A Distributed Solver for Massive Scale Resource Allocation Linear Programs

MohammadHossein Bateni
Google Inc.

Dragos Florin Ciocan
Massachusetts Institute of Technology

Vivek Farias
Massachusetts Institute of Technology

Vahab Mirrokni
Google Inc.

Abstract

The present paper focuses on the problem of solving terabyte sized LPs on an hourly basis given a distributed computational infrastructure; solving these massive LPs is the computational primitive required for yield management problems arising in online advertising. Here we design a linear optimization algorithm borrowing from the multiplicative weights framework of Plotkin, Shmoys and Tardos that fits a paradigm for distributed computation referred to as "Map-Reduce". An implementation of our solver in a shared memory environment where we can benchmark against solvers such as CPLEX shows that the algorithm outperforms those solvers on the types of LPs that arise in the online advertising context.

1. Introduction

The "yield management" problem is a central optimization problem that must be solved by ad networks in the process of optimally matching "impressions" (sessions on web sites/ mobile apps, etc.) to advertisements. The problem is non-trivial since, in addition to many other business constraints, advertisers have finite budgets, the supply of impressions of a given type is limited, and finally, the economic value generated from a single impression can vary widely across ads and advertisers. The net economic value of this problem is large (on the order of tens of billions of dollars in a year).

A "first best" approach to solving this problem in practice entails solving a certain linear program – the so-called "DLP" – and today forms the basis to solving yield management problems in several large industries (airlines, hospitality, etc.). This approach has not seen wide adoption in advertising applications which instead rely on certain "adaptive greedy" approaches to allocation in spite of a potentially large up-side to using the former approach. One of the primary reasons motivating this choice is efficient computation at scale. In particular, we do not have effective tools for computing the DLP at web-scale and in the sorts of distributed computational environments that are germane to those settings.

This paper presents a candidate algorithm that is amenable to solving a large class of structured linear programs that include the DLP in Map-Reducible environments, and at web-scale. In particular, we make the following contributions:

1. We develop an algorithm for solving structured LPs (such as the DLP) by solving a sequence of "projected" versions of the DLP, a general scheme introduced in seminal work by Plotkins, Shmoys and Tardos two decades ago (Plotkin et al. [1995]). Here, we choose the projection carefully so that solving the projection relies on a fully combinatorial algorithm for the computation of a 2-D convex hull, which in turn relies on a sort. Put another way, the *key large scale computational step in our scheme is a large scale sort*; an operation that many modern distributed computational frameworks, such as Map-Reduce handle well (see for instance O'Malley [May 2008]).

2. We prove that the number of projections solved by our approach for an $\epsilon-$optimal solution scales like $O\left(\frac{\rho \log A}{\epsilon^2}\right)$, where A is the number of advertisers in the problem and ρ is a certain sparsity parameter. Importantly, the number of rounds (or sorts) is independent of the number of impression types which can be very large – in the case of a naive application of PST, the dependence would be $O(\log(I + A))$. Our algorithm is hence particularly appealing for "lop-sided" bipartite matching problems where $I \gg A$ (perhaps even exponentially so).

3. Most importantly, we implement our scheme in a large scale shared memory environment where comparisons with commercial, *non-distributed* solvers are possible alongside comparisons with state of the art distributed approaches tailored to solving packing programs. Here we show that we outperform distributed approaches for packing programs by up to an order of magnitude on typical instances. Surprisingly, we also outperform a state-of-the-art commercial solver that was optimized for network structured programs and made full use of all cores for linear algebraic operations; this commercial solver obviously cannot scale beyond a shared memory environment.

1.1. Literature Review

Our work lies at the intersection of several streams of research dealing with (a) dynamic resource allocation, (b) distributed algorithms for linear optimization .

Dynamic Resource Allocation: While the model of dynamic resource allocation we consider covers most models relevant to modern yield management, we single out two specific problems relevant to online advertising: *Display Ads Allocation (DA)* and *AdWords (AW)* both of which are concerned with the allocation of impressions to advertisers. Ciocan and Farias [2012] presents a model predictive control approach to solve such problems that requires repeatedly solving a certain LP and presents constant factor relative performance guarantees assuming that impression arrivals are governed by a general class of stochastic processes. A related class of stochastic models assumes that the stream of impressions form an exchangeable sequence. In this setting, Agrawal et al. [2013], Devanur and Hayes [2009], Feldman et al. [2010], Molinaro and Ravi [2012], Vee et al. [2010], all develop near optimal (i.e. $(1 - \epsilon)$-approximation) algorithms that rely on "learning" demand from a few early samples, solving a 'sampled' linear program and then deriving an allocation mechanism from the solution of this program. The class of linear programs we consider in this paper subsumes the class of programs considered in this stream of literature. It is worth noting that when the budget or capacity of advertisers is "large", the adversarial online AW and DA problems admit a $(1 - \frac{1}{e})$-competitive algorithm by applying a primal-dual technique Feldman et al. [2009], Mehta et al.. This primal-dual analysis again rests on analyzing the properties of a linear program of a similar nature as in the literature above and yields the sort of adaptive greedy algorithms that find use today. However, this approach eschews the use of impression traffic statistics taking a “worst-case” approach instead.

Distributable First Order Methods for Linear Programming. While designing a parallel algorithm for solving general linear programs is a P-complete problem, various primal-dual type techniques have been developed for packing and covering LPs. The multiplicative weights framework of Plotkin et al. [1995] is closest to the current work – our algorithm can be interpreted as a multiplicative update, customized to our special “resource allocation” LP structure. Other algorithms that share commonality with multiplicative weights have been studied in Garg and Könemann [2007], Luby and Nisan [1993]. Stateless distributed algorithms have been developed for these problems Awerbuch and Khandekar [2008] and have been recently generalized for solving mixed packing-covering LPs Manshadi et al. [2013]. There has been relatively less focus devoted to understudying these algorithms from a distributed systems perspective – to the best of our knowledge, Manshadi et al. [2013] is the only existing attempt at building a MapReducible LP algorithm. From a theoretical perspective, this work differs from the previous in that we seek a solution that is optimized for resource allocation LPs, which we define shortly, rather than a generic algorithm for packing/covering.

2. Linear Programs for Yield Management

The linear program we study is associated with a bi-partite graph with I sources indexed by i and J sinks indexed by j. The edge set of this graph has size E, and a generic edge is denoted by e. We use the notation $i(e)$ for the source node for edge e and $j(e)$ for its sink node. Given this graph, define the following primitives:

Demand: We associate each source i with a deterministic inflow equal to D_i.

Resources and Resource consumption: We are given a set of A distinct resources indexed by a. The available capacity of these resources is given by a vector $B \in \mathbb{R}_+^A$. We assume that allocating a unit of impression (demand) type i to advertiser (product) j (i.e. making an allocation along edge $e \triangleq (i,j)$) consumes $c_{a,e}$ units of resource a. We write $a \in j'$ if $c_{a,e} > 0$ for some edge e incident on j', (i.e. $j(e) = j'$).

Prices/ Revenues: Allocating a unit of demand along edge e generates revenue p_e. Denote by $p \in \mathbb{R}_+^E$ the column vector whose eth component is p_e.

The objective is to maximize total revenues from fractionally allocating demand from sources to sinks:

$$\begin{aligned} &\max_{x\geq 0} \sum_e p_e x_e \\ (1)\qquad &\text{s.t.} \ \sum_e c_{a,e} x_e \leq B_a \quad \forall\ a \\ &\sum_{e:i(e)=i} x_e \leq D_i \quad \forall\ i. \end{aligned}$$

We refer to the first set of constraints (indexed by a) as the *resource constraints*, and the second set (indexed by i) as the *source constraints*. We also note that, while our algorithm solves the primal allocation problem, it is quite easy (via complementary slackness) to transform an optimal primal solution to this LP into a dual solution and obtain bid prices.

In practice, D_i is obviously not known and must be learned; the following recipe has been proposed by a number of authors: estimate D_i by observing the demand process for a short time; use the LP with the estimated D_i. Re-estimate D_i at reasonable intervals of time and re-solve the LP.

2.1. Applicability

While the family of general resource allocation problems described above is applicable to many settings, we focus on modeling several large scale *offline* ad allocation problems of practical interest. In Section 4, we will show how the computational toolkit we build for the offline case yields powerful algorithms for the online setting as well.

Online Ad-Display. The source nodes correspond to arriving impression types,[1] and the sink nodes correspond to advertisers. There is a one-to-one correspondence between sinks and resources, as each advertiser has its own budget B_a which specifies the maximum number of times their ad can be displayed. Advertiser a declares a bid $b_{i,a}$ for each impression type i and there is an edge between i and a iff $b_{i,a} > 0$. Typically, the allocation mechanism is first price, so we set $p_e = b_{i(e),a(e)}$ and $c_{a,e} = 1$ if and only if $p_e > 0$. The model described here is typically representative of display advertising systems.

Generalized Second Price AdWords. This structure models sponsored search ad systems and differs from Ad-Display in several key features:

1. Mutiple slots: per impression, there are $k \geq 1$ slots to be assigned to advertisers. Each slot has a non-increasing quality factor θ_l, with $\theta_1 = 1$; this reflects the fact that the ad shown in the top-most slot is more likely to be clicked on than the same ad placed in a lower slot. Advertiser a declares a bid $b_{i,a}$ for the top slot for impression i, and its bids for the lower slots are scaled down by the quality factor.

2. Budget: in this version of the problem, budgets are specified in dollar terms rather than impression counts.

3. Generalized Second Price (GSP) allocation mechanism: upon the arrival of an impression of type i, the ad network selects a set of $k+1$ advertisers; the advertisers are sorted in decreasing order of their bids and assigned slots beginning from the top-most downwards. The amount the advertiser in the l-th slot is charged is equal to the bid of the advertiser in the $(l+1)$-th slot; the $(k+1)$-th advertiser does not receive a slot and is selected simply to determine the amount the k-th advertiser pays. There are two variants of GSP that have been studied in the literature: (a) the *strict* model, in which only advertisers with positive budgets can participate in the auction and (b) the *non-strict* model in which advertisers who have exhausted their budgets may still participate and set prices.

For non-strict GSP, the mapping to our resource allocation graph is as follows: while the source nodes still correspond to impressions, the sink nodes now correspond to $(k+1)$-tuples of advertisers which we call *slates*; the number of slates is $\binom{A}{k+1}$, where A is the number of advertisers. For an edge $e = (i,j)$ with $j = \{a_1, \ldots, a_{k+1}\}$ and, wlog, $b_{i,a} \geq b_{i,a'}$ for $a \geq a' \in j$, we set $p_e = \sum_{l=1}^{k} \theta_l b_{i(e),a_{l+1}}$

[1] An impression can be thought of as a unit of web-traffic satisfying certain pre-assigned criteria, such as say gender, age, website etc.

and $c_{a,e} = \theta_l b_{i(e),a_{l+1}}$ if and only if $a = a_l$. In Section 4, we discuss the usefulness of this formulation in the strict model as well.

Practical Scale For LP: In online ad related applications, the size of the bipartite graph is up to 1 billion impressions and 1 million advertisers, with 100 billion edges between them – in terms of data sizes, representing such a graph sparsely requires on the order of 100Gb. Lastly, we note that in both applications, it is quite natural to think of the number of an impression as scaling exponentially in some feature dimension d; for example, a given impression may be specified by hundreds of features describing the user's demographic information, browsing history and other parameters.

3. Map-Reducing The Yield Management LP

The yield management linear program (1) consists of A resource specific constraints and I impression type constraints. Our overall approach will solve a sequence of relaxations to this problem by 'averaging' the resource constraints. Each such relaxation will be solved using a (distributed) algorithm that exploits strengths of the Map-Reduce paradigm with the key step that uses 'all' the data being a single (large) sort. In greater detail, the following is a schematic view of the algorithm:

1. Begin with the uniform measure w^0 on all resources.
2. At the t-th stage construct a linear program, Relax(w^t) that relaxes (1) by replacing the A resource constraints
$$\sum_e c_{a,e} x_e \leq B_a \quad \forall\ a$$
with a single 'averaged' constraint
$$\sum_a w_a^t \sum_e c_{a,e} x_e \leq \sum_a w_a^t B_a$$
using the measure w^t.
3. Solve Relax(w^t) using a fully combinatorial distributed algorithm that exploits map-reduce; described in detail in Section 3.1.
4. Use the solution of the relaxation to compute w^{t+1} (discussed in Section 3.2); go to step 2.

3.1. Solving Relax(w)

Our goal here is to solve the relaxed program Relax(w) that forms the crux of each iteration of our scheme. We first write this relaxation as

$$\begin{aligned} &\max_{x \geq 0} \sum_e p_e x_e \\ &\text{s.t.}\ \sum_e c_e x_e \leq B \\ &\qquad \sum_{e:i(e)=i} x_e \leq 1 \quad \forall\ i, \end{aligned} \tag{2}$$

where $c_e \triangleq \sum_a w_a c_{a,e}$ and $B \triangleq \sum_a w_a B_a$. Let us begin by observing that we can partition all the decision variables x_e of (2) by source node (i.e. impression type) i, suggesting I impression types specific subproblems whose primal and dual respectively are:

$$\begin{aligned} f^i(B^i) = \max_{x \geq 0} \quad & \sum_{e:i(e)=i} p_e x_e \\ \text{s.t.} \quad & \sum_{e:i(e)=i} c_e x_e \leq B^i \\ & \sum_{e:i(e)=i} x_e \leq 1 \end{aligned}$$

$$\begin{aligned} \min_{u,v \geq 0} \quad & uB^i + v \\ \text{s.t.} \quad & uc_e + v \geq p_e \quad \forall e \text{ s.t. } i(e) = i \end{aligned}$$

Relax(w) is then equivalent to the following program:

$$\begin{aligned} \max_{B \geq 0} \quad & \sum_i f^i(B^i) \\ \text{s.t.} \quad & \sum_i B^i \leq \tilde{B}, \end{aligned} \tag{3}$$

This equivalent program is now easily seen to be a non-linear knapsack problem that we solve as follows:

1. Compute a representation of f^i independently for each impression type i: We can show that f^i is concave and admits the following representation:

   $$f^i(B) = \begin{cases} u^1 B + v^1, & \text{if } B \in [0, B_1^i) \\ \dots & \\ u^j B + v^j, & \text{if } B \in [B_{j-1}^i, B_j^i) \\ \dots & \\ u^l B + v^l, & \text{if } B \in [B_{l-1}^i, \infty). \end{cases}$$

   The number of pieces is $l \leq d+1$ where d is the number of non-zero coefficients in the constraint $\sum_{e:i(e)=i} c_e x_e \leq B^i$. Moreover, this representation can be computed in $O(d \log d)$ time. Denote the k-th 'segment' of this representation by the tuple (u^k, v^k, Δ_k) where $\Delta_k = B_k^i - B_{k-1}^i$ is the length of the segment with the convention that $B_0^i = 0$ and $B_l^i = \infty$. Define u^k as the slope of segment k.

2. Sort segments across all f^i by slope (the key step that cannot be solved independently across all impression types): Construct an ordered list L consisting only of segments with positive slope, with the property that segment slope is non-increasing in the order of the list. his step requires $E \log E$ operations.

3. Build a solution: We must now allocate the budget $\tilde{B}$ in (4) across impression types. We do this as follows: Consider segments in the list L in order. For segment (u^k, v^k, Δ_k) allocate the smaller of the remaining budget and Δ_k to that segment and decrement the remaining budget.

4. Construct primal solution: Given the optimal individual budget allocation to each subproblem, the primal solution to each subproblem can be computed using complementary slackness.

Before giving proofs for the validity of the algorithm's steps, we note that:

1. The *key computational step in the procedure above is the sort entailed in constructing the list L*; the remaining procedures are easily solved as (easy) independent sub problems. In particular, our overall algorithm will spend the bulk of its runtime in this step so that such an algorithm will benefit tremendously from a system that is optimized to sort very large sets of numbers.

2. The relaxation provides an upper bound on the true optimum and, consequently, a certificate for the optimality gap.

Lemma 1. *The optimal value of the i-th subproblem with respect to its assigned budget B^i is a piecewise linear function of the form*

$$f^i(B) = \begin{cases} u^1 B + v^1, & \text{if } B \in [0, B_1^i) \\ \dots \\ u^j B + v^j, & \text{if } B \in [B_{j-1}^i, B_j^i) \\ \dots \\ u^l B + v^l, & \text{if } B \in [B_{l-1}^i, \infty). \end{cases}$$

Additionally, $l \leq d+1$, where where $d = |\{e : i(e) = i\}|$ and the coefficients specifying f^i can be computed in $O(d \log d)$ time.

Proof. Consider the feasible region of the dual. Although there are $(d+2)^2$ possible intersections of the LP inequalities, each constraint cannot create two intersections points with other constraints that lie on the envelope of the feasible region. Hence, the feasible region has at most $d+1$ extreme points $(u^1, v^1), \dots, (u^l, v^l)$ with $l \leq d+1$.

It remains to prove that f^i indeed has the structure from the lemma's statement. Clearly, for the $(i+1)$-th extreme point to achieve at least the objective value than the i-th, it must be that:

$$B^i > \frac{v_{i+1} - v_i}{u_i - u_{i+1}}$$

and via a simple inductive argument it follows that extreme point (u^j, v^j) is optimal for B^i in the range

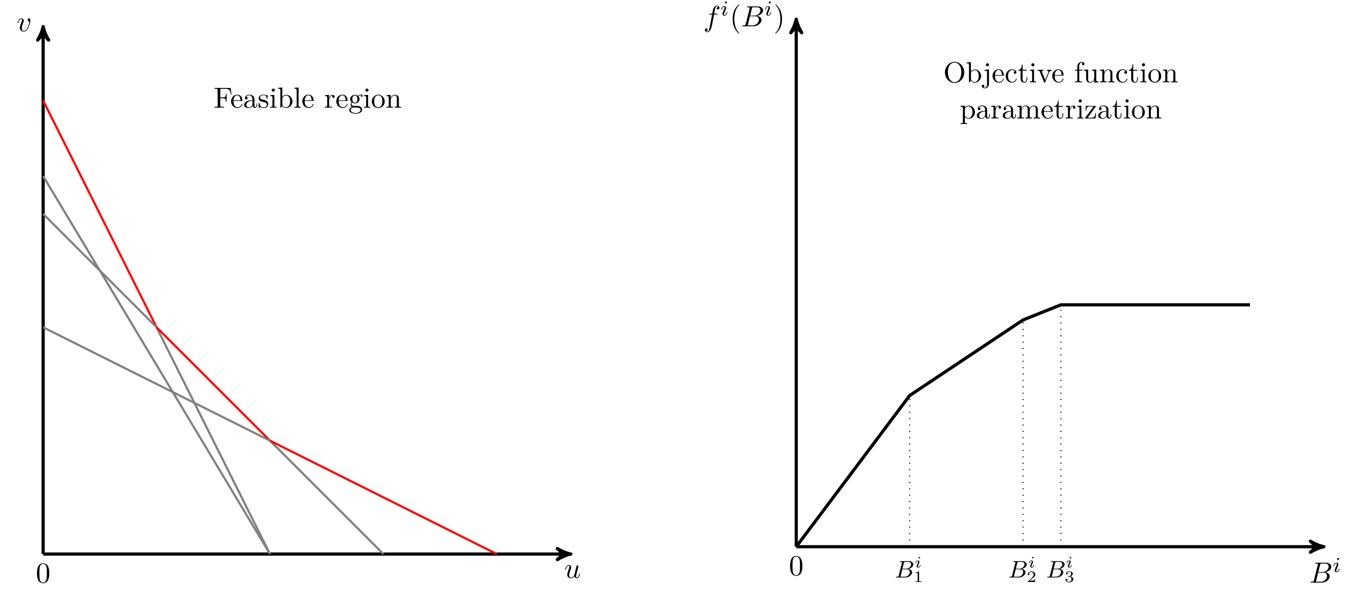
$$[B_{j-1}^i, B_j^i) = \left[\frac{v_i - v_{i-1}}{u_{i-1} - u_i}, \frac{v_{i+1} - v_i}{u_i - u_{i+1}}\right).$$

Moreover, it is a well known fact in computational geometry that finding the active constraints that define the envelope can be reduced to finding the convex hull of at most $d+1$ points in the plane (Har-Peled [2011]). Hence, the constraints that determine the upper envelope can be computed in $O(d \log d)$ time. ∎

The following lemma proves that it is easy to convert optimal dual solutions to optimal primal solutions. It relies on a standard complementary slackness-based argument.

Lemma 2. *Given a level of resource allocation B^i and an optimal dual solution (u, v), the optimal (primal) solution to the i-th subproblem can be found in $O(d)$ time.*

Figure 1: Feasible regions and objective value parametrization of the i-th subproblem.



Proof. Let us begin by writing down the complementary slackness conditions for the optimal u, v, x:

$$u\left(\sum_{e:i(e)=i} c_e x_e - B^i\right) = 0$$

$$v\left(\sum_{e:i(e)=i} x_e - 1\right) = 0$$

$$x_e\left(uc_e + v - p_e\right) = 0, \forall e \,\text{subject to}\, i(e) = i.$$

There are three cases:

1. $u = 0, v > 0$. Then, $\sum_{e:i(e)=i} x_e = 1$ and the optimal primal is found by allocating fully to the edge e with maximum price p_e. This follows from the feasibility condition $uc_e + v \geq p_e, \forall e \,\text{subject to}\, i(e) = i$, so only the x_e with maximal price can be positive. Note that we are assuming here there is a single item with maximal price - this can achieved without loss of generality by an arbitrarily small random perturbation of the price vector.

2. $v = 0, u \geq 0$. Similarly to the first case, it follows from the complementary slackness conditions that it is optimal to fully allocate along the edge e with the maximal ratio p_e/c_e.

3. $u > 0, v > 0$. Thus implies that $\sum_{e:i(e)=i} x_e = 1$ and $\sum_{e:i(e)=i} c_e x_e = B^i$. By construction of the envelope, we cannot have more than two tight constraints $v + uc_e - p_e = 0$, so by the last complementary slackness condition only two x_e's can be positive. Therefore, we can completely identify the primal solution.

In either case, the solution can be determined using no more than $O(d)$ calculations. ■

Having parametrized the subproblems, we now compute the optimal partitioning of $\tilde{B}$, i.e. find:

$$
\begin{aligned}
(B^1, \ldots, B^I) \in \operatorname{argmax} \quad & \sum_i f^i(B^i) \\
\text{subject to} \quad & \sum_i B^i \leq \tilde{B}, \qquad\qquad (4)\\
& B^i \geq 0 \quad \forall\, i,
\end{aligned}
$$

where f^i is the parametrization of the objective of the i-th subproblem (as in Lemma 1). Fortunately, it turns out that the optimal solution to this problem has a similar structure to the solution to a simple fractional knapsack program. In particular, the following algorithm computes the optimal budget partition to the subproblems by simply allocating budget in order of highest marginal return:

```
initialize $L = \emptyset$ and $B^i = 0, \forall i$; initialize unallocated budget $r = \tilde{B}$;
for $i = 1, \ldots, I$ do
  calculate parameters $(u^i_j, v^i_j, B^i_j)$ of $f^i$;
  add all slopes $u^i_j > 0$ of $f^i$ to the list $L$;
end for

sort $L$ in decreasing order;
while $r > 0$ do
  pick next $u^i_j \in L$ and increase $B^i$ by $\min\{r, B^i_j - B^i_{j-1}\}$;
end while
```

Theorem 1. *The resource partition $(B^1, \ldots, B^I)$ that minimizes (4) can be found in $O(E \log E)$ time by the above algorithm.*

Proof. We show that the procedure described above terminates with an optimal partition. In particular, the procedure initializes $(B^1, \ldots, B^n)$ to 0 and increases the resource allocation B^i to the subproblem i with the highest possible marginal return, given by u^i_j, to the point that either $\tilde{B}$ is exhausted or it not possible to increase budgets with positive marginal returns.

We prove the correctness of this procedure by contradiction. Assume that the optimal solution does not have the structure described above. Then, there must exist i_1, i_2, j_1, j_2 with $u^{i_1}_{j_1} > u^{i_2}_{j_2}$ such that $B^{i_1} < B^{i_1}_{j_1} - \nu$ and $B^{i_2} \geq B^{i_2}_{j_2-1} + \nu$ for some $\nu \geq 0$. Then, one can decrease B^{i_2} by ν and increase B^{i_1} by the same amount – the overall budget allocation remains the same, while the objective value of the global problem increases by $\nu(u^{i_1}_{j_1} - u^{i_2}_{j_2}) \geq 0$, obtaining the desired contradiction. ∎

3.2. Updating Weights

Given an initial set of weights w^t, and a solution to the program Relax(w^t) (obtained via the procedure above), we proceed to generate an updated set of weights w^{t+1} as follows. Now a solution to Relax(w^t) may well violate some (if not most) of the resource constraints in the LP we are trying to solve, (1). We first compute the extent by which each of these constraints is violated as defined by

$$v_a(x) = \sum_e c_{a,e} x_e - B_a.$$

Before we update weights we define an error parameter ϵ and a parameter ρ typically called the LP y*width* which equals $\max_a \max_x |v_a(x)|$. Now we update weights according to

$$w_a^{t+1} = w_a^t \left(1 - \epsilon \frac{v_a(x)}{\rho}\right).$$

Finally, we normalize w^{t+1} so that the entires sum to unity.

We are left with showing that the algorithm does not need to perform too many weight updates before converging to a good solution. In order to do this, we appeal to the multiplicative weights (MW) framework for solving packing/covering LPs Arora et al. [2005], Plotkin et al. [1995]. Note that the MW framework gives us algorithmic freedom in terms of what relaxation to choose for (1) – as opposed to our approach, a naive approach would instead relax all constraints of the LP, making the resulting relaxation a fractional knapsack. This, however, fails to take advantage of the special structure of our problem, and leads to inferior theoretical and practical performance.

The following theorem bounds the number of weight updates, and uses the machinery in Plotkin et al. [1995]:

Theorem 2. *Let $\delta > 0$ be an error parameter and $\epsilon = \min\left\{\frac{\delta}{4\rho}, \frac{1}{2}\right\}$. For $T \geq \frac{16\rho \log A}{\delta^2}$, consider the sequence $x^1, \ldots, x^T$ of solutions obtained thorough performing T multiplicative weight updates with update step size ϵ, and let $\overline{x}$ be the average solution, $\overline{x} = \sum_{t=1}^{T} x^t$. Then $\overline{x}$ satisfies:*

1. *$\overline{x} \in \chi$ and $\overline{x}$ is δ-feasible for all resource constraints, $\sum_e c_{a,e}\overline{x}_e \leq B_a + \delta, \forall\ a$.*
2. *The objective value of $\overline{x}$ at least equal to the optimal objective of (1).*

Proof. The first part follows directly from applying Theorem 3 from Arora et al. [2005]. The second part follows from the fact that the objective value of each relaxation is greater than the optimal objective of (1). ■

Before proceeding, we note that this approach is completely symmetrical: instead of relaxing all resource constraints while keeping the source constraints explicit, it is equally possible to relax the source constraints instead, and obtain an analogous relaxation with the same computational complexity of finding a solution. In fact, in our experiments described in Section 3.4, we test both approaches and find that choosing between these two relaxations carefully can have significant impact on performance.

3.3. Distributed Implementation of the Algorithm

While a comprehensive systems design document is beyond the scope of this paper, we outline how the inner algorithm is expressible in the MapReduce framework. One round is carried in two Map-Reduce steps. Note that, as in the implementation of TeraSort O'Malley [May 2008], we use a customized partitioner with randomized pivots for the aggregate list of slopes output by the mappers; this ensures that the list of slopes is globally sorted, and that the load on each reducer is balanced.

```
MapReduce round 1
  Mapper
    Input: subproblem $i$ data
    Calculate the set of slopes and budget cutoffs of $f^i$,
            $L_i = \{(i, j, u^i_j, \Delta^i_j),$ for all $j \in [|\{e : i(e) = i\}| + 1]\}$
        Emit (key, value) $= (u^j_i, (i, j, u^j_i, \Delta^i_j))$
    **Partitioner**
        Sample num_reducers keys from $\cup_i L_i$
        Assign $u^j_i$ to Reducer $k$ iff key$[k-1] \leq u^j_i <$ key$[k]$
    **Reducer**
        $B_k = 0$
        **for all** $u^j_i$ received **do**
            $B_k \leftarrow B_k + \Delta^i_j$, $L^k = L^k \cup (i, j, u^j_i, \Delta^i_j)$
        **end for**
        Emit $(k, B_k, L_k)$
**Synchronization round**
    Compute threshold $k^*$ at which $\sum_{1 \leq k \leq k^*} B_k \geq \tilde{B}$
    Output $(k^*, \tilde{B} - \sum_{1 \leq k \leq k^*-1} B_k)$
**MapReduce round** 2
    **Mapper**
        Input: $k^*, \tilde{B} - \sum_{1 \leq k \leq k^*-1} B_k, L_k$
        **if** $k \neq k^*$ **then**
            Allocate $B^i_j - B^i_{j-1}$ of $\tilde{B}$ along all slopes $u^i_j$ in $L_k$
        **else**
            Allocate $\tilde{B}$ in increasing slope order up to level $\tilde{B} - \sum_{1 \leq k \leq k^*-1} B_k$
        **end if**
        **for all** slopes in $L_k$ **do**
            Emit (key, value) $= (i, (u^j_i, i, j,$ resource_allocation$))$
        **end for**
    **Reducer**
        Aggregate optimal resource allocation $\tilde{B}_i$ for subproblem $i$
        Compute and output primal solution of subproblem $i$
```

3.4. Experimental Performance

We implemented a shared memory parallel version of our algorithm using C++ and OpenMP. The code can be found at: `https://sites.google.com/site/nips2014mrlp/`. We run our experiments on a machine with dual 2.93GHz Intel Xeon 6-core processors and 128Gb of memory. While our algorithm is ultimately designed to port to the *non*-shared memory model, the purpose of this shared memory implementation is to allow benchmarking versus: (i) state-of-the-art shared memory implementations of simplex: the benchmark we use is CPLEX 12.5 (using primal simplex) and (ii) first-order methods that are also amenable to distributed implementations: we pick the primal/ dual method ("AW P/ D") of Awerbuch and Khandekar [2008], Manshadi et al. [2013] as a benchmark and note that this method also employs multiplicative updates for the dual variables.

We test our implementation on a family of synthetic instances of Ad-Display and GSP AdWords problems. Each instance has 1mm impressions and 1mm advertisers. We generate the matrix of bids from advertisers to impressions in the following way: for each advertiser, we choose 100 impressions uniformly at random for which we assign an edge (i.e., a non-zero bid.) At this sparsity, storing the bid matrix (represented as a list) in memory requires 3Gb. We generate the bid values according to a factor model that is designed to simulate "hot"/"cold" advertisers and impressions. In particular, for each impression i, we generate a 5-dimensional feature vector ϕ_i with each component drawn i.i.d. from a log-normal distribution. We also generate a similar feature vector θ_a for each advertiser. We then set $b_{i,a} =< \phi_i, \theta_a >$ for each impression to advertiser edge.

We choose advertiser budgets in order to simulate several load factor scenarios. Depending on the instance type, we define its load factor in alternate ways:

$$\text{LF} = \begin{cases} \frac{\sum_a B_a}{I} & \text{if instance is Ad-Display} \\ \frac{\sum_a B_a}{I\mathbb{E}[b_{i,a}]} & \text{if instance is GSP AdWords} \end{cases}$$

We set the budgets B_a uniformly to achieve load factors taking values in $\{0.5, 0.75, 1, 1.5, 2\}$. We heuristically initialize the multiplicative weight of advertiser a to be proportional to the average bid going into a ($w_a \propto \sum_i b_{i,a}/100.$) For both sets of experiments, we set $\epsilon = 0.5$. Also, we set a burn-in period for which still do the regular multiplicative updates but which we do not count into the calculation of the primal solution; we use a burn-in of either 25 or 50 iterations and report whichever run results in fewer iterations. As noted before, our algorithm works analogously if we relax the source (impression) constraints rather than the resource (advertiser) ones; in the results below, we report performance of the best.

Table 1 reports the number of iterations required for our algorithm to reach 95% optimality and Figure 2 shows the speed of convergence for a particular Ad-Display instance where we set the load factor to 1. We note that the algorithm is remarkably fast at reaching 95% optimal solutions, requiring fewer than 100 iterations. Since in many distributed frameworks such as MapReduce, the setup time to run one round is quite high, an algorithm which requires a small number of rounds is highly desirable. The slow convergence beyond the 95% point can be improved by lowering ϵ at the expense of slower initial progress.

Table 2 compares the wall clock times required by our algorithm versus CPLEX and an implementation of AW P/D [2]; we measure the time required to reach 95% the optimal values for out set of Ad-Display and, respectively, AdWords experiments:

1. Comparison with CPLEX: Our algorithm is consistently as fast, if not better, than CPLEX. This is quite surprising, since CPLEX is optimized for speed in multicore shared memory environments.

2. Comparison with AW P/D: On Ad-Display instances, AW P/D has a running time of almost a factor of magnitude larger and, in fact, the best approximation we could achieve with AW P/D was only around 90% for some of our problems. We interpret this as an indication that the buy in our algorithm does not necessarily come from the use of a multiplicative update rule, but rather from the ability to tractably solve a particularly tight relaxation of the original feasible space. On our GSP AdWords instances, AW P/D is competitive with our scheme and CPLEX; this is consistent with our intuition that AdWords type instances should be easier to approximate due to the one to one correspondence between the objective value and the advertiser budget constraints.

[2]We also use the primal initialization and dynamic stepsizes employed in Manshadi et al. [2013]

Table 1: Iteration count to 95% optimality.

Instance Type \ LF	0.5	0.75	1	1.5	2
Ad-Display	52	48	59	55	73
GSP Ad-Words	34	46	56	59	62

Figure 2: Progress per iteration for our algorithm for an Ad-Display instance with LF= 1.

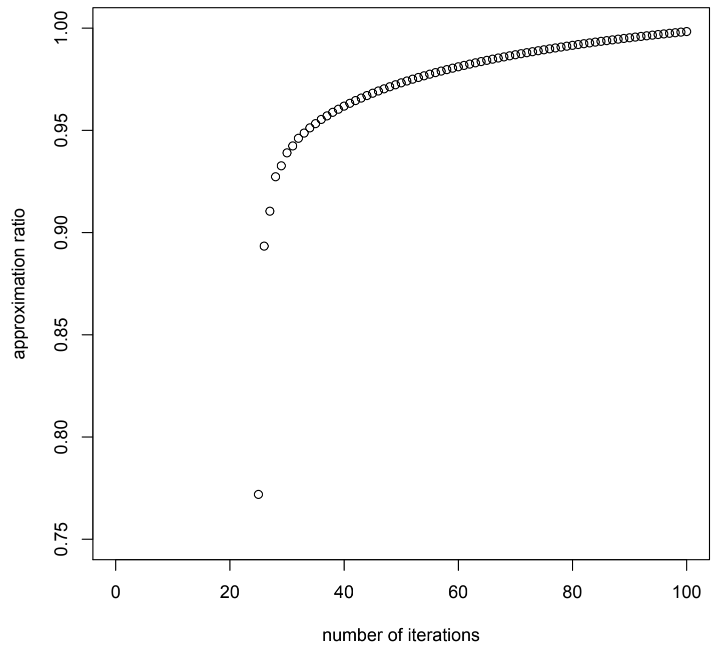


Table 2: 95% optimality wall clock times (in minutes) for Ad-Display.

	Ad Display				
Algo \ LF	0.5	0.75	1	1.5	2
MW	10.76	10.58	13.75	11.73	14.73
CPLEX	14.86	15	15.3	16.45	14.91
AW P/D Algo	86.66	110.2	132.2	>200	>200

Table 3: 95% optimality wall clock times (in minutes) for GSP AdWords.

	GSP AdWords				
LF / Algo	0.5	0.75	1	1.5	2
MW	7.03	9.51	13.06	11.38	11.72
CPLEX	15.23	15	15.3	15.71	14.91
AW P/D Algo	8.8	12.3	20.1	78.1	180.6

4. Optimal Allocation for Generalized Second Price AdWords

In this section, we extend the re-optimization scheme from Ciocan and Farias [2012] to online Generalized Second Price AdWords in the strict model. The main barrier to directly applying the result lies in the endogeneity of prices in the strict GSP setting: since any given slate j remains valid only if for every advertiser $a \in j$, $B_a > 0$, the rewards and resource consumption profiles are state dependent and nonlinear. Note that endogeneity is not an issue in the offline version of the problem for either the strict or non-strict models. Indeed, if impression frequencies are known ahead of time, the LP formulation described in Section 2 achieves uniform advertiser budget consumption over time, ensuring that every slate remains valid until the end of the time horizon. However, for the online version of the allocation problem, this task is much harder – as the the frequencies of impression arrivals vary over time, it is not clear the system can be controlled in a way that ensures smooth budget consumption.

We exhibit an algorithm with a guarantee in the strict model GSP. In the Appendix, we present a proof of the validity of the algorithm in two stages. First, we look at a non-strict model version of the system where the GSP price constraints have been relaxed (i.e., slates never become invalid); we show that re-solving a sequence of linear programs at discrete time intervals yields a constant factor of roughly 1/3 versus the offline optimal. Secondly, we make use of a crucial *budget balancing* property to show that following the prescriptions given by our relaxation do not lead to any revenue loss when we apply the GSP constraints ex post in the strict model system. In this abridged version of the paper, we present the main result for strict model GSP.

Impression arrival model. We associate each impression type i with a continuous sample path stochastic process $\{\Lambda_{i,t}\}_{t \in [0,T]}$. All I processes are defined on a common probability space $(\Omega, \mathcal{F}, P)$. We make the following assumption on Λ_t:

Assumption 1. ***Structure of*** $\{\Lambda_t\}$**:**

1. $\Lambda_{i,t} = \left(\overline{\Lambda}_{i,t}\right)^+$, *where* $\overline{\Lambda}_t$ *is a Gaussian process with continuous sample paths.*

2. $\mathsf{E}\left[\overline{\Lambda}_{i,t}\right] \triangleq \lambda_i$ *is positive.*

3. *The variance of the random variable* $\overline{\Lambda}_{i,t}$, $\sigma^2_{i,t}$, *is non-decreasing as a function of* t *and concave.*

GSP rewards and resource consumption. As before, let $b_{i,a}$ be the bid of advertiser a for impression i. Consider an edge e from an impression type i to a slate $j = \{a_1, \ldots, a_{k+1}\}$ and assume without loss of generality that the advertisers are indexed in order of decreasing bids ($b_{i,a} \geq b_{i,a'}$

for $a \geq a'$). Define the event $I_{e,t} = \{B_{a,t} > 0, \forall a \text{ s.t. } a \in j(e)\}$. At time $t \in [0, T]$, the rewards are:

$$p_e^{\mathrm{GSP}}(t) = \sum_{l=1}^{k} \theta_l b_{i(e),a_{l+1}} 1(I_{e,t})$$

Similarly, we define the budget consumption rates to be:

$$c_{a,e}^{\mathrm{GSP}}(t) = \begin{cases} \theta_l b_{i(e),a_{l+1}} 1(I_{e,t}) & \text{if } a = a_l, 0 \leq l \leq k \\ 0 & \text{otherwise.} \end{cases}$$

Control and Dynamics. Our control is applied continuously but only updated at N discrete time intervals $\{0, T/N, 2T/N, \ldots, T\}$. The control x that is calculated at time iT/N and remains in effect over $[iT/N, (i+1)T/N]$ is specified by a vector $x \in \mathcal{X}$, where $\mathcal{X} = \left\{x \in R_+^E : \sum_{e:i(e)=i} x_e \leq 1 \ \forall i\right\}$. An *admissible control policy* is a $\mathcal{X}$-valued process adapted to the filtration $\mathcal{F}_t$ which we will denote $\{x_t\}$. Denote the set of admissible control policies by Π_N. We also denote by $\{x_{e,t}\}$ the e-th component of $\{x_t\}$, i.e. the process describing the allocations across a particular edge e.

The state of the system at time t is given by the level B_t of advertiser budgets that remain at t. The evolution of B_t is specified by the differential equation:

$$\frac{d}{dt} B_{a,t} = -\sum_e c_{a,e}^{\mathrm{GSP}}(t) x_{e,d(t)}$$

for all a. Here $d(t) = \max_{\{i : iN/T \leq t\}} iN/T$.

Optimum value. The optimization problem is to find an admissible control policy $\{x_t\}$ that maximizes the overall revenues:

$$(5) \qquad \max_{\{x_t\} \in \Pi_N} \quad \mathsf{E}\left[\int_0^T \sum_e p_e^{\mathrm{GSP}}(t) \Lambda_{i(e),t} x_{e,d(t)} dt\right]$$

We denote the optimal value to this optimal control problem by $J_{\mathrm{GSP}}^{*,N}(B_0)$. We now define a re-optimization procedure which at discrete time intervals $t = iT/N$, observes the instantaneous demand rate Λ_t, assumes this rate remains unchanged for all $\tau \geq t$ and calculates the optimal control that would be optimal if this assumption were true, and if the rewards and resource consumption rates remained equal to their values at time 0. Specifically, for $(t, \Lambda, B) \in \mathbb{R}_+ \times \mathbb{R}_+^I \times \mathbb{R}^A$, define the linear program $\mathrm{LP}(t, \Lambda, B)$ according to:

$$\begin{aligned} \max \quad & \sum_e p_e^{\mathrm{GSP}}(0) x_e \Lambda_{i(e)} \cdot (T - t) \\ \text{subject to} \quad & \sum_e c_{a,e}^{\mathrm{GSP}}(0) x_e \Lambda_{i(e)} \cdot (T - t) \leq B_a^+ \quad \forall \ a, \\ & x \in \mathcal{X}. \end{aligned}$$

Let us define a control policy $\{x_t^{\mathrm{R}}\}$ defined so that

$$x_t^{\mathrm{R}} = x_{d(t)}^{\mathrm{R}},$$

where $x_{iT/N}^{\mathrm{R}}$ is any optimal solution to the linear program $\mathrm{LP}\left(iT/N, \Lambda_{iT/N}, B_{iT/N}\right)$. Moreover, we define the revenues garnered by this heuristic in the relaxed GSP system

$$J_{\mathrm{GSP}}^{\mathrm{R},N}(B_0) = \int_0^T \sum_e p_e^{\mathrm{GSP}}(t) \Lambda_{i,t} x_{e,t}^{\mathrm{R}} dt,$$

as well as an offline upper bound on $J^{*,N}_{\text{GSP}}(B_0)$

$$J^{\text{UB}}_{\text{GSP}}(B_0) = \text{LP}\left(0, \frac{1}{T}\int_0^T \Lambda_t dt, B_0\right).$$

We omit a formal proof that the offline is an upper bound on $J^{*,N}_{\text{GSP}}$; this can be found in Ciocan and Farias [2012].

The following theorem concerns the performance of the re-solving heuristic in this non-strict system:

Theorem 3 (Ciocan and Farias [2012]). *Consider demand processes $\{\Lambda_t\}$ satisfying Assumption 1. In addition assume that $\sigma_t/\lambda \leq \sqrt{2\pi\nu}$. We then have, assuming an initial inventory of x_0:*

$$\liminf_N \frac{\mathsf{E}\left[J^{\text{R}}_{GSP}(B_0)\right]}{\mathsf{E}\left[J^{\text{UB}}_{GSP}(B_0)\right]} \geq \max\left\{0.342, \frac{1}{1+\nu} - \frac{B}{1+\nu}\left(\exp(-1/4\pi\nu^2) + 0.853\right)\right\}$$

4.1. Guarantee in the Strict GSP Model

Let us now return to the strict GSP AdWords model; in the following, we make use of Lemma **??** to show that the control computed assuming a non-strict model will never exhaust advertiser budgets mid-way through the time horizon of the problem. Hence, across all sample paths of impression arrivals, all slates will remain active, implying equivalence between the strict and non-strict versions of the problem.

The stochastic control model defined for the non-strict GSP problem remains in place with the exception that now the rewards and resource consumption profiles become dependent on the remaining budget B_t in the system. Mathematically, we define now define:

$$p^{\text{GSP}}_{e,t} = \sum_{l=1}^{k} \theta_l b_{i(e),a_{l+1}} 1(I_{e,t})$$

Similarly, we define the budget consumption rates to be:

$$C^{\text{GSP}}_{a,e,t} = \begin{cases} \theta_l b_{i(e),a_{l+1}} 1(I_{e,t}) & \text{if } a = a_l, 0 \leq l \leq k \\ 0 & \text{otherwise.} \end{cases}$$

Effectively, the state of the system is no longer comprised of only B_t, but the tuple (B_t, p_t, c_t). The dynamics of B_t are specified by

$$\frac{d}{dt}B_{a,t} = -\sum_e C^{\text{GSP}}_{a,e,t} x_{e,d(t)}$$

and the optimal control problem becomes

$$\max_{\{x_t\}\in\Pi_N} \quad \mathsf{E}\left[\int_0^T \sum_e p^{\text{GSP}}_{e,t}\Lambda_{i(e),t} x_{e,d(t)} dt\right]. \tag{6}$$

Denote by $J^{*,N}_{\text{GSP}}(B_0)$ the optimal value of the above problem and note that $J^{\text{UB}}_{\text{non-strict-GSP}}(B_0)$ is an upper bound on the achievable optimum.

Lemma 3. *For any N,*

$$J^{*,N}_{GSP}(B_0) \le J^{UB}_{\textit{non-strict-GSP}}(B_0).$$

Proof. Let $\{x_t\}$ be an ϵ-optimal control for $J^{*,N}_{\text{GSP}}(B_0)$. Such a control must exist for any $\epsilon > 0$ (Bertsekas and Shreve [2007].) By construction, this control will achieve at least the same value in the relaxed system. Since the choice of ϵ was arbitrary, it follows that $J^{*,N}_{\text{GSP}}(B_0) \le J^{*,N}_{\text{non-strict-GSP}}(B_0)$. Since $J^{\text{UB}}_{\text{non-strict-GSP}}(B_0) \ge J^{*,N}_{\text{non-strict-GSP}}(B_0)$, the statement follows directly. ■

Lastly, define

$$J^{\text{R},N}_{\text{GSP}}(B_0) = \int_0^T \sum_e p^{\text{GSP}}_{e,t} \Lambda_{i,t} x^{\text{R}}_{e,t} \mathbf{1}(I_{e,t}) dt,$$

the revenues garnered in the real system using the sequence of re-optimized controls x^{R} for the relaxed GSP problem. In the following we will show that

$$\liminf_N J^{\text{R},N}_{\text{GSP}}(B_0) = \liminf_N J^{\text{R},N}_{\text{non-strict-GSP}}(B_0).$$

The following lemma states that due to smooth budget consumption, advertisers would be charged equal amounts in either strict or non-strict models:

Lemma 4. *Consider using the control $\{x^R_t\}$ for the GSP system. Then, almost surely for all t,*

$$\liminf_N p^{GSP}_{e,t} = p^{\textit{non-strict-GSP}}_e, \forall e$$
$$\liminf_N C^{GSP}_{a,e,t} = C^{\textit{non-strict-GSP}}_{a,e}, \forall a, e.$$

Proof. By Lemma **??**, it follows that $\liminf_N B_{a,t} \ge 0$ for all $t \in \{0, T/N, 2T/N, \dots T\}$ and consequently for all $0 \le t \le T$ since budget is depleted monotonically. Hence,

$$\begin{aligned}\liminf_N p^{\text{GSP}}_{e,t} &= \liminf_N p^{\text{non-strict-GSP}}_e 1(I_{e,t}) \\ &= p^{\text{non-strict-GSP}}_e \liminf_N 1(I_{e,t}) \\ &= p^{\text{non-strict-GSP}}_e.\end{aligned}$$

The proof for C^{GSP} is similar and omitted. ■

We are now ready to state and prove the validity of our algorithm in the strict model.

Theorem 4. *Consider demand processes $\{\Lambda_t\}$ satisfying Assumption 1. In addition assume that $\sigma_t/\lambda \le \sqrt{2\pi}\nu$. We then have, assuming an initial inventory of x_0:*

$$\liminf_N \frac{\mathsf{E}\left[J^{\text{R},N}_{GSP}(B_0)\right]}{\mathsf{E}\left[J^{\text{UB}}_{\textit{non-strict-GSP}}(B_0)\right]} \ge \max\left\{0.342, \frac{1}{1+\nu} - \frac{\nu}{1+\nu}\left(\exp(-1/4\pi\nu^2) + 0.853\right)\right\}$$

Proof. We have

$$\begin{aligned}\liminf_N \frac{\mathsf{E}\left[J^{\text{R},N}_{\text{GSP}}(B_0)\right]}{\mathsf{E}\left[J^{\text{UB}}_{\text{GSP}}(x_0)\right]} &= \liminf_N \frac{\mathsf{E}\left[J^{\text{R},N}_{\text{non-strict-GSP}}(B_0)\right]}{\mathsf{E}\left[J^{\text{UB}}_{\text{non-strict-GSP}}(x_0)\right]} \\ &\ge \max\left\{0.342, \frac{1}{1+B} - \frac{B}{1+B}\left(\exp(-1/4\pi B^2) + 0.853\right)\right\},\end{aligned}$$

where for the equality follows from Lemma 4, and the inequality follows from Theorem 3. ■

We end by noting that these results hold in a setting that is more general than strict model GSP. In fact, the proof generalizes for any mechanism where $I_{e,t}$ is a sufficient statistic for $p_{e,t}$ and $c_{a,e,t}$.

4.2. Reducing the Dimensionality of the LPs

A potential downside of our algorithm is the necessity to consider all possible $\binom{A}{k+1} = O(A^k)$ slates in the LP formulation. While it is unlikely an exact formulation of the problem could significantly reduce this number (for example, to $O(Ak)$), we make the simple observation that, due to the balancing property of our allocation policy, we can reduce the number of variables for the LP from $I\binom{A}{k+1}$ to $I\binom{A}{k}$. Hence, when $k = 1$, the dimensionality of the LP is equal to the first price case.

Consider the following linear program, together with its dual, and note that it has precisely the form of LP(t, Λ_t, B_t) except that, without loss of generality, we have normalized all $\Lambda_{i,t}$s to 1:

$$\begin{aligned}
\max \sum_{(i,j)} & p_{(i,j)} x_{(i,j)} \\
\text{s.t.} \sum_{j \ni a} & c_{a,(i,j)} z_{(i,j)} \le B_a \quad \forall\, a \\
\sum_{e:i(e)=i} & x_{(i,j)} \le 1 \quad \forall\, i \\
x & \ge 0.
\end{aligned}
\qquad\qquad
\begin{aligned}
\min \sum_i & \alpha_i + \sum_a B_a \beta_a z \\
\text{s.t.}\ & \alpha_i + \sum_{a \in j} c_{a,(i,j)} \beta a \ge p_{(i,j)} \quad \forall\, i, j \\
& \alpha, \beta \ge 0.
\end{aligned}
\tag{7}$$

The following theorem states that, instead of explicitly solving the above LP, we can solve a pruned version of it where we eliminate many superfluous decision variables.

Theorem 5. *For every impression i, consider the slate $j = (a_1, \ldots, a_k, a^{best}_{k+1})$ such that $b_{i,a_1} \ge \ldots \ge b_{i,a_k} \ge b^{best}_{i,a_{k+1}}$. Moreover, assume a^{best}_{k+1} is picked such that $\nexists a$ with $b_{i,a_k} \ge b_{i,a} > b^{best}_{i,a_{k+1}}$. Then there exists a solution to (7) such that for every slate $j' = (a_1, \ldots, a_k, a)$ with $b^{best}_{i,a_{k+1}} > b_{i,a}$, $x_{(i,j)} = 0$.*

Proof. Let us pick a slate $j' = (a_1, \ldots, a_k, \tilde{a}_{k+1}) \ne j$. Assume that $x_{(i,j')} = \Delta > 0$ in an optimal solution to the linear program 7. Consider building another solution $\tilde{x}$ such that

$$\begin{aligned}
\tilde{x}_{(i,j')} &= 0 \\
\tilde{x}_{(i,j)} &= x_{(i,j)} + \min\{\Delta, B_{a_k}/b_{i,a^{\text{best}}_{k+1}}\}.
\end{aligned}$$

Since $b^{\text{best}}_{i,a_{k+1}} > b_{i,\tilde{a}_{k+1}}$, $\sum_{e:i(e)=i} \tilde{x}_e \le 1$. Moreover, $\tilde{x}$ is built such the budget constraint of advertiser a_k remains feasible and the objective value of $\tilde{x}$ is at least as large as that of x. Hence $\tilde{x}$ is also optimal. We can apply this transformation repeatedly until we arrive at an optimal solution which satisfies the statement in the lemma. ■

The implication of this lemma is that, for every impression i, we only need to consider $\binom{A}{k}$ decision variables $x_{(i,j)}$ instead of $\binom{A}{k+1}$. This also implies that, in the case of single slot GSP AdWords ($k = 2$), the dimensionality of the LPs we need to solve is the same as for First Price AdWords.

5. Conclusions and Future Directions

The present paper is a first step towards building a distributed solver for allocation linear programs. The multiplicative weights algorithm we employ naturally takes advantage of the computational primitives that existing distributed computing frameworks are optimized to solve and has attractive theoretical guarantees. Quite surprisingly, our practical experiments indicate it also competitive or better in terms speed versus existing shared memory solvers. There are several future directions that we find interesting:

(i) *Alternating relaxations:* Our algorithm works symmetrically for the cases where we relax resource or source constraints. It would be interesting to see if an algorithm which alternates between solving a resource constraints relaxation and a source constraints one could yield better convergence.

(ii) *Warm start guarantees:* As suggested, the setting that is most interesting to us is one where the LP is resolved over the length of an ad campaign. Access to warm starts could potentially strengthen the convergence guarantees for our algorithm.

References

Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. A dynamic near-optimal algorithm for online linear programming, 2013.

Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta algorithm and applications. Technical report, 2005.

Baruch Awerbuch and Rohit Khandekar. Stateless distributed gradient descent for positive linear programs. In *STOC*, pages 691–700, 2008.

D. Bertsekas and S. Shreve. *Stochastic Optimal Control: The Discrete-Time Case*. Athena Scientific, 2007.

D.F. Ciocan and Vivek Farias. Model predictive control for dynamic resource allocation. mathematics of operations research. *Mathematics of Operations Research*, 37(3):501–525, 2012.

Nikhil R. Devanur and Thomas P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proceedings of the 10th ACM conference on Electronic commerce*, EC '09, pages 71–78, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-458-4. doi: 10.1145/1566374.1566384. URL `http://doi.acm.org/10.1145/1566374.1566384`.

Jon Feldman, Nitish Korula, Vahab Mirrokni, S. Muthukrishnan, and Martin Pal. Online ad assignment with free disposal. In Stefano Leonardi, editor, *Internet and Network Economics*, volume 5929 of *Lecture Notes in Computer Science*, pages 374–385. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-10840-2. doi: 10.1007/978-3-642-10841-9_34. URL `http://dx.doi.org/10.1007/978-3-642-10841-9_34`.

Jon Feldman, Monika Henzinger, Nitish Korula, Vahab S. Mirrokni, and Cliff Stein. Online stochastic packing applied to display ad allocation. In *Proceedings of the 18th annual European conference on Algorithms: Part I*, ESA'10, pages 182–194, Berlin, Heidelberg, 2010. Springer-Verlag.

ISBN 3-642-15774-2, 978-3-642-15774-5. URL `http://dl.acm.org/citation.cfm?id=1888935.1888957`.

Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, May 2007. ISSN 0097-5397. doi: 10.1137/S0097539704446232. URL `http://dx.doi.org/10.1137/S0097539704446232`.

Sariel Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, Boston, MA, USA, 2011. ISBN 0821849115, 9780821849118.

Michael Luby and Noam Nisan. A parallel approximation algorithm for positive linear programming. pages 448–457, 1993.

Faraz Makari Manshadi, Baruch Awerbuch, Rainer Gemula, Rohit Khandekar, Julián Mestre, and Mauro Sozio. A distributed algorithm for large-scale generalized matching. *PVLDB*, 6(9):613–624, 2013.

A. Mehta, A. Saberi, U Vazirani, and V. Vazirani. Adwords and generalized on-line matching. In *FOCS '05: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*.

Marco Molinaro and R. Ravi. Geometry of online packing linear programs. In Artur Czumaj, Kurt Mehlhorn, Andrew M. Pitts, and Roger Wattenhofer, editors, *ICALP (1)*, volume 7391 of *Lecture Notes in Computer Science*, pages 701–713. Springer, 2012. ISBN 978-3-642-31593-0. URL `http://dblp.uni-trier.de/db/conf/icalp/icalp2012-1.html#MolinaroR12`.

Owen O'Malley. Terabyte sort on apache hadoop, May 2008.

Serge A. Plotkin, David B. Shmoys, and Eva Tardos. Fast approximation algorithms for fractional packing and covering problems, 1995.

Erik Vee, Sergei Vassilvitskii, and Jayavel Shanmugasundaram. Optimal online assignment with forecasts. In *ACM EC*, 2010.