Balanced Allocation with Succinct Representation

Saeed Alaei[*] Ravi Kumar[†] Azarakhsh Malekian[*] Erik Vee[†]

[*]Dept. of Computer Science
University of Maryland
College Park, MD 20742.
{saeed, malekian}@cs.umd.edu

[†]Yahoo! Research
701 First Ave
Sunnyvale, CA 94089.
{ravikumar, erikvee}@yahoo-inc.com

ABSTRACT

Motivated by applications in guaranteed delivery in computational advertising, we consider the general problem of balanced allocation in a bipartite supply-demand setting. Our formulation captures the notion of deviation from being balanced by a convex penalty function. While this formulation admits a convex programming solution, we strive for more robust and scalable algorithms.

For the case of L_1 penalty functions we obtain a simple combinatorial algorithm based on min-cost flow in graphs and show how to precompute a *linear* amount of information such that the allocation along any edge can be approximated in *constant* time. We then extend our combinatorial solution to any convex function by solving a convex cost flow. These scalable methods may have applications in other contexts stipulating balanced allocation.

We study the performance of our algorithms on large real-world graphs and show that they are efficient, scalable, and robust in practice.

1. INTRODUCTION

In the *guaranteed delivery* setting, advertisers and users are mediated by the publisher (e.g., a search engine, an online newspaper). The advertiser buys a contract for a certain number of impressions (user visits to the publisher's page) and declares interest in a subset of user population called buckets. The goal of the publisher is to satisfy the demands by placing an ad from the advertiser on the web page visited by a user, if the user (i.e., the impression) belongs to the advertiser's bucket.

Motivation. Consider the following setting in the context of computational advertising and search engines. Each search engine user has three attributes (gender, age, location) and there are four advertisers who have bought contracts to target various user subsets; each advertiser specifies its bucket of interest by specifying appropriate user attributes. Advertiser A's bucket is young females (gender=female, age=young), B's bucket is all males (gender=male), C's bucket is senior Floridans (age=old, location=FL), and D's bucket is all Californians (location=CA). If the search engine only needs to satisfy the contract's requirement, assigning a sufficient number of users to an advertiser as long as they belong to the advertiser's bucket is an apparent feasible solution.

[*]Part of this work was done while the authors were visiting Yahoo! Research.

Unfortunately, such an assignment can be unfair and unrewarding to the advertiser for the following two main reasons.

(1) While each advertiser's bucket has some of the user attributes specified explicitly, the unspecified attributes are subject to interpretation. Most often, the advertiser is *equally* interested in *all* the users who belong to the bucket. For instance, it will be undesirable if B were a sports car dealer and the search engine assigns mostly middle-aged men to B. Likewise, it is undesirable if a disproportionate number of old women from Florida were assigned to C, who happens to be Florida real-estate agent. Thus, there is a tacit assumption by each advertiser that the users assigned to it are as "balanced and fair" as possible from the set of available users.

(2) There can be a large number of attributes and each at different levels of granularity (location=city, state, country, etc.) that it might never be *fully* possible for any advertiser to specify the desired buckets to the *finest* conceivable detail. For instance, A could be a toy store who failed to specify an explicit age group; likewise, D could be an earthquake insurance agent primarily targeting young homeowners near the fault line. In either case, it is more desirable for the search engine to assign a balanced set of users rather than blame the advertiser for under-specifying its bucket.

In this paper we address this problem. Given a set of impressions (i.e., the supply) and contracts (with demands and buckets), how to find a feasible assignment of impressions to contracts that is as *balanced* as possible? Answering this question involves formulating what being balanced precisely means in this context. And, given the large number of advertisers (typically, in the hundreds of thousands) and the astronomical number of impressions (typically, in the hundreds of millions) in an online setting, we insist on an algorithm that is *efficient*, in both time and space, that is scalable, and that yields insights into the structure of the allocation problem itself. In particular, we desire an allocation algorithm that is and whose allocation can be stored *succinctly*, ideally, using space linear in the number of impressions and contracts as opposed to the naive storage that is linear in the number of pair of impression and contracts (which can be quadratic). Of course, this succinct representation should let us reconstruct the allocation along every pair of contract and impression in a time-efficient manner. Thus, we trade space for time.

Our contributions. We consider the general problem of balanced allocation in a bipartite supply-demand setting. Our formulation, inspired by [14], is combinatorial and captures the notion of deviation from being balanced by a natural and general form of a penalty function. While this formulation admits a convex programming solution (assuming the penalty function is convex), it is undesirable in practice because of efficiency considerations and therefore we seek more robust and scalable solutions.

For the case of L_1 penalty functions we obtain a simple combinatorial algorithm for the balanced allocation problem. Our solution is based on solving a min-cost flow problem on bipartite graphs, which can be done very efficiently. By using a powerful dual formulation stemming from our combinatorial treatment of allocation and constraining the flow to be unique in a certain way, we also show how to precompute and store a *linear* amount of information such that the allocation along *any* edge in the bipartite graph can be approximately answered in *constant* time, under mild assumptions on the input instances. This space-efficient reconstruction method might be of independent interest in contexts beyond balanced allocation.

We also prove two additional properties of our formulation. First is robustness, where we show how to upper bound the performance loss when the supply estimates are only approximately known. Second is extensibility, where we show an even simpler greedy approximate algorithm when some of the demand constraints are allowed to be violated.

We perform an experimental evaluation of our algorithm on a large real-world dataset obtained from the Yahoo!'s display advertising system. Our experiments demonstrate the efficiency and the scalability of our min-cost based algorithm. In particular, our combinatorial solution (as opposed to a black-box linear/convex programming solution) makes it feasible to solve large instances effortlessly. Furthermore, our experiments also illustrate the space savings enabled by the reconstruction procedure.

Finally, we extend our combinatorial solution to any convex function. This involves solving a convex cost flow, which once again is more efficient than solve a general convex program.

Related work. The related work falls into three classes. The first is work on online allocation problems. The second is the ever-increasing body of literature in the area of computational advertising. The third is network flow problems and the role of primal-dual methods in computational advertising.

Vee, Vassilvitskii, and Shanmugasundaram first studied the online allocation with forecast problem, where given an approximation of the online supply, the goal is to create an efficiently reconstructible plan for performing some form of balanced allocation [15]. They focus on the efficiency and sampling aspect of the problem and consider only the *strictly* convex version, which makes it amenable to using fixed point criteria such as KKT conditions for non-linear optimization. Ghosh et al. [10] studied the problem of representative allocation for display advertising when there are both spot markets and guaranteed contracts; they propose a solution where guaranteed contracts are implemented by randomized bidding in spot markets. Devanur et al. [7] presented a sampling based method that computes a near-optimal allocation for online keyword matching with budget constrained advertisers and random permutation model. Their approach also computes a compact plan consisting of the dual variables of an LP which can later be used to efficiently compute the primal allocation online. None of these methods is combinatorial like ours and hence are more likely to be less scalable.

Online advertising is one of the most profitable resources for the large search engine companies and publishing sites such as `cnn.com`, `nytimes.com`. Two popular methods used for online advertising are slot ad auction and display advertisement. Most of the recent literature for online advertisement are focused on studying slot ad auction from the game-theoretic perspective [8]. There have been some recent work on display advertisement and guaranteed delivery. In [9], Feige et al. studied the guaranteed delivery for display advertisement with penalties. In the guaranteed delivery model, advertisers act as contractors. Each advertiser requests some number of impressions. If this request is accepted by the search engine, it would be called a contract. In this model, for each accepted contract, either the whole demand requested in the contract should be satisfied or the search engine will pay extra penalties for the non-satisfied portion of the demand. They showed that there is no constant approximation for their problem and present a bicriteria algorithm. Also they proved a structural approximation result for the adaptive greedy algorithm. The problem of advanced booking with costly cancellation also have been studied in [5] and [3] from a game-theoretic point of view.

Our solution is mainly based on the network flow problem and its dual. There is a large amount of literature on the network flow problem (e.g., [2]). The closest work to our method is the push-relabel algorithm of Goldberg and Tarjan [12]; they introduced a method for computing the maximum flow problem without using augmenting paths. The reconstruction of the min-cost flow instance is based on the dual variables of the min-cost flow solution. Primal-dual methods have been largely used as a tool to find approximation algorithms for various problems (e.g., [4, 1]). Recently, Devanur et al. [6] and Jain and Vazirani [13] used primal-dual methods and KKT conditions for solving market equilibria problems.

2. PRELIMINARIES

Suppose we are given a set I of *impressions* and a set C of *contracts*. Each impression $i \in I$ has a supply $s_i > 0$. Each contract $c \in C$ has (i) a *weight* $W^c > 0$ that captures its importance, (ii) a desired *bucket* $\text{imp}(c) \subseteq I$ of impressions, and (iii) a *demand* $d^c > 0$, denoting the number of impressions that need to be allocated to this contract. For an impression i, let $\text{con}(i) \subseteq C$ denote the set of contracts that desire i.

As stated earlier, the goal is to find the most balanced allocation of impressions to contracts. Let y_i^c be the number of impressions i that are assigned to contract c in a given allocation. Let

$$\delta_i^c = d^c \cdot \frac{s_i}{\sum_{i' \in \text{imp}(c)} s_{i'}},$$

be quantity that captures the ideal balanced allocation of impression i to contract c, i.e. a perfectly balanced number of impressions from impression i are assigned to contract c. Let $w^c = W^c/d^c$. The goal is to minimize

$$\sum_c w^c \sum_{i \in \mathrm{imp}(c)} \Delta(y_i^c, \delta_i^c),$$

where $\Delta(\cdot,\cdot)$ is the *penalty function* that penalizes deviation from the ideal balanced allocation, subject to the supply and demand constraints. Different norm/distance functions can be used for $\Delta(\cdot,\cdot)$; for example, $\Delta = L_1$, $\Delta = L_2$, $\Delta = \mathrm{KL}$, and so on [14]. If $\Delta(\cdot,\cdot)$ is not restricted to be convex, then the problem becomes NP-hard to even approximate to within a constant factor (proof omitted).

We define the notion of ϵ-robust input.

DEFINITION 1 (ϵ-ROBUST INPUT). *An input instance to our problem is* ϵ-robust *if there is a feasible assignment of impressions to contracts when we scale up all the demands by a factor of* $1 + \epsilon$.

Henceforth, we will assume that our input instances are ϵ-robust for a suitable ϵ; this is a mild technical assumption that typically holds in practice. Also, a superscript c will always denote a contract and a subscript i will always denote an impression.

3. THE L_1 PENALTY FUNCTION

The balanced allocation problem with the L_1 penalty function can be formulated as a linear programming (LP) problem. Our main result is that we can solve this LP by instead solving a min-cost flow problem, i.e., there is a combinatorial solution to the balanced allocation problem with L_1 penalty. In Section 4, we show that by preprocessing the network flow solution, we can find a succinct representation that only stores $O(|C| + |I|)$ values and can reconstruct the asymptotically optimal solution in $O(1)$ time. Later in Section 5, we obtain an approximate solution (along with efficient reconstruction) when the demand constraints are "soft."

We first consider an LP formulation of the problem:

$$\min \sum_{c \in C} w^c \sum_{i \in I} \left| y_i^c - \delta_i^c \right|, \qquad (1)$$

subject to

$$\begin{aligned} \forall c \in C,\ \sum_{i \in \mathrm{imp}(c)} y_i^c &= d^c \qquad \text{(demand)} \\ \forall i \in I,\ \sum_{c \in \mathrm{con}(i)} y_i^c &\le s_i \qquad \text{(supply)} \end{aligned}$$

To simplify the description, for a given allocation A, we define $\mathsf{unfair}(y) = \sum_c w^c \sum_i |y_i^c - \delta_i^c|$. The flow network, with capacity $\mathrm{cap}(i,c)$ and cost $\mathrm{cost}(i,c)$ on each edge (i,c), is constructed as a four layer graph G (Figure 1).

Note that minimizing absolute value can be converted to two linear inequalities without loss of generality. The first and the last layers are the source s and sink t respectively. The second layer represents the set C of contracts, and the third layer stands for the set I of impressions. Source s has an edge (s,c) to each contract $c \in C$ in the second layer, with $\mathrm{cap}(s,c) = d^c$ and $\mathrm{cost}(s,c) = 0$. Contract $c \in C$ in the second layer is connected to the impression $i \in I$ in the third layer iff $i \in \mathrm{imp}(c)$. In this case, there are two edges — a top edge and a bottom edge — between c and i. The

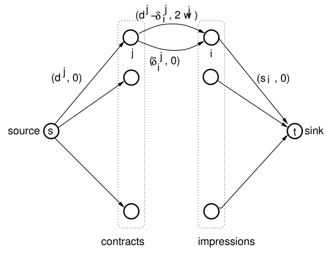

Figure 1: The network construction with (capacity, cost) on the edges for L_1.

top edge has capacity $d^c - \delta_i^c$ and cost $2w^c$ and the bottom edge has capacity δ_i^c and cost zero. Finally each impression $i \in I$ in the third layer is connected to the sink t by an edge (i,t) with $\mathrm{cap}(i,t) = s_i$ and $\mathrm{cost}(i,t) = 0$. Let E denote the set of edges between the second and the third layers in Figure 1.

THEOREM 2. *The min-cost s-t flow on G is the solution to (1).*

PROOF. We need to show the following: (i) a maximum s-t flow in G is a feasible solution to (1), provided (1) has a feasible solution; (ii) any feasible solution to (1) is a feasible s-t flow in G; and (iii) minimizing the cost of the maximum s-t flow in G is equivalent to minimizing the objective in (1). It is easy to see that if (1) has a feasible solution, then a maximum s-t flow in G is a feasible solution. Since we look for the maximum flow, if there is any feasible solution to the LP that can satisfy all the demands and the supply constraint, then that solution would be selected as the maximum flow as well. This means if (1) has a feasible solution, then any feasible maximum flow in our network is a feasible solution to (1). Second, it is easy to see that any feasible solution to (1) is also a feasible s-t flow in G. And last, we argue that the cost of the optimal solution to (1) is the same as the minimum cost of the maximum flow in G. In other words, we need to show that the costs of the flow are computed correctly in G. Note that the total contribution to the cost of the pairs of impressions and contracts that are over-assigned, i.e., $y_i^c > \delta_i^c$, is half of the total cost. So it is enough to compute twice this cost. Now consider an impression i and a contract c. If $y_i^c > \delta_i^c$, then we can route at most δ_i^c flow through the edge with cost 0 and route $y_i^c - \delta_i^c$ through the edge with cost $2w^c$, which is equal to $2w^c(y_i^c - \delta_i^c)$, which is exactly twice the cost of this pair. $\square$

So we showed that our min-cost flow solution has exactly the same value as (1). Next, we show that in fact, we can preprocess the min-cost flow solution so that by keeping track of just $O(|C| + |I|)$ values, we can reconstruct the complete flow.

4. RECONSTRUCTION FOR L_1

Even though the min-cost flow formulation obtains the optimal solution, as we mentioned earlier, it is not feasible to store the entire allocation information. Naively storing the solution representation is expensive since it uses $O(|E|)$ space. (As we will see in Section 6, supply–demand bipartite graphs that occur in practice tend to have high average degree, i.e., $|E|$ is much larger compared to $|C| + |I|$.) Ideally, we wish to store just $O(|C| + |I|)$ information (i.e., amortized $O(1)$ per node) that will let us reconstruct the flow along every edge; for practical reasons, we also require such a reconstruction to be time-efficient.

We consider the L_1 formulation and show an approximate reconstruction via dual variables for the nodes. First, we write the LP corresponding to the min-cost flow; from Theorem 2, this LP is equivalent to (1).

$$\min \sum_{c} \sum_{i \in \mathrm{imp}(c)} 2w^c x_i^c \tag{2}$$

subject to

$$\begin{aligned} \forall c \in C, \quad & \sum_{i \in \mathrm{imp}(c)} (x_i^c + \overline{x}_i^c) = d^c \\ \forall (i,c) \in E, \quad & \overline{x}_i^c \le \delta_i^c \\ \forall i \in I, \quad & \sum_{c \in \mathrm{con}(i)} (x_i^c + \overline{x}_i^c) \le s_i \\ \forall (i,c) \in E, \quad & \overline{x}_i^c, x_i^c \ge 0 \end{aligned}$$

Here, x_i^c denotes the flow along the top edge and $\overline{x}_i^c$ denotes the flow along the bottom edge from c to i. The dual of the above LP looks as follows.

$$\max \sum_{c \in C} Y^c d^c - \sum_{i \in I} Z_i s_i - \sum_{(i,c) \in E} A_i^c \delta_i^c \tag{3}$$

subject to

$$\begin{aligned} \forall (i,c) \in E, \quad & Y^c - Z_i - A_i^c \le 0 \\ \forall (i,c) \in E, \quad & Y^c - Z_i \le 2w^c \\ \forall (i,c) \in E, \quad & A_i^c, x_i \ge 0 \end{aligned}$$

Here, A denotes the allocation, where A_i^c is the allocation amount from impression i to contract c. Since we want to maximize the dual function and the coefficient of each A_i^c in the objective function is negative, we would like to set the A_i^c's as small as possible. From the constraints in the dual, this means $A_i^c = \max(0, Y^c - Z_i)$ and hence we do not need to keep track of A_i^c's. Next, we show how to reconstruct y_i^c's by only keeping Y^c's and Z_i's and then we show that in fact it is enough just to keep track of the Y^c's.

Since we are considering an optimal solution, because of the complementary slackness, we have three cases for each edge between i and c. First we consider the bottom edges.

(i) $Y^c - Z_i < 0$: in this case, we have $\overline{x}_i^c = 0$.

(ii) $Y^c - Z_i > 0$: in this case, $\overline{x}_i^c$ is fully saturated.

(iii) $Y^c - Z_i = 0$: in this case, we have $Y^c = Z_i$. This is the only case that we have an edge between (i, c), with the only constraint being the edge capacity constraint.

We have the same scenario for the top edges as well. In the third case, the dual variables do not give us any information on the value of the primal. We call the edges belonging to the third class *slack edges*. Next, we argue that any feasible assignment of flow to slack edges would be a solution. To see this, first notice that the cost of any slack edge (i, c) is exactly to $Y^c - Z_i$. Further, any path from i to c consisting of only slack edges have the same cost. Also, any cycle consisting of slack edges has a cost of 0. Therefore, any feasible maximum flow on the slack edges would constitute a solution. However, this means we have to be able to reconstruct a maximum flow on the slack edges. Thus, in the worst case, reconstructing an arbitrary maximum flow using the dual is no easier than finding a maximum flow from scratch! Therefore, we need to store extra information to be able to reconstruct a maximum flow on the slack edges efficiently.

Next, we provide a combinatorial solution in which we show how to compute a representation for approximate reconstruction (Section 4.1) and how to use this representation to do the actual reconstruction (Section 4.2). In Section 4.3 we discuss the generic effect of supply and/or demand scaling, which is of interest when supply forecasts, for instance, are not available precisely.

Remark. At a first glance, a extreme point solution to the primal LP of the maximum flow might appear to require only linear storage; this is not the case, however. The primal LP of a maximum flow for a graph with n nodes and m edges has m variables and $m + n$ constraints. If an extreme point solution to this LP has k non-zero variables, there should be k linearly independent tight constraints. Among those constraints, at most n of them could be node constraints and the remaining $k - n$ should be edge capacity constraints. However, a tight edge capacity constraint uniquely identifies the flow on it respective edge so we do not need to store the flow value for these edges. So we need to store the flow explicitly for at most n edges. This argument however requires that we *already* know which edges have non-zero flow. The number of edges with non-zero flow could be of $O(m)$ which would require more than linear space to store.

To illustrate this with an example, consider a maximum flow on a graph with three nodes s, v, and t. Suppose there is one edge of capacity k from s to v and $k + 1$ edges from v to t, each of capacity 1. Among the $k + 1$ edges from v to t, at most k edges can be tight and we need to store which k of them are non-zero. Note that there is no way to identify those k edges from the dual LP.

4.1 Computing the representation

As described earlier, by computing the dual variables of the min-cost flow, we can decide which edges are saturated and which are empty. For the remaining (i.e., slack) edges, the problem is reduced to computing a maximum flow in the new subgraph. Here, we present a way to find a *specific* maximum flow solution that is *easy* to reconstruct. We start by developing some definitions.

For a given node $v \in V$ and a given flow function $\mathsf{flow} : E \to R^+$ on the edges, let

$$\mathsf{in}(v) = \sum_{u:(u,v) \in E} \mathsf{flow}(u, v), \quad \mathsf{out}(v) = \sum_{u:(v,u) \in E} \mathsf{flow}(v, u),$$

$$\mathsf{excess}(v) = \mathsf{in}(v) - \mathsf{out}(v).$$

Definition 3 (ϵ-feasible flow). *A flow function* $\mathsf{flow}(x, y) : E \to R^+$ *is called* ϵ-feasible *if and only if for any contract* c, $0 \le \mathsf{excess}(c) \le \epsilon \cdot \mathsf{out}(c)$ *or equivalently* $\mathsf{out}(c) \le \mathsf{in}(c) \le (1 + \epsilon)\,\mathsf{out}(c)$.

Suppose we have a function $\mathsf{height} : V \to R^+$ such that the flow on each edge $e = (i, c)$ is given by

$$\mathsf{flow}(i, c) = \min(1, \max(\mathsf{height}(i) - \mathsf{height}(c), 0)) \cdot \mathrm{cap}(i, c). \tag{4}$$

Before we show the existence of such a function, we define another function $\mathsf{xheight}(v)$ as follows. For a given $\mathsf{height}(\cdot)$, if by changing $\mathsf{height}(v)$ to $\mathsf{height}(v) + \delta$ we get $\mathsf{excess}(v) = 0$, then we define $\mathsf{xheight}(v) = \delta$. Intuitively, $\mathsf{xheight}(v)$ indicates how much we need to increase the height of v (while keeping other nodes fixed) to reduce the excess flow of v down to 0. It is easy to see that $\mathsf{xheight}(v)$ can be computed using a binary search. (To see this, first assume $\mathsf{height}(v')$ is fixed for all $v' \neq v$ and wlog $\mathsf{height}(v_1) \leq \cdots \leq \mathsf{height}(v_{|I|+|C|-1})$. If we assume that when $\mathsf{excess}(v) = 0$, $\mathsf{height}(v_i) \leq \mathsf{height}(v) \leq \mathsf{height}(v_{i+1})$, then computing $\mathsf{xheight}(v)$ will be reduced to solving a linear equation. Now to find i, we can perform a binary search over all the nodes at the given heights.)

Next we show that there exists a function $\mathsf{height}(v)$ for which the corresponding flow is ϵ-feasible and it is greater or equal to maximum feasible flow on the slack edges. Note that for a given function $\mathsf{height}(v)$, computing the corresponding flow is trivial just from its definition using (4). The method we use for computing the height function is as follows. Let $\mu = \frac{\epsilon}{2|I|(1+\epsilon)}$.

Algorithm 1 Computing $\mathsf{height}(\cdot)$.

```
1: Initialize $\mathsf{height}(s) = 2|C|+2$ and $\mathsf{height}(v) = 0$ for $v \neq s$
2: repeat
3:     For the current function height, find the node $v$ with largest $\mathsf{xheight}(v)$
4:     Set $\mathsf{height}(v) = \mathsf{height}(v) + \mathsf{xheight}(v)$ and update the excess flow values and $\mathsf{xheight}(\cdot)$ for the rest of the nodes
5: until $\mathsf{xheight}(v) \leq \mu$ for all $v \in V$
```

Note that the above algorithm might appear similar to the push-relabel algorithm of Goldberg and Tarjan [12]. However, the requirement that the amount of flow going through an edge is a function of the height difference of its endpoints and the non-integrality of the heights makes our setting different from theirs. We now show that the corresponding flow for $\mathsf{height}(\cdot)$ after the termination of the algorithm is ϵ-feasible and is at least as large as the maximum feasible flow. First we show some properties of the algorithm.

LEMMA 4. *After the termination of the algorithm, the set of edges between s and contract nodes are all fully saturated or in other words for any contract node c, $\mathsf{height}(c) \leq \mathsf{height}(s) - 1$.*

PROOF. First we partition the nodes into two sets X and Y based on whether they are reachable from s on the residual flow graph. We first claim that $Y \neq \emptyset$ and $t \in Y$. This follows since there can be no simple path from s to t in the residual graph. Any simple path from s to t is of length at most $2|C| + 1$ and since $\mathsf{height}(s) = 2|C| + 2$, it should be the case that for some edge (u, v) on the path, $\mathsf{height}(u) > \mathsf{height}(v) + 1$ and therefore $\mathsf{flow}(u, v) = \mathrm{cap}(u, v)$, which means (u, v) is not in the residual graph. Thus, s and t are disconnected.

Since X and Y are non-empty and disjoint, (X, Y) is a cut. Notice that the size of this cut is less than or equal to the cut which consists of edges between s and the contract nodes, because we may have some excess flow on nodes in X. We can conclude that if the cut (X, Y) is not the same as the cut between s and the contract nodes then the demands of the contracts were not satisfiable to begin with which contradicts our assumption. Therefore, $X = \{s\}$ and $Y = \bar{X}$, so for any contract node c, $\mathsf{height}(c) \leq \mathsf{height}(s) - 1$. □

LEMMA 5. *For all $v \in V$, $\mathsf{excess}(v)$ is always non-negative during running the algorithm.*

PROOF. First, we can see that at the initialization step, $\mathsf{excess}(v) \geq 0$. We show that after running each step, the property still holds. Suppose the algorithm selected v at a round and relabeled it so after the relabel step $\mathsf{excess}(v) = 0$. For all nodes that has an outgoing edge to v, their outgoing flow is only decreased so their excess will increase. And for all nodes with incoming edges from v, their incoming flows only increase so their excess will increase as well. For the rest of the nodes the excess will not change. □

THEOREM 6. *The flow computed by Algorithm 1 is ϵ-feasible.*

PROOF. Consider the time when the algorithm has terminated. We know that for each $v \in V$, if we increase its heights by $\mathsf{xheight}(v) < \mu$ then $\mathsf{excess}(v)$ becomes 0. Consider a node c from the contract layer at the end of the algorithm. Since we know that all the contracts are satisfiable, the set of edges from s to the contract nodes is the minimum cut and $\mathsf{height}(c) \leq \mathsf{height}(s) - 2$. (Note that $\mathsf{height}(s) = 2|C| + 2$ and $\mathsf{height}(t) = 0$ and any simple path between each contract and t has a length less than or equal to $2|C|$.) Therefore, $\mathsf{flow}(s, c) = \mathrm{cap}(s, c)$ and raising c by μ will not change its incoming flow. Suppose $\mathsf{out}^{\max}(c)$ is the total capacity of edges going out of c. We know that after the algorithm terminates, $\mathsf{xheight}(c) \leq \mu$. Hence, $\mathsf{in}(c) - \mathsf{out}(c) = \mathsf{excess}(c) \leq \mu\, \mathsf{out}^{\max}(c)$. Also note that there can be at most $2|I|$ edges going out of c, each one with a capacity of less than the incoming capacity of c which was $\mathrm{cap}(s, c)$. This implies $\mathsf{out}^{\max}(c) \leq 2|I|\mathrm{cap}(s, c)$ and therefore $\mathsf{in}(c) - \mathsf{out}(c) \leq 2\mu|I|\mathrm{cap}(s, c)$. On the other hand since $\mathsf{in}(c) = \mathsf{flow}(s, c) = \mathrm{cap}(s, c)$, we can conclude that $\mathsf{in}(c) - \mathsf{out}(c) \leq 2\mu|I|\, \mathsf{in}(c)$. By rearranging the terms, we get $\mathsf{in}(c) \leq (1 + \epsilon)\, \mathsf{out}(c)$. □

Next, we bound the running time of the algorithm that computes the representation.

LEMMA 7. *The algorithm terminates after at most $O((|C| + |I|)|I| \cdot |C|/\epsilon)$ iterations.*

PROOF. First of all, at each iteration, we increase the $\mathsf{height}(v)$ for some node v by at least μ. Notice that no node will ever go higher than the source s so each node can be relabeled at most $(2|C| + 2)/\mu$ times so the total number of iterations (for all nodes) in the worst case is $O((|C| + |I|)|C|/\mu)$ which is $O((|C| + |I|)|I| \cdot |C|/\epsilon)$. □

Note that the algorithm 1 is used in the preprocessing step which is offline. Furthermore, even at its worst-case complexity which is $O((|C| + |I|)|I| \cdot |C|/\epsilon)$, it is still faster than mincost-flow or LP.

4.2 Reconstruction using the representation

We now describe how to reconstruct the flow using just $\mathsf{height}(v)$. For now we assume that after the termination of the algorithm there is no excess flow on any node, i.e., $\forall v, \mathsf{excess}(v) = 0$. Obviously, because of the way we constructed the height function, the flow of every edge (u, v) is $\mathsf{flow}(u, v) = \min(1, \max(\mathsf{height}(u) - \mathsf{height}(v), 0)) \cdot \mathrm{cap}(u, v)$. This would work perfectly well if there was no excess flow on any node. However because of the excess flows, the solution may not be feasible. To fix that we first tweak the demands before computing the height function and show that if the input instance is $(2\epsilon + \epsilon^2)$-robust, then we can reconstruct a feasible solution. Consider the following modification of our method.

(i) Scale up all the demands by a factor of $(1+\epsilon)^2$, compute the height function $\mathsf{height}(\cdot)$ as explained in Algorithm 1, and then set the demands back to their original values.

(ii) At reconstruction time, for each contract c and impression i, reconstruct the flow on (c, i) using $\mathsf{height}(c)$ as before but then scale it down by a factor of $(1 + \epsilon)$.

THEOREM 8. *Suppose that the given input instance is $(2\epsilon+\epsilon^2)$-robust. Then, the reconstructed solution according to the above modification is feasible. Furthermore, it may assign impressions to contracts up to $(1 + \epsilon)$ times their demand.*

PROOF. First, notice that since the input is $(2\epsilon + \epsilon^2)$-robust, we can still satisfy all the demands which means the set of all demand edges is still a minimum cut. After Algorithm 1 terminates, since the solution is ϵ-feasible, for any contract c we have $\mathsf{out}(c) \geq \mathsf{in}(c)/(1 + \epsilon)$. But notice that we scaled up all the demands by $(1 + \epsilon)^2$ at the beginning out height computation algorithm, so $\mathsf{in}(c) = (1+\epsilon)^2 d^c$ where d^c is the original demand of the contract c. Therefore, $\mathsf{out}(c) \geq (1 + \epsilon)d^c$ and clearly if we scale down the flow that we reconstruct for the edges going out of c by $(1+\epsilon)$, still the outgoing flow of c is at least as much as d^c which means the demand constraints are satisfied. Using a similar argument for the supply side we can show that for any supply node i the incoming flow of i cannot exceed its supply s_i. $\square$

A summary of the whole process is given in Algorithm 2 and Algorithm 3.

Algorithm 2 Preprocessing.

```
1: Build the min-cost flow graph $G$, run the min-cost flow
   and compute the dual variables $\{Z_i\}_{i\in I}$ and $\{Y^c\}_{c\in C}$
2: Remove the edges that are forced to be saturated or
   empty, update the supplies and demands; let $G' =
   (I', C', E')$ be the new graph
3: Scale all the demands by $(1 + \epsilon)^2$
4: $\forall v \in I' \cup C'$, compute the $\mathsf{height}(v)$ using Algorithm 1
```

Algorithm 3 Reconstruction (c, i).

```
1: $\{(c, i)$ is the given edge$\}$
2: if $Y^c - Z_i < \mathrm{cost}(i, c)$, then let $\mathsf{flow}(c, i) = 0$
3: if $Y^c - Z_i > \mathrm{cost}(i, c)$, then let $\mathsf{flow}(c, i) = \mathrm{cap}(c, i)$
4: if $Y^c - Z_i = \mathrm{cost}(i, c)$, then let $\mathsf{flow}(c, i) =
   \min(1, \max(\mathsf{height}(c) - \mathsf{height}(i), 0)) \cdot \mathrm{cap}(c, i)/(1 + \epsilon)$
```

Note that note 3 is the algorithm that is used in real-time to serve the requests.

4.3 Effect of supply scaling

Notice in applying Theorem 8, we scale up the demands before running the height algorithm but we use the same supply. There might be also other reasons for scaling up the demands. For example, suppose we do not know the exact supply of the supply nodes (i.e., the s_i values), but we may have an estimate of each supply node which we call s'_i. For example suppose we know that with high probability, $s_i \geq \frac{s'_i}{1+\epsilon}$. Under, such a circumstance we may want to scale down all the supply estimate s'_i by some factor $1 + \alpha$ (or equivalently scale up all the demand constraints, compute the flow and then scale the flow down by the same factor) to make sure that with high probability we can always meet the the supply constraints.

Scaling the demands (or supplies) may affect the value of our objective function. The next result gives an upper bound on the change of the objective function when we scale the demands by an arbitrary factor.

THEOREM 9. *For a given input instance which is α-robust, and with the optimal objective function value* OPT, *if we scale the demands by $1 + \alpha$, then the new optimal value of the objective function* OPT$'$ *is at most $|C|^2\alpha{\cdot}\max_{c\in C} d^c$ away from* OPT *and that is tight.*

PROOF. Consider the flow corresponding to the optimal allocation of the original input (before scaling up the demands). Now, for each contract c one by one, we scale $\mathrm{cap}(s, c)$ by $1 + \alpha$. Since the input instance is α-robust, we should be able to augment the flow by αd^c which is the amount of increase in the capacity of (s, c). By applying the augmentation we may change the flow of each of the other contract nodes by at most αd^c which means the value of the objective function may increase at most by αd^c. Since there are $|C|$ augmentations and each augmentation may affect the flow of all the other contract nodes, in the worst case the total change in the objective function value is upper bounded by $|C|^2\alpha \max_{c\in C} d^c$. The tight example is omitted in this version. $\square$

5. A GREEDY SOLUTION FOR L_1 ALLOCATION WITH SOFT DEMAND

In this section we present a simple greedy approach for a slightly generalized version of the L_1 penalty function. In this version, we assume the demand constraints are soft, meaning, it is possible to satisfy a contract partially. (The search engine, however, should pay extra amount per unsatisfied demand, similar to the model used in [9]; we will capture this by a parameter β.) We also show how to preprocess and then reconstruct the greedy solution using $O(|C| + |I|)$ space for storing the preprocessed information and $O(\max_{i\in I} |\mathrm{con}(i)|)$ time to recompute the allocation.

We assume each contract has its own weight $w^c = \frac{W^c}{d^c}$ and to implement the soft demand constraint, we assume an amount of $\beta \cdot w^c$ is paid for each impression that cannot be allocated to the contract c, where the factor $\beta \geq 1$. We now present a greedy algorithm and prove that the total cost of its solution is at most $1+\beta/2$ that of the optimal solution. The greedy algorithm proceeds as follows.

Algorithm 4 Greedy allocation

```
1: repeat
2:    Let $c$ be the next contract with the largest $w^c$
3:    Give the most balanced allocation possible to contract
      $c$
4: until all contracts are considered
```

We now show that this algorithm obtains an approximation to the optimum.

LEMMA 10. *Algorithm 4 is a $1 + \beta/2$-approximation for the L_1 penalty function for the soft demand case with factor β.*

PROOF. The proof is based on charging. We start by defining some notation. As usual, for a given allocation A, let A_i^c be the number of allocations from impression i to contract c. In an allocation A, we call a contract c on impression i as *under-represented* if $A_i^c < \delta_i^c$; let $\mathsf{under}_i^c(A) = \max(0, \delta_i^c - A_i^c)w^c$. Similarly, we call c *over-represented* on i if $A_i^c > \delta_i^c$ and let $\mathsf{over}_i^c(A) = \max(0, A_i^c - \delta_i^c).w^c$. Let Opt denote the optimal allocation and Greedy denote the greedy allocation. Two kinds of penalties are considered in the objective function: Unbalance part and partial contract allocation. Now we make the following claim: in any allocation A, we have $\mathsf{unfair}(A) \le 2\sum_{i\in I, c\in C} \mathsf{under}_i^c(A)$. The claim holds because for any contract c, $\sum_{i\in I} \mathsf{over}_i^c(A) \le \sum_{i\in I} \mathsf{under}_i^c(A)$, where the inequality changes to equality when d^c is completely satisfied in A. In addition,

$$\mathsf{unfair}(A) = \sum_{i\in I, c\in C} (\mathsf{under}_i^c(A) + \mathsf{over}_i^c(A)).$$

This means that the unbalance of our solution is at most twice as under representativeness of our solution. Also, since the allocation is greedy, the amount of under-balance on each impression in greedy is lower than any other allocation. I.e., $\sum_{i\in I, c\in C} \mathsf{under}_i^c(\text{Greedy}) \le \sum_{i\in I, c\in C} \mathsf{under}_i^c(A)$ for any allocation A including Opt.

Now, let us consider the total cost of under-balance on impression i in the greedy solution,

$$\mathsf{under}_i(\text{Greedy}) = \sum_{c\in \text{con}(i)} \mathsf{under}_i^c(\text{Greedy}).$$

Even if we do not accommodate these contracts in any other impression and pay the β factor instead of allocating them, the total amount would be at most $\beta \cdot \mathsf{under}_i(\text{Greedy})$. Now the total value of the objective function for the greedy solution is at most $(\beta+2)\sum_{i\in I} \mathsf{under}_i(\text{Greedy})$. From the earlier argument, we know that $\sum_{i\in I} \mathsf{under}_i(\text{Greedy}) \le \text{Opt}/2$. These imply that the greedy solution is a $1+\beta/2$ approximation for the L_1 penalty with soft demand. □

5.1 Reconstructing the greedy solution

Next, we show how we can reconstruct the greedy solution by storing only $O(|C|)$ preprocessed information. The running time for reconstructing the allocation based on stored information is $O(\max_{i\in I} |\text{con}(i)|)$. In the preprocessing phase, we compute the greedy allocation as described in Section 4. The stored information for each contract c is $\sum_{i\in \text{imp}(c)} \mathsf{over}_i^c(\text{Greedy})$. The reconstruction is as follows:

Algorithm 5 Greedy reconstruction (i)

```
1: {A new impression from bucket $i$}
2: For all contracts $c$, set $\mathsf{over}^c = \sum_{i\in \text{imp}(c)} \mathsf{over}_i^c(\text{Greedy})$
3: $C' = \{c \in \text{con}(i) \mid c$ is under-represented$\}$
4: if $C' \ne \emptyset$ then
5:     Assign the new impression to $\arg\max_{c\in C'} w^c$
6: else
7:     $C' = \{c \in \text{con}(i) \mid \mathsf{over}^c > 0\}$
8:     Assign the new impression to $\arg\max_{c\in C'} w^c$
9:     $\mathsf{over}^c = \mathsf{over}^c - 1$.
```

As described earlier, we need to keep track of only one variable for each contract. Also at each impression arrival, in the worst case we have $O(\max_{i\in I} |\text{con}(i)|)$ processing time. With similar argument given in Lemma 10, we can show that the computed solution here is also $1+\beta/2$-approximation for the balanced allocation with soft demands.

$\lvert I\rvert$	13,414
$\lvert C\rvert$	14,880
$\lvert E\rvert$	1,113,776
$(1/\lvert I\rvert)\sum_{i\in I} \text{con}(i)$	83.03
$(1/\lvert C\rvert)\sum_{c\in C} \text{imp}(c)$	74.85
$\sum_{i\in I} s_i$	184×10^9
$\sum_{c\in C} d^c$	89×10^9

Table 1: Impression–contract graph G: characteristics.

6. EXPERIMENTS

In this section we perform an experimental evaluation of our algorithm on large real-world datasets. The goal of the experiment is three-fold: (i) to demonstrate the practical feasibility and the scalability of our min-cost based algorithm, on large instances of the problem, especially compared to using a black-box linear/convex programming solution, (ii) to compare the performance of our algorithm to that of baselines such as Greedy (Section 5) and its heuristic variants, and (iii) to understand the space savings enabled by the reconstruction procedure.

Data and implementation. The datasets consist of various subsets of actual impressions buckets and advertiser contracts from Yahoo!'s display advertising system. For simplicity of exposition, we discuss the results for the largest graph in this collection. A gist of results for other graphs is given in Table 4.

The basic characteristics of the largest graph is shown in Table 1. We can see that the average degree of both the impression and contractor nodes is fairly high. The supplies and demands are integral and in our experiments, we treat all contracts equally, i.e., $w^c = 1$ for all $c \in C$. First, we ensure that this instance is in fact feasible by checking that the maximum flow in the instance is at least the sum of the demands of the contracts.

Next, we implement the min-cost based algorithm for minimizing the L_1 penalty. To do this, we use Goldberg's algorithm [11] for min-cost flow, obtained from `http://www.avglab.com/andrew/soft.html`. Since this implementation requires integral capacities and costs, we round the costs and capacities when constructing the graph in Figure 1. Additionally, since parallel edges are not supported in the implementation, we split each bottom edge in the middle layer of Figure 1 by introducing a new node; the costs and capacities of the newly created edges are the same as the original bottom edge.

The program was run on a single CPU 1.8GHz Linux machine with 16G memory. The min-cost based algorithm took 178 seconds to run on our input instance; we believe this can be significantly improved by careful fine-tuning. Besides providing insights into the problem itself, this is more time-efficient than applying a black-box LP solver (an untuned version of which took over 4000 seconds on our instance). Also, the LP solver is unhelpful in efficient reconstruction.

Performance. The performance of this algorithm is compared to that of the Greedy algorithm. Since the contracts are unweighted in our instance, we try two additional variants of Greedy: the contracts are ordered by decreasing demands (denoted Greedy$_<$) and increasing demands (denoted Greedy$_>$). The results are shown in Table 2. The second column reports the L_1 penalty as in (1). For com-

Algorithm	L_1 penalty	L_2^2 penalty ($\times 10^8$)	% unsatisfied demands
Theorem 2	1315	4.252	0
Greedy	2754	5.337	1.594
Greedy$_<$	5521	10.27	2.645
Greedy$_>$	746.3	2.457	0.673

Table 2: Performance of various algorithms.

parison purposes, we also compute the L_2 penalty as in (5) and report it in the third column. The fourth column contains the fraction of contracts that are unsatisfied by the allocation; notice that the min-cost based algorithm always satisfies *all* the demands, as long as the instance is feasible. As we see, the L_1 penalty of the min-cost based algorithm is much less than the first two versions of Greedy. Interestingly, the third version Greedy$_>$ does better in terms of both L_1 and L_2 penalties, but unfortunately *does not* satisfy *all* the demands. If however the demands are soft, i.e., there is a cost associated with not fully satisfying a demand, we can apply Lemma 10 to comment on the performance of Greedy.

To understand the solution produced by the min-cost algorithm better, we consider the distribution of the L_1 penalties per contract (i.e., for a contract c, the penalty value $w^c \sum_{i \in I} |y_i^c - \delta_i^c|$). Figure 2 shows this distribution: the x-axis represents the L_1 penalty values and the y-axis represents the fraction of contracts with this penalty value. As we see, the penalties of the min-cost flow-based algorithm is sandwiched between the greedy counterparts. This shows that in order to satisfy all the contracts, one has to incur fairly high L_1 penalties at least for some contracts. (Greedy and its variants do not take this into account and hence might end up not satisfying some contracts.)

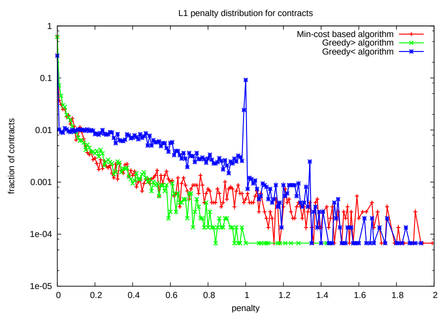

Figure 2: Distribution of penalties.

Efficacy of reconstruction. Finally, we look at the efficacy of the reconstruction. To study the space savings, we examine the values of the dual variables in (3). Table 3(a) shows the values. It is clear that the flow is either saturated or empty for more than 99% of the edges in the graph. Thus, the reconstruction becomes an efficient proposition since we only need to address the remaining 0.36% of the edges.

The graph G' in Algorithm 2 for our instance is fairly small; Table 3(b) shows its properties. From the data, we can store $(|C| + |I| + |C'| + |I'|)$ values (instead of $|E|$) and can reconstruct the solution. This is 3.1% — smaller by almost two orders of magnitude — of the space without the reconstruction! The time for reconstructing the flow for each

Condition on (i, c)	% edges
$Y^c < Z_i$	59.71
$Y^c > Z_i$	14.77
$Y^c = Z_i$	0.363

(a)

$\lvert I'\rvert$	2001
$\lvert C'\rvert$	4160
$\lvert E'\rvert$	5407
$(1/\lvert I'\rvert) \sum_{i \in I'} \mathrm{con}(i)$	2.70
$(1/\lvert C'\rvert) \sum_{c \in C'} \mathrm{imp}(c)$	1.30

(b)

Table 3: (a) Distribution of the dual variables. (b) Reconstruction graph G': characteristics.

G	$\lvert I\rvert$	$\lvert C\rvert$	m (K)	L_1 cost	time (sec)	$\lvert I'\rvert$	$\lvert C'\rvert$	$\lvert E'\rvert/\lvert E\rvert$
1	6683	7485	269	1252	26.5	1855	2351	1.23
2	3942	4461	104	811	8.03	932	1173	0.86
3	10720	11793	71	1278	95.2	2166	3586	0.63
4	10772	4455	264	136	25.9	370	805	0.21
5	3948	11862	260	4613	22.6	5157	2807	7.73
6	1318	13373	973	4524	8.40	5835	1236	6.47
7	12092	1509	890	14	5.77	53	93	0.06
8	1358	1500	104	258	0.23	265	350	0.52
9	12024	13416	906	1368	125.0	2483	4091	0.72

Table 4: Summary of results for nine other graphs.

edge is essentially negligible by applying Algorithm 3.

Results for other graphs. We present a summary of results for nine other graphs of varying sizes and characteristics. From Table 4, we see that our min-cost based algorithm performs uniformly well over all these graphs, especially in terms of the running time and the reconstruction size.

7. CONVEX PENALTY FUNCTIONS

In this section we describe the combinatorial solution for a general convex penalty function. Our solution is based on finding a convex cost flow. For simplicity of exposition, we describe the method for the L_2^2 function; the general method, however, works for any convex distance function.

For L_2^2 penalty function, the quadratic programming formulation of the problem is as follows.

$$\min \sum_c w^c \sum_i (y_i^c - \delta_i^c)^2, \tag{5}$$

subject to

$$\begin{aligned} \forall c, \sum_i y_i^c &= d^c \qquad \text{(demand)} \\ \forall i, \sum_c y_i^c &\le s_i \qquad \text{(supply)} \end{aligned}$$

We show how to solve (5) using a convex cost flow. The flow network, with capacity and cost on each edge, is constructed as follows. We create a node for each contract and a node for each impression. Now compute the completely balanced allocation for each contract. In this tentative allocation, some of the impressions could be overfull and some of them could have excess supply. The goal is to reallocate the contracts to impressions in the least expensive way (according to the L_2^2 penalty) such that none of the impressions is overfull. We represent the excess of impression i by $e_i = s_i - \sum_{c \in \mathrm{con}(i)} \delta_i^c$. If $e_i \le 0$, then we consider the impression node as a supply node and set its supply equal to $-e_i$. If $e_i \ge 0$, then we set the impression node as a demand node and set its demand equal to e_i. Also we connect each supply impression node

i to each contract c that is interested in that impression, setting $\text{cap}(c,i) = \delta_i^c$ and cost $2x^2W^c$, where x is the flow on that edge (Figure 3).

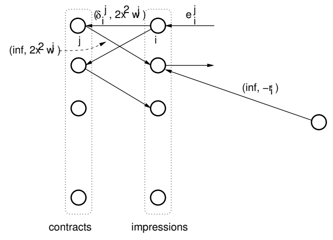

Figure 3: The network construction with (capacity, cost) on the edges for L_2^2.

From our construction, it is easy to see the following (proof similar to that of Theorem 2).

Lemma 11. *The solution to network described above is equivalent to the solution to (5).*

7.1 Efficient reconstruction for L_2^2

In [15], Vee et al. showed that the solution to the L_2^2 can be reconstructed using the Lagrange multipliers of the quadratic program. As part of their method, they show that, assuming that the Lagrange multiplier for supply constraint is p_i and the Lagrange multiplier for demand constraint is α^c, the best allocation that minimizes L_2^2 norm and is a feasible solution satisfies

$$\begin{aligned} y_i^c &= \max(0, g_i^c(\alpha^c - p_i)), \text{ where} \\ g_i^c(x) &= d^c \cdot \frac{s_i}{\sum_{i \in \text{imp}(c)} s_i} \left(1 - \frac{x}{d^c}\right). \end{aligned} \tag{6}$$

and store the values, we can reconstruct the allocation. They also showed that the solution to L_2^2 is unique, which means that the returned solution by the convex cost function also fits in their argument. Earlier in this section, we showed how to compute y_i^c values. Here, we give a solution sketch on how we can compute Lagrange multipliers combinatorially as well, given the primal solution. For simplicity, we denote $\phi = d^c \cdot \frac{s_i}{\sum_{i \in \text{imp}(c)} s_i}$ and $\sigma = \frac{s_i}{\sum_{i \in \text{imp}(c)} s_i}$. Rewriting (6), we have $y_i^c = \phi - \sigma \cdot (\alpha^c - p_i)$. Thus if we compute y_i^c, we can compute $\alpha^c - p_i$ as well. So, for all pairs (c, i) with $y_i^c > 0$, we first compute $d_i^c = \alpha^c - p_i$. We now construct a graph with one node for each Lagrange multiplier (in total $|I|+|C|$ nodes) and place an edge between two of them if there is a corresponding $y_i^c > 0$ for them; let the length of the edge be d_i^c. First of all it is easy to see that in each connected component, if we know the Lagrange multiplier for one of the nodes (for example p_i), the Lagrange multiplier for the rest of the nodes in the same component can be computed as well. So for each connected component, we assign some variable p to one of the nodes and compute the distance of all the nodes from that node. This way, the value of the Lagrange multipliers in each component can be represented by $p+d$ for which d is already computed and the only variable is p. So the only remaining question is how to choose p values in each component. Looking back at the Lagrangian for the quadratic program, and replacing y_i^c's and α^c and p_i by their computed values, the objective can be written as a linear function in terms of the selected representing values (p values) for each component. It is enough now to compute the best value for each of them assuming that we want to minimize the Lagrange objective function. We omit the details in this version.

8. CONCLUSIONS

In this paper we considered the problem of balanced allocation in a bipartite supply-demand setting. We worked with a simple formulation and circumvented the need for mathematical programming solutions by taking a more direct and combinatorial look at the formulation. We obtained a flow-based algorithm for the L_1 penalty case, the insights from which allowed us to precompute and store a linear amount of information such that the allocation can be reconstructed in constant time per edge. We extended the flow-based algorithm to the general convex function case.

Interesting directions for future work are: Is there a non-flow based solution to the L_1 penalty case, assuming certain structure on the contracts and/or impressions? Can sampling be used to approximate the L_1 penalty case and obtain an even more efficient solution? Is there a regular (i.e., not convex) flow solution to the L_2^2 penalty case?

9. REFERENCES

[1] A. Agrawal, P. N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 134–144, 1991.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, 1993.

[3] M. Babaioff, J. Hartline, and R. Kleinberg. Selling banner ads: Online algorithms with buyback. In *Proc. 4th Workshop on Ad Auctions*, 2008.

[4] R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *J. Algorithms*, 2(2):198–203, 1981.

[5] F. Constantin, J. Feldman, S. Muthukrishnan, and M. Pál. An online mechanism for ad slot reservations with cancellations. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1265–1274, 2009.

[6] N. R. Devanur, C. H. Papadimitriou, A. Saberi, and V. V. Vazirani. Market equilibrium via a primal-dual algorithm for a convex program. *J. ACM*, 55(5), 2008.

[7] N. R. Devenur and T. P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *EC '09: Proceedings of the tenth ACM conference on Electronic commerce*, pages 71–78, New York, NY, USA, 2009. ACM.

[8] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 97(1):242–259, 2007.

[9] U. Feige, N. Immorlica, V. Mirrokni, and H. Nazerzadeh. A combinatorial allocation mechanism with penalties for banner advertising. In *Proc. 17th International Conference on World Wide Web*, pages 169–178, 2008.

[10] A. Ghosh, P. McAfee, K. Papineni, and S. Vassilvitskii. Bidding for representative allocations for display advertising. In *Proc. 5th Workshop on Internet and Network Economics*, 2009.

[11] A. V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *J. Algorithms*, 2:1–29, 1997.

[12] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.

[13] K. Jain and V. V. Vazirani. Eisenberg-Gale markets: Algorithms and structural properties. In *Proc. 39th Annual ACM Symposium on Theory of Computing*, pages 364–373, 2007.

[14] P. McAfee and K. Papineni. Maximally representative allocation for guaranteed delivery advertising campaigns, 2008. Manuscript.

[15] E. Vee, S. Vassilvitskii, and J. Shanmugasundaram. Online allocation with a compact plan, 2008. Manuscript.