \log N/\epsilon^2)$ time and outputs a $(1-\epsilon)$-approximate feasible solution [You14]. This implies, as a corollary $(1-\epsilon)$-approximation algorithms for packing and covering in $O(N \log N/\epsilon^2)$-time.
- For packing there is a randomized algorithm that yields a $(1-\epsilon)$-approximation in $O(N \log N \log(1/\epsilon)/\epsilon)$ time [AZO15].
- A similar time bound as above for covering [WRM15].

2 Multiplicative Weight Updates Method

The MWU method is a meta-algorithm that has many applications and arises in a number of areas even though the central analysis approach is very similar. The survey of Arora, Hazan and Kale [AHK12] outlines the utility of seeing the general approach. Here we are interested in applications of MWU to solving positive LPs. Although one can derive some of these via the experts framework outlines in [AHK12] there are some limitations as well as the overhead of introducing the experts framework.

Even within offline optimization, the use of MWU is varied and it is not always easy to figure out the similarities and differences even among closely related papers. We will follow the ideas and exposition from [CJV15].

2.1 Background

We consider a more abstract problem involving convex functions since it makes some of the ideas and notation cleaner and more general, and also indicates where we use linearity and positivity when we want refined results. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a concave function and let $g_i : \mathbb{R}^n \to \mathbb{R}$, $i \in [m]$ be a set of convex functions. Let P be a convex set in $\mathbb{R}^n$. We will assume that all functions are continuous and bounded in the domain of interest, and ignore some of the technical difficulties that arise when considering unbounded functions/domains. Even though we typically use P as the non-negative orthant in $\mathbb{R}^n$, we can usually restrict the domain to be a box of sufficiently large size in the non-negative orthant. Consider the following optimization problem which generalizes the setting of packing LPs.

$$\begin{aligned} \max f(x) & \\ g_i(x) &\leq 1 \quad i \in [m] \\ x &\in P \end{aligned}$$

The preceding problem is a constrained convex optimization problem (we can think of maximizing f as minimizing $-f$ which would make the objective also convex). Unconstrained convex

minimization is itself a non-trivial problem and is typically addressed via gradient descent, cutting planes, and other methods. Here we use it as a modeling tool for explanation purposes and eventually work with linear functions.

The soft-max function: One way to view the multiple constraints is to think of it as a single constraint $h(x) \leq 1$ where $h(x) = \max_{i \in [m]} g_i(x)$. Note that $h(x)$ is also a convex function but it is not smooth even when the g_i are. One can make it smooth by considering the so-called soft-max function defined as follows. For a parameter $\eta > 0$,

$$g_\eta(x) = \frac{1}{\eta} \ln(\sum_i \exp(\eta g_i(x)))$$

Claim 1. *For all x, $h(x) \leq g_\eta(x) \leq h(x) + \frac{\ln m}{\eta}$.*

It is known that g_η is convex. By choosing $\eta = (\ln m)/\epsilon$ we can get a smooth proxy for $h(x)$ that has an additive error of at most ϵ.

Lagrangean relaxation: A standard approach in continuous (convex and non-linear) optimization is to replace a constrained problem by its Lagrangean relaxation. In the context of the above, we would replace the constrained optimization problem by the following concave optimization problem:

$$\max_{x \in P} f_w(x) = \max_{x \in P} f(x) - \sum_{i \in [m]} w_i (g_i(x) - 1)$$

where $w_1, w_2, \ldots, w_m \geq 0$ are Lagrange multipliers. The idea is that one penalizes the objective for violating the constraints.

We have the following easy weak-duality claim.

Claim 2. *For any non-negative weights $w_1, \ldots, w_m$, $\max_{x \in P} f_w(x) \geq$ OPT where* OPT *is the optimum value of the original optimization problem. Thus, $\min_{w \geq 0} \max_{x \in P} f_w(x) \geq$ OPT.*

For convex optimization problems, the duality theorem says that under some mild conditions usually called Slater conditions, strong-duality holds, that is: $\min_{w \geq 0} \max_{x \in P} f_w(x) =$ OPT.

Although Lagrangean relaxation is nice, it is not obvious how to find good weights that lead to good upper and lower bounds. One can combine the idea of soft-max function and Lagrangean relaxation to reduce the problem to have a single constraint, namely $g_\eta(x) \leq 1$. Thus we want to solve the problem $\max_{x \in} f(x)$ subject to $g_\eta(x) \leq 1$. One can use some gradient descent like approach for solving this problem and the gradient of $g_\eta(x)$ is relevant in this context. We will elaborate on this at a later time.

Oracle: Motivated by the Lagrangean relaxation approach we will assume that we have a black-box oracle for the following problem for any given non-negative weights $w_1, w_2, \ldots, w_m$.

$$\max_{x \in P} f(x) \text{ subject to } \sum_{i=1}^{m} w_i g_i(x) \leq \sum_{i=1}^{m} w_i.$$

We will see later that there is a simple and easy oracle for the above problem when f and g_i are positive linear functions and P is the non-negative orthant.

2.2 From discrete to continuous and back

We consider an adaptive iterative algorithm that uses a notion of time that goes from 0 to 1. From a discrete point of view we can think of taking T steps of size $\delta = \frac{1}{T}$ where δ is small. Thus, after ℓ steps the time is $t = \ell\delta$. Later, for ease of analysis we will make this a continuous process. The algorithm will maintain a set of non-negative weights $w_i(t)$, one for each constraint. The non-negative weights represent the relative importance of each constraint at time t. Initially we have no information so all weights are equally important and we set $w_i(t) = 1$. An alternative view is to think of the normalized weights $\frac{w_i(t)}{\sum_\ell w_\ell(t)}$ as a probability distribution on the constraints. In each step we will use the oracle to compute a solution $v(t)$ for the Lagrangean relaxation defined by the weights. Clearly $f(v(t)) \geq \text{OPT}$ since we are solving a relaxation. At the end we will take an average, or convex combination, of these solutions. The concavity of f will guarantee that we are doing well on the objective value. We let $x(t) = \int_0^t v(t)$ denote the evolving solution, as an accumulation of the solutions computed so far with initial point at $t = 0$ being $x(0) = 0$. The main question is to deal with the constraint violation. For this we will maintain the invariant that $w_i(t) = \exp(\eta \int_0^t g_i(v(t))$ which tries to keep track of the violation of constraint i.

MWUUniformStep$(f, g_1, \ldots, g_m, P, \eta, \delta)$:
  $w_i(0) = 1$ for $i \in [m]$
  $x(0) = 0$
  $t = 0$
  While $t < 1$ do
    Use oracle with weights to find optimum solution $v(t)$
    For $i \in [m]$ do
      $w_i(t + \delta) = w_i(t)e^{\eta\delta g_i(v(t))}$
    $x(t + \delta) = x(t) + \delta v(t)$
    $t = t + \delta$
  endWhile
  return $x_{out} = x(1)$.

Suppose we make $\delta \to 0$. Then we obtain a continuous time algorithm whose analysis can be done simply and is illuminating. For that we set up a differential equation on the evolution of the weights. The algorithm sets $w_i(t + \delta) = w_i(t)e^{\eta\delta g_i(v(t))}$. If we make the step size go to 0,

$$\frac{dw_i(t)}{dt} = \lim_{\delta\to 0} \frac{w_i(t + \delta) - w_i(t)}{\delta} = \lim_{\delta\to 0} \frac{w_i(t)(e^{\eta\delta g_i(v(t))} - 1)}{\delta} = w_i(t)\eta g_i(v(t))$$

by using the fact that $e^x - 1 \to x$ when $x \to 0$.

This leads to the following continuous-time algorithm.

MWUContinuous$(f, g_1, \ldots, g_m, P, \eta)$:
  $w_i(0) = 1$ for $i \in [m]$
  For $t = 0$ to 1 do continuously
    Use oracle with weights to find optimum solution $v(t)$
    For $i \in [m]$ do
      $\frac{dw_i(t)}{dt} = w_i(t)\eta g_i(v(t))$
  endWhile
  return $x_{out} = \int_0^t v(t)$.

We now analyze the continuous algorithm.

The first claim shows the optimality of the output.

Claim 3. *Suppose f is concave. Then $f(x_{out}) \geq \text{OPT}$.*

Proof. By concavity, $f\left(\int_0^1 v(t)dt\right) \geq \int_0^1 f(v(t))dt$. And $f(v(t)) \geq \text{OPT}$ since $v(t)$ is an optimum solution to a Lagrangean relaxation of the original problem. This gives the desired claim. ■

We now express the constraint values in terms of the weights.

Claim 4. *For each $i \in [m]$, $g_i(x_{out}) \leq \frac{1}{\eta}\ln(w_i(1))$.*

Proof. By convexity $g\left(\int_0^1 v(t)dt\right) \leq \int_0^1 g(v(t))dt$. From the algorithm $\frac{dw_i(t)}{dt} = \eta w_i(t)g_i(v(t))$. Hence,

$$\int_0^1 g(v(t))dt \leq \frac{1}{\eta}\int_0^1 \frac{dw_i(t)}{dt}\frac{1}{w_i(t)}dt \leq \int_0^1 \frac{dw_i(t)}{w_i(t)} = \ln w_i(1) - \ln w_i(0).$$

We have $\ln w_i(0) = 0$ since $w_i(0) = 1$. ■

We do not have a direct way of bound $w_i(1)$ but we will bound $w_i(1)$ by $\sum_i w_i(1)$. For this we wil see how the sum of weights evolves.

Claim 5. *Let $w(t) = \sum_i w_i(t)$. Then $w(t) \leq e^{\eta t}w(0)$. Hence $w(1) \leq me^{\eta}$.*

Proof. Here is where we use the fact that we solve the Lagrangean relaxation. Note that $v(t)$ satisfies the constraint that $\sum_i w_i(t)g_i(v(t)) \leq \sum_i w_i(t)$. We have $\frac{dw_i(t)}{dt} = \eta w_i(t)g_i(v(t))$ for $i \in [m]$. Hence,

$$\frac{dw(t)}{dt} = \sum_i \frac{dw_i(t)}{dt} = \eta\sum_i w_i(t)g_i(v(t)) \leq \eta\sum_i w_i(t) = \eta w(t).$$

Thus the total weight satisfies the differential equation $\frac{dw(t)}{dt} \leq \eta w(t)$ which implies that $w(t) \leq e^{\eta t}w(0) = e^{\eta}m$. ■

Putting together the claims we obtain the following lemma.

Lemma 6. *The output of* MWUContinuous*, x_{out}, satisfies the property that (i) $f(x_{out}) \geq \text{OPT}$ and (ii) $g_i(x_{out}) \leq 1 + \frac{\ln m}{\eta}$ for all $i \in [m]$. If there is a point $x_0 \in P$ such that $g_i(x_0) = 0$ for all $i \in [m]$ then $x'_{out} = \theta x_{out} + (1-\theta)x_0$ is feasible, in that $g_i(x'_{out}) \leq 1$ for all i and $x'_{out} \in P$.*

Proof. The only thing to verify is that $g_i(x_{out}) \leq 1 + \frac{\ln m}{\eta}$ which follows from

$$g_i(x_{out}) \leq \frac{1}{\eta} \ln(w_i(1)) \leq \frac{1}{\eta} \ln(w(1)) \leq \frac{1}{\eta} \ln(e^{\eta} m) \leq (1 + \frac{\ln m}{\eta}).$$

The second part of the lemma is easy to verify. ■

Remark 7. *The second part of the preceding lemma allows us to obtain a feasible solution x'_{out} since x_{out} does not necessarily satisfy the constraints. The impact of using x'_{out} instead of x_{out} in terms of the objective function f is less easy to see. However, for non-negative linear objectives we see that $f(x'_{out}) \geq \theta f(x_{out})$ and thus we get an approximation to the objective while making the solution feasible.*

Remark 8. *To obtain a $(1+\epsilon)$ in the constraint violation we can choose $\eta = \frac{\ln m}{\epsilon}$.*

Remark 9. *The analysis of* MWUCONTINOUS *does not use non-negativity of f or that of the g_is. Hence the analysis can also be applied to have constraints of the form $h_i(x) \geq 1$ where h_i is concave which can be modeled as a convex constraint $-h_i(x) + 2 \leq 1$.*

Remark 10. *Our analysis relied on keeping track of the total weight $w(t)$ in a careful manner. In some works, the potential function $\phi(t) = \frac{1}{\eta} \ln w(t)$ or proxies for it are used. Potential functions can be powerful tools that can yield strong results though they can sometimes be mysterious. See [BG17] for potential function analysis of first order methods in optimization.*

Duality: One can prove a duality theorem via the continuous argument. For packing LPs this also allows one to recover an approximate dual LP solution form the MWU algorithm. Details TODO.

2.3 Analysis of Discretized Algorithms For Packing Problems

In this section we want to obtain algorithm that terminate in a small number of iterations. For this we will work with the extra condition that $g_i(x) \geq 0$ for all $x \in P$. This models the packing constraints of interest and is needed for the algorithms/analysis unlike the continuous algorithm.

Width dependent time bound for uniform step size: We first consider the simple uniform step size algorithm MWUUNIFORMSTEP. We will assume for simplicity that δ is chosen such that $T = 1/\delta$ is an integer. Thus, the algorithm runs for T time steps. How small can we make T (or how large can we make δ) such that the discrete implementation ensures that the constraints are approximately satisfied? It turns out that we can use a parameter called the *width* of the given instance defined as follows.

Definition 1. *The width ρ of the given instance is* $\sup_{x \in P, i \in [m]} g_i(x)$.

In other words it is asking how much can some arbitrary point in P violate a specific constraint ((in a relative sense since we have normalize the rhs of each constraint to 1 by considering $g_i(x) \leq 1$ as the constraint). Why is this parameter natural to consider in the context of the algorithm we discussed. Recall that we solve a Lagrangean relaxation in each step and find a point $v(t)$. $v(t)$ can

violate some specific constraint i by a significant factor since we are only solving the relaxation by taking weighted sum of the constraints. ρ allows us to bound the worst-case violation.

It may seem that ρ is unbounded for even simple instances because g_i may be an increasing function and P is typically the non-negative orthant. However, for many problems of interest there are implicit bounds on the variables and P can be implicitly taken to be a bounding box that restrict ρ. We will give some examples later on to illustrate this. Suppose for now that ρ is some bounded number.

Theorem 11. *Suppose we set* $\eta = \frac{\ln m}{\epsilon}$ *and* $\delta = \frac{\epsilon}{\eta} \frac{1}{\rho}$ *and* $\epsilon \in (0, 1/2)$. *Then the output of* MWU-UniformStep *will output a point* x_{out} *such that (i)* $f(x_{out}) \geq \mathrm{OPT}$ *and (ii)* $g_i(x_{out}) \leq 1 + 2\epsilon$ *for all* $i \in [m]$. *The algorithm terminates in* $O(\frac{\rho \ln m}{\epsilon^2})$ *iterations.*

We will derive the analysis of the preceding theorem in the context of the non-uniform step size analysis. Notice that the dependence on ϵ is now $1/\epsilon^2$ when compared to the continuous algorithm. One can view this factor as the price for discretization since we cannot follow the continuous algorithm precisely.

Non-uniform step size and a width independent algorithm and analysis: Garg and Konemann [GK07] obtained a width-independent algorithm and analysis by a seemingly simple but important tweak to the algorithm. The motivation for the non-uniform step-size would be more clear if one does the uniform step size analysis first but we will simply mention the choice of the step and ask the reader to do the uniform step size analysis post-facto. The non-uniform step size can be thought of as choosing the maximum step size at any point greedily so that the analysis goes through. Surprisingly this yields an algorithm that bounds the number of iterations to $O(m \log m/\epsilon^2)$ which depends only on the number of constraints and ϵ. This is the reason this is called width-independent. Note that step size choice requires each g_i to be postive over the domain P, the algorithm is not well-defined otherwise.

MWUNonUniformStep$(f, g_1, \ldots, g_m, P, \eta)$:
  $w_i(0) = 1$ for $i \in [m]$
  $x(0) = 0$
  $t = 0$
  While $t < 1$ do
    Use oracle with weights $w_i(t)$ to find optimum solution $v(t)$
    Let δ be max step size such that $\delta \eta g_i(v(t)) \leq \epsilon$ for all $i \in [m]$
    $\delta = \min(\delta, 1 - t)$
    For $i \in [m]$ do
      $w_i(t + \delta) = w_i(t) e^{\eta \delta g_i(v(t))}$
    $x(t + \delta) = x(t) + \delta v(t)$
    $t = t + \delta$
  endWhile
  return $x_{out} = x(1)$.

We will mimic the continuous-time analysis in some ways and point out where the discretization matters and how we bound the errors. First, we set up some notation. The algorithm runs for some

number T of iterations; we will use j to index the iterations and i as before to index the constraints. We think of the time before the start of iteration $j+1$ as t_j and the step size in that iteration as δ_j. We have $t_0 = 0$ and $t_{j+1} = t_j + \delta_j$ for $j = 0, 2, \ldots, T-1$. We have $\delta_0 + \delta_1 + \ldots + \delta_{T-1} = 1$. Note that in iteration $j+1$ we compute $v(t_j)$ and δ_j with respect to the weights $w_i(t_j)$. In other words t_{j+1} is the time at the end of iteration $j+1$.

The following is easy to prove even with a discrete step size.

Claim 12. *Suppose f is concave. Then $f(x_{out}) \geq \text{OPT}$.*

The following claim still holds in the discrete setting but the proof is useful to see.

Claim 13. *For each $i \in [m]$, $g_i(x_{out}) \leq \frac{1}{\eta} \ln w_i(1)$.*

Proof. We have $x_{out} = \sum_{j=0}^{T-1} \delta_j v(t_j)$ and hence by convexity of g_i, $g(x_{out}) \leq \sum_{j=1}^{T-1} \delta_j g(v(t_j))$. We see that in iteration $j+1$, the weight of constraint i is updated as follows:

$$w_i(t_{j+1}) = w_i(t_j) e^{\eta \delta_j g_i(v(t_j))}$$

Hence

$$\delta_j g_i(v(t_j)) = \frac{1}{\eta} \left(\ln w_i(t_{j+1}) - \ln w_i(t_j) \right).$$

Thus, summing up the two sides over j and using telescoping implies that

$$g(x_{out}) \leq \sum_{j=0}^{T-1} \delta_j g(v(t_j)) \leq \frac{1}{\eta} \left(\ln w_i(1) - \ln w_i(0) \right).$$

■

The main technical claim is the one below that bounds the evolution of the total weight and this is where the choice of the step size is crucial.

Claim 14. *Let $w(t_j) = \sum_i w_i(t_j)$. Then $w(t_j) \leq e^{(1+\epsilon)\eta t_j} w(0)$. Hence $w(1) \leq m e^{(1+\epsilon)\eta}$.*

Proof. Here is where we use the fact that we solve the Lagrangean relaxation.

We ensured that $\eta \delta_j g_i(v(t_j)) \leq \epsilon$ by our choice of δ_j. Note also that all the numbers involved are postive since g_i is positive on the domain as per our assumption. And $\epsilon < 1/2$. We use the fact that $e^a \leq 1 + a + a^2$ when $a \in (0, 1/2)$. For ease of notation let $a_i = \eta \delta_j g_i(v(t_j))$. Thus, $w(t_{j+1}) \leq w_i(t_j)(1 + a_i + a_i^2) \leq w_i(t_j)(1 + a_i(1+\epsilon))$. Summing up over i,

$$\begin{aligned}
\sum_i w_i(t_{j+1}) &\leq \sum_i w_i(t_j)(1 + a_i(1+\epsilon)) \\
&\leq \sum_i w_i(t_j) + (1+\epsilon)\eta\delta_j \sum_i w_i(t_j) g_i(v(t_j)) \\
&\sum_i w_i(t_j) + (1+\epsilon)\eta\delta_j \sum_i w_i(t_j) \\
&\leq \sum_i w_i(t_j)(1 + (1+\epsilon)\eta\delta_j).
\end{aligned}$$

We now use the fact that $1 + b \leq e^b$ for all $b \geq 0$ to derive

$$\sum_i w_i(t_{j+1}) \leq e^{1+\epsilon)\eta\delta_j} \sum_i w_i(t_j).$$

Note that we now have essentially derived an exponential increase in the total weight in terms of the step size δ_j. We thus rewrite the above as $w(t_{j+1}) \leq e^{(1+\epsilon)\eta\delta_j} w(t_j)$ where w is the sum of weights. The rest follows by summing over the steps. ■

Remark 15. *We note that the proof of the preceding theorem will go through easily for a fixed step size algorithm as long as the step size guarantees that no weight of a constraint goes up by more than an e^ϵ factor. This is satisfied by the choice of the step size based on the width! The rest of the analysis is the same.*

Theorem 16. *Let x_{out} be the output of* MWUNonUniformStep. *If $\eta \geq \frac{\ln m}{\epsilon}$ then (i) $f(x_{out}) \geq$ OPT and (ii) $g_i(x_{out}) \leq 1 + 2\epsilon$ for all $i \in [m]$. Moreover, the algorithm terminates in $O(\frac{m \log m}{\epsilon^2})$ iterations.*

Proof. We already established (i). For (ii) we saw that $g_i(x_{out}) \leq \frac{1}{\eta} \ln w_i(1)$ and that $\ln w(1) \leq (1 + \epsilon)\eta \ln m$. Thus,

$$g_i(x_{out}) \leq \frac{1}{\eta} \ln w_i(1) \leq \frac{1}{\eta} \ln w(1) \leq (1 + \epsilon) + \frac{\ln m}{\eta} \leq (1 + 2\epsilon).$$

Here, we switch to exponential weights instead of reasoning with logs since it will be useful later. We will also use a loose argument though one can do a tighter analysis. We see that the total weight $w(1)$ at the end is $e^{(1+\epsilon)\eta} m \leq m^{1+(1+\epsilon)/\epsilon} = m^{O(1/\epsilon)}$. Now consider the greedy step size. In each iteration since we choose δ_j maximum, there is some i in that iteration where $\eta\delta_j g_i(v(t_j)) = \epsilon$. But this implies that for that i its weight w_i increases by e^ϵ multiplicative factor. Note that each variables weight starts at 1. We established that total weight $w(1)$ is $m^{O(1/\epsilon)}$. This implies that no weight w_i can have its weight updated by an e^ϵ factor more than $O(\log m/\epsilon^2)$ times. But there are only m constraints and each iteration must increase at least one constraint's weight by a multiplicative factor of e^ϵ. Thus, we cannot have T more than $cm \log m/\epsilon^2$ for some small but fixed constant c that we leave the reader to figure out. ■

Scaling weights: In the analysis so far we have used constraints of the form $g_i(x) \leq 1$ which makes the notation clean. Of course any constraint of the form $g_i(x) \leq b_i$ with $b_i > 0$ can be scaled to achieve the standard form. However, in some combinatorial applications, when A the packing matrix corresponds to an incidence matrix of a combinatorial object and b_i represents some capacity, scaling makes the matrix unnatural. We can avoid this by modifying the MWU algorithm as follows. Suppose we have constraints of the form $g_i(x) \leq b_i$, $i \in [m]$ where $b > 0$. We start with $w_i(0) = \frac{1}{b_i}$ for each i. Then we do the analysis with $b_i w_i$ instead of w_i. Since the weights are updated in a multiplicative fashion the whole analysis goes through cleanly. Note that with weights set up like this we still solve the oracle in each iteration with the constraint $\sum_i w_i g_i(y) \leq \sum_i w_i$; the b_i's are taken care of automatically.

3 Applications to Packing LPs, explicit and implicit

We show how the generic scheme that we have described via MWU can be used to derive fast approximation algorithms for packing LPs. Recall that a packing LP is of the form

$$\max c^T x \quad \text{such that} Ax \leq b, x \geq 0$$

In terms of the generic framework we have $f(x) = c^T x$ and $g_i(x) = \sum_{j=1}^{n} A_{ij} x_j$ corresponds to the i'th row of A. And P corresponds to $R^n_{\geq 0}$. In some settings we think of P as a bounding box implied by constraints with $0 \leq x_j \leq u_j$ where u_j is an upper bound on x_j implicitly implied by the constraints in of $Ax \leq b$. This often helps in figuring out the width of the system.

Oracle is efficient: Recall that we need an oracle that solve $\max f(x)$ subject to $\sum_i w_i g_i(x) \leq \sum_i w_i$ and $x \geq 0$. This is the same as $\max \sum_j c_j x_j$ subject to $\sum_j \alpha_j x_j \leq 1$ and $x \geq 0$, where $\alpha_j = \frac{1}{\sum_i w_i} \sum_i w_i A_{ij}$. Note that $\alpha_j \geq 0$ since all entries are positive. This is a continuous Knapsack problem and an optimum solution is to pick the coordinate $j^* = \arg\max_j c_j/\alpha_j$ and set $x_{j^*} = 1/\alpha_{j^*}$ and the rest of the values to 0. An important aspect of this solution is that there is an optimum solution x^* to the oracle whose support is a *single coordinate*. Note that the width-independent analysis shows that the number of iterations depends only on the number of constraints m. Thus, we can apply the method to implicit LPs with exponential (in some original problem size) number of variables but a polynomial number of constraints as long as we can implement the oracle efficiently.

It is instructive to see what the oracle corresponds to when considering combinatorial problems.

Obtaining feasible solution via scaling down: Recall that the MWU method, as we described it, naturally gives an optimum solution that violates the constraint. We can then scale the solution down to obtain a feasible solution while losing a bit in the objective. This is particularly easy for packing problems with linear objectives. Thus we can obtain a $(1-\epsilon)$-approximation in the objective while obtaining a feasible solution.

Approximate oracle: One of the nice aspects of the MWU method is that it accumulates a series of solutions of Lagrangean relaxations by taking non-negative sums. In the positive settings this means that we do not use subtraction at all. This is convenient for approximation. Suppose we only have an α-approximate oracle in the relative sense. Then the process yields an $(1-\epsilon)\alpha$-approximate solution. This is convenient not only in obtaining approximate solutions but also in speeding up MWU based algorithms substantially by using various tricks and data structures that can avoid updating information at every step since we are allowed to use an approximate oracle. Note that if $\alpha = (1-\epsilon)$ then $(1-\epsilon)\alpha \geq 1 - 2\epsilon$.

Interpreting the weights as exponential in "loads": Note that in the MWU method we are maintaining $w_i(t)$ as $\exp(\eta \int_0^t g_i(v(t))$. When g_i is linear and P is the non-negative orthant, $\int_0^t g_i(v(t))$ is simply $\sum_j A_{ij} x(t)_j$ where $x(t)$ is the current accumulated vector. We think of this as the total load $\ell_i(t)$ on constraint i at time t and hence $w_i(t) = \exp(\eta \ell_i(t))$.

3.1 Explicit Packing LPs

If we use the method the algorithm takes $O(m \log m/\epsilon^2)$ iterations. The oracle for each iteration can be done in $O(N)$ time easily and naively. Thus we can obtain a $(1-\epsilon)$-approximate solution in $O(mN \log m/\epsilon^2)$ time. By being slightly careful and using a simple approximate oracle we can reduce the running time to $O(N \log m/\epsilon^2)$ which is near-linear. We will see it later in the notes.

3.2 Max weight matching

Consider the max-weight matching problem in bipartite graphs. The variables x_e correspond to the edges and the constraints correspond to the vertices. Thus the MWU method maintains a weight w_u for each $u \in V$. What is the oracle? Given vertex weights w_u it finds the edge $uv \in E$ that maximizes $c_e/(w_u + w_v)$ in each step. Note that updating the weights is simple fast because we are only touching two vertex weights. The bottleneck is finding the edge with the maximum ratio in each iteration.

We can explore simple tricks that will help speed this up. First, we bucket edges into logarithmic groups by considering weights in powers of $(1+\epsilon)$; this affects the approximation factor only by a $(1-\epsilon)$-factor. Within each bucket we are trying to find the weight with the minumum weight where the weight of an edge is the sum of the weights of its end points. We can keep a priority queue for edges based on their weight. How do we update these weights? Suppose in teration j of the algorithm we pick edge $e_j = (u, v)$. The algorithm updates w_u and w_v. Now we need to update the weights of edges that are incident to u and v and these will affect the priority queues. How many edges do we need to update? It is $deg(u) + deg(v)$. This seems like a lot but we make a crucial observation. Recall that the width-independent MWU analysis proves that the weight of any specific constraint i is updated only $O(\log m/\epsilon^2)$ times! In particular the weight of a vertex u is updated only $O(\log n/\epsilon^2)$ times. Hence the total number of updates is only $O(\sum_u deg(u) \log n/\epsilon^2) = O(|E| \log n/\epsilon^2)$! The priority queue operations have only a logarithmic overhead. Thus we can see that with some basic observations we can implement the MWU algorithm in $\tilde{O}(|E|/\epsilon^2)$ time.

3.3 Tree Packing

Now we consider the tree packing LP. This is an implicit LP with an exponential number of variables but only m constraints where $m = |E|$. Note that the coefficient of each variable x_T in the objective is 1. In this example the right hand side of the constraint corresponding to edge e is its capacity $u(e)$ and these are non-uniform. It now makes sense to use weights that start with $w(e) = \frac{1}{u(e)}$. What is the oracle? Given weights $w(e)$ on the edges we wish to find a tree T that maximizes $\frac{1}{\sum_{e \in T} w(e)}$. Aha, this corresponds to finding the MST with respect to weights! Recall that MST is the separation oracle for the dual LP! As you can see, the MWU oracle turns out to be the dual LP separation oracle. Thus, each iteration of the algorithm simply needs to solve an MST problem and update the weights of the edges in the chosen tree T. There are only $O(m \log m/\epsilon^2)$ iterations. We will see later how to speed up the entire computation to run in $\tilde{O}(m/\epsilon^2)$ time.

3.4 Maximum Multicommodity Flow

Now we consider the maximum multicommodity flow. Once again we have an implicit packing LP with an exponential number of variables corresponding to the paths for all the commodity pairs. The number of constraints is m corresponding to the edges. The weights correspond to exponentials of the loads on the edges where the load on an edge is the total flow routed so far on the edge. What is the oracle? It is to find the shortest path among all the commodity pairs, according to the weights on edges! We can find for each pair (s_i, t_i) a shortest s_i-t_i path and choose the minimum. The algorithm routes some flow on that path and updates the weights and iterates. The bottleneck is to compute the shortest paths.

4 Covering and Mixed Packing and Covering

TODO

5 Faster Implementations via Tricks and Data Structures

TODO. See [Qua19, Wan17] and some papers.

References

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of computing*, 8(1):121–164, 2012.

[AZO15] Zeyuan Allen-Zhu and Lorenzo Orecchia. Nearly-linear time positive lp solver with faster convergence rate. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 229–236, 2015.

[BG17] Nikhil Bansal and Anupam Gupta. Potential-function proofs for first-order methods. *arXiv preprint arXiv:1712.04581*, 2017.

[BGM14] Bahman Bahmani, Ashish Goel, and Kamesh Munagala. Efficient primal-dual graph algorithms for mapreduce. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 59–78. Springer, 2014.

[CJV15] Chandra Chekuri, TS Jayram, and Jan Vondrák. On multiplicative weight updates for concave and submodular function maximization. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 201–210, 2015.

[CQ19] Chandra Chekuri and Kent Quanrud. On approximating (sparse) covering integer programs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1596–1615. SIAM, 2019.

[GK94] Michael D Grigoriadis and Leonid G Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM Journal on Optimization*, 4(1):86–107, 1994.

[GK07] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

[LN93] Michael Luby and Noam Nisan. A parallel approximation algorithm for positive linear programming. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 448–457, 1993.

[PST95] Serge A Plotkin, David B Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Mathematics of Operations Research*, 20(2):257–301, 1995.

[Qua19] Kent Quanrud. *Fast approximations for combinatorial optimization via multiplicative weight updates*. PhD thesis, University of Illinois at Urbana-Champaign, 2019.

[SM90] Farhad Shahrokhi and David W Matula. The maximum concurrent flow problem. *Journal of the ACM (JACM)*, 37(2):318–334, 1990.

[Wan17] Di Wang. *Fast approximation algorithms for positive linear programs*. PhD thesis, University of California, Berkeley, 2017.

[WRM15] Di Wang, Satish Rao, and Michael W Mahoney. Unified acceleration method for packing and covering problems via diameter reduction. *arXiv preprint arXiv:1508.02439*, 2015.

[You14] Neal E Young. Nearly linear-work algorithms for mixed packing/covering and facility-location linear programs. *arXiv preprint arXiv:1407.3015*, 2014.