AD DELIVERY WITH BUDGETED ADVERTISERS: A COMPREHENSIVE LP APPROACH

Zoë Abrams
Yahoo!, Inc.,
Sunnyvale, CA 94089
za@yahoo-inc.com

Ofer Mendelevitch
Yahoo!, Inc.,
Sunnyvale, CA 94089
oferm@yahoo-inc.com

S. Sathiya Keerthi
Yahoo!, Inc.,
Sunnyvale, CA 94089
selvarak@yahoo-inc.com

John A. Tomlin
Yahoo!, Inc.,
Sunnyvale, CA 94089
tomlin@yahoo-inc.com

ABSTRACT

We study a comprehensive framework for sponsored search which incorporates advertiser budgets, query frequency forecasts, and pricing and ranking schemes. We propose a linear program for optimizing revenue (or the total value to advertisers) that has an exponential number of variables; however, we describe how it can be solved efficiently using column generation. The formulation is easily extendable to various levels of problem complexity, adaptable to dynamic environments, fast, and works well in terms of practical considerations. Simulations show significant improvements in revenue and efficiency.

Keywords: column generation, sponsored search, budgets, advertising

1. Introduction

Electronic commerce is thriving [Kim 2004; Chen 2003; Alczak 2006] and one of the driving forces behind its success is internet advertising. Search engine companies earn millions of dollars every day by auctioning off advertisement slots. Internet advertising is primarily accomplished through Sponsored Search, ads placed on publisher pages and banner ads. In this paper we will primarily be concerned with Sponsored Search, but we note that scheduling of ads for third party publishers can benefit from a very similar treatment. In Sponsored Search there are several sources of data which must be considered. Firstly, there are the actual bids by advertisers for the search terms (queries), secondly the distribution of query frequencies and finally the advertiser budgets, when these are declared.

The query frequencies limit the number of times a search engine can display its advertisers. Note that, unlike the bids and budgets, query frequencies are not under the control of the advertisers or, for that matter, of the search engine. It is well known [Silverstein 1999] that search engine query frequency distribution typically has relatively few queries with large volume and revenue, and a very large number of queries with extremely low volume. Therefore, we overcome most of the uncertainty in query volumes by selecting a relatively small subset of queries whose near-term volumes are easy to forecast, yet still constitute a large amount of the overall revenue.

Advertisers or their agents, on the other hand, do have the ability to control their budgets. An advertiser's budget may constrain the number of times their ads appear, even when they have made a high bid on a query term. One might ask why they would wish to do so. There are several possible reasons, among them: protection against click-fraud, an over-all company advertising budget, and the desire to control the allocation of that budget between various media and campaigns. Whatever the reason, the search engine must determine which advertisers to display for which queries, given these constraints.

Pricing and ranking are additional parameters of the system that influence revenue. The VCG mechanism [Vickrey 1961; Clarke 1971; Groves 1973] can be applied, but in practice search engines predominantly use the *generalized second price* (GSP) auction, described in more detail by Edelman et al. [Edelman 2006]. Advertisers are ranked according to the product of the price they bid for receiving a click, and a quality score. Each is charged a price per click equal to the minimum they would have had to pay to maintain their rank.

The problem we consider is how to allocate advertisers to queries such that budget constraints are satisfied and efficiency or revenue is maximized. This problem is posed as a linear program that takes a global view, and coordinates advertiser spend across the chosen time-period, such as the next hour, or an entire day. With the combined knowledge of forecast query volumes, advertiser budgets, advertiser bids and the pricing and ranking algorithm, we

formulate a comprehensive mathematical framework. Because of the flexibility of *LP* models, we are able to extend our approach beyond its original form to take onto consideration multi-level budgets, dynamic changes in the data, and a number of other practical extensions.

1.1. Related Work

Incorporating advertiser budgets into the marketplace design is recognized as crucial, and a growing amount of research addresses this subject. Several recent papers have considered the effect of budgeted advertisers specifically within the context of internet keyword auctions (e.g. [Borgs 2005; Abrams 2006]). The fields of on-line and approximation algorithms have also approached the topic. The widely noted paper [Vazirani 2005] presents an online algorithm, with a competitive ratio $1-\frac{1}{\varepsilon}$, when the volume and sequence of queries is unknown. Mahdian et al. [2007] extend this work by considering a tradeoff depending on the level of accuracy in volume predictions.

There has been a considerable amount of work done in the related field of (one-off) multi-item combinatorial auctions, leading to algorithms which are practical and efficient in a wide variety of settings. (see [Schrage 2001] for a useful survey). These algorithms typically employ linear or integer programming (*LP/IP*), exploiting a very mature and efficient group of technologies. Although the characteristics of the sponsored search problem we consider make it difficult to apply the known techniques directly—for instance the frequently repeated nature of the auctions—the approach is appealing. In particular the work of Dietrich and Forrest [2001], which uses column generation to determine the set of winning bids, is suggestive.

1.2. Motivating Example

To illustrate how critical the proper consideration of advertiser budgets might be, and how poorly a greedy algorithm might perform in the presence of these budgets, we examine the following highly simplified example. Suppose we have two queries q_1, and q_2. A single advertiser is displayed for each query, there is a reserve price of $C_1-\varepsilon$, all advertisers have the same expected clickthrough rate, and the relevant bids and budgets are shown in *Table 1*.

Table 1: Bids and budgets

Bidder	**Bid for** q_1	**Bid for** q_2	**Budget**
b_1	$C_1+\varepsilon$	C_1	C_1
b_2	C_1	0	C_1

As *Table 2* shows, a straightforward application of GSP that displays the highest bidder is not optimal, both in terms of efficiency and revenue. Let us assume that within the budgets' time intervals, q_1 appears, followed by q_2. Then bidder b_1 would pay the second bid price C_1 (bid by bidder b_2) and exhaust his budget on query q_1. When query q_2 arrives, no bidder is eligible for the query. Now consider the alternative allocation that shows b_2 for q_1, producing revenue $C_1-\varepsilon$, then shows b_1 for query q_2, also producing revenue $C_1-\varepsilon$ for a total revenue of $2C_1-\varepsilon$, nearly double the revenue of the greedy allocation. Thus, a more global viewpoint, one which takes into account the keywords throughout the time period, and the budget situation for each advertiser, can lead to increased efficiency and revenues. When this simple example is complicated many fold by thousands of queries, bidders and budgets, the potential for inefficiency is obvious.

Table 2: Allocation options

Allocation	**Shown for** q_1	**Shown for** q_2	**Total Revenue**	**Total Efficiency**
Greedy	b_1	-	C_1	$C_1+\varepsilon$
Optimal	b_2	b_1	$2C_1-2\varepsilon$	$2C_1$

1.3. Paper Outline

In this paper, we start out defining the model, problem and *LP* formulation in Section 2. We then describe the core technology in Section 3, i.e. the column generation algorithm that is used to solve the *LP*. One advantage of our approach is that the problem formulation and algorithmic solution are quite flexible and extensible. Section 4 describes three possible extensions: providing guarantees for individual bidders, allowing more flexible types of budget specifications, and accommodating pricing schemes based on the Vickrey-Clarke-Groves mechanism. The presented approach is also adaptable in the face of dynamic environments (described in Section 5). Next, in Section 6,

we move to practical considerations in implementing our solution. Finally, we end with describing our simulation results in Section 7.

2. Model and Problem Definition

Let the auction marketplace consist of a set of queries $Q = \{q_1, q_2, ..., q_N\}$ and bidders $B = \{b_1, b_2, ..., b_M\}$. We usually use the index i to denote query q_i and the index j to refer to bidder b_j. The *bidding state* of the marketplace at time t is defined by a (sparse) matrix A^t, where A^t_{ij} is the bid amount that the j-th bidder's is bidding on the i-th query q_i. For simplicity of analysis we assume a static bidding state ($A^t = A$) over some time-slot. While realizing that in practice this is not true due to bid management (either manually or by software), simulations suggest the effects of these types of changes are negligible. We will accommodate this dynamic aspect by frequently resolving our model as the data evolves, as is done in many other applications (see Section 2). We also assume that bids do not change as the allocation rule changes (i.e. this is not an equilibrium analysis). For each bidder b_j, we denote by d_j the daily budget limit specified by the bidder. d_j is an account level limit, i.e., it represents a spend limit across all queries for that account[1]. If a budget is not specified, we refer to this bidder as an *unbudgeted* bidder and set $d_j = \infty$.

Given a time-slot of interest, let v_i be a deterministic estimate of the number of times each query q_i will appear within that time slot. For each query, we define the *bidding landscape* as an ordered set of bidder indices $L_i = \{j_p : j_p \in B, p = 1, ..., P_i\}$, where the indices j_p are sorted by some ranking function, and P_i is the number of bidders in the landscape for query q_i.

In principle, we could now formulate an optimization model in which the variables corresponded to the number of times each available ad was shown with each query. However, such a model would require complex constraints and auxiliary discrete features to ensure the ordering required by the bidding landscapes for each query. We quickly abandoned such an approach in favor of a column-oriented approach which enforces the precedence explicitly. This modeling approach has a considerable history, and was quite recently used in a related model for item allocation in combinatorial auctions (see [Dietrich 2001]). The algorithm here is quite different, since the payoffs are not deterministic and the auction is frequently repeated, among other features, but the spirit is similar.

We now define the crucial concept of a *slate* of ads corresponding to (and in fact a subset of) the bidding landscape. These slates will correspond to the columns and variables of the linear program (*LP*) we formulate below. Each bidding landscape L_i is mapped into a set of slates L_i^k, each being a unique subset of L_i which can be obtained by deleting members of L_i (while maintaining the ordering) and then truncating (if necessary) to P_i^k (at most P) members. More formally, the k-th slate for ad i includes a unique subset (of length P_i^k) of the indices of L_i, and is defined as $L_i^k = \{j_l^k : j_l^k \in L_i, l \le P_i^k \le P\}$, where P is the maximum number of slots available for advertising on the page. The indices in L_i^k are ordered as in L_i (i.e., in order of ranking). By convention, if there are less than $P+1$ members an additional dummy member, bidding the reserve price, may be added for the purpose of computing second-bid prices.

Two ranking (ordering) methods have been commonly used. The first and older method, sometimes known as the "Overture method" is *bid-ranking*, so that (by a slight abuse of notation):

$$A_{i1} \ge ... \ge A_{ij} \ge ... \ge A_{iP_i}$$

This scheme has now been generally superseded by expected *revenue-ranking* where the bidders on the term are ranked by product of the bid value A_{ij} and a *quality score* or *clickability* value Q_{ij} which is assumed to incorporate the likelihood of the ad being clicked on, based on relevance of the ad to the query, among other factors. The bidders are thus ordered so that:

$$A_{i1}Q_{i1} \ge ... \ge A_{ij}Q_{ij} \ge ... \ge A_{iP_i}Q_{iP_i}$$

Hence in any slate k for query i we expect the subset of ads chosen to also satisfy:

$$A_{ij_1^k}Q_{ij_1^k} \ge ... \ge A_{ij_p^k}Q_{ij_p^k} \ge A_{ij_{p+1}^k}Q_{ij_{p+1}^k} \ge ...$$

The pricing scheme we use charges a price per click equal to some small increment plus the bid times quality

[1] We could also associate budgets with other entities such as a campaign (see Section 2).

score of the advertiser one slot below, divided by the bidder's own quality score. This scheme ensures that a bidder pays just enough to hold his position in the slate. Precisely, the price per click (PPC) of bidder j_p, denoted $PPC^k_{ij_p}$, is:

$$PPC^k_{ij_p} = A_{i,j^k_{p+1}} \cdot \frac{Q_{i,j^k_{p+1}}}{Q_{i,j^k_p}} + \varepsilon$$

In practice ε is some small quantity, such as a penny. Note that this is a modified *second bid auction* where the price per click actually paid depends on the next bid, *and* the ratio of clickabilities. Since setting all the Q_{ij} to 1 would result in the same slate and PPC as if we had used bid ranking, we shall ignore bid-ranking as a special case and consider only revenue ranking.

Let us also introduce a *click through rate* (CTR) denoted $T_{i,j^k_p,p}$ for the rate at which the ad from bidder j^k_p in position p in slate k for query i is clicked on per showing of the slate. These CTRs are estimated based on historical click data as well as other factors, and will be used both in the algorithm and the simulation described in Section 3. Assuming independence of the CTRs, we can now express the expected revenue-per-search (rps) in our model for this slate and this query as the sum of the individual expected revenues per click:

$$r_{ik} = \sum_{p=1}^{P^k_i} A_{i,j^k_{p+1}} \cdot \frac{Q_{i,j^k_{p+1}}}{Q_{i,j^k_p}} \cdot T_{i,j^k_p,p} \tag{1}$$

We now introduce the variables x_{ik}, which represent the number of times slate L^k_i appears in the given time slot. The expected total revenue for the time-slot, over all queries, is therefore:

$$\sum_{i=1}^{N} \sum_{k} r_{ik} x_{ik} \tag{2}$$

We represent the *total spend* for each bidder j as

$$\sum_{i=1}^{N} \sum_{k} c_{ijk} x_{ik} \tag{3}$$

where:

$$c_{ijk} = \begin{cases} A_{i,j^k_{p+1}} \cdot \dfrac{Q_{i,j^k_{p+1}}}{Q_{i,j^k_p}} \cdot T_{i,j^k_p,p} & 0 < p \le P^k_i \le P \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

is the expected cost to bidder j of appearing in position p and in slate k for query i. Observe that this expectation is with respect to the click through rate, and that the quality scores are deterministically known.

2.1. Linear Programming Formulation

We may now formally define the following linear programming (*LP*) problem:

Indices

$i = 1,...,N$	The queries
$j = 1,...,M$	The bidders
$k = 1,...,K_i$	The slates (for query i)

Data

d_j	The total budget of bidder j
v_i	Expected number of occurrences of keyword i
c_{ijk}	Expected cost to bidder j if slate k is shown for keyword i
r_{ik}	Expected revenue from slate k for keyword i

Variables

x_{ik}	Number of times to show slate k for keyword i

Budget Constraints

$$\sum_{i} \sum_{k} c_{ijk} x_{ik} \le d_j \quad \forall j \tag{5}$$

Query Volume Constraints

$$\sum_k x_{ik} \le v_i \ \ \forall i \tag{6}$$

Revenue Objective

$$\text{Maximize} \sum_i \sum_k r_{ik} x_{ik}$$

2.2. Alternate Objective Functions

While maximizing expected overall revenue is obviously attractive from the auctioneer's point of view, it may not appear so attractive to the individual bidders. However the *LP* approach can accommodate a variety of alternate objective functions. One possibility is to maximize expected efficiency under the assumption that the bidders have expressed their true value for a click by the bids[2]. This would correspond to an *LP* objective of:

Value Objective

$$\text{Maximize} \sum_{i,k} \sum_p A_{i,j_p^k} \cdot T_{i,j_p^k,p} \cdot x_{ik} .$$

Note that this objective has coefficients computed from the first prices, not second prices.

Even more simply we may decide to optimize the number of clicks obtained. This is accomplished by using an *LP* objective:

Clicks Objective

$$\text{Maximize} \sum_{i,k} \sum_p T_{i,j_p^k,p} \cdot x_{ik} .$$

For these particular two objectives, we could also reformulate the *LP* to have a polynomial number of variables and constraints, versus using the more involved column generation approach. Regardless of how the solutions are computed, it is obvious that other objectives could be formulated. If we use the column generation approach, we might take some composite weighted sum of the value, clicks, and revenue objectives. In general the "Value Objective" has advantages when considering the problem from the perspective of economics.

3. Column Generation Algorithms

It is clear that the number of potential columns in the above *LP* could be very large indeed. For each query, the number of possible slates is exponential in the number of budgeted bidders on the query, and there are many queries. It is therefore impractical to enumerate all the possible columns corresponding to the variables x_{ik}. If there were no binding budgets, then the optimal solution would be the trivial one consisting only of the top P bidders from the landscape (called a *base slate*) being shown for each query. In fact we would expect, and experiment confirms, that for many queries, the base slate will be the only one shown even when there are active (binding) budget constraints, while some others use multiple slates—usually not more than a handful. The trick is to know which handful. The same situation occurs in many other *LP* applications where each column represents one of many possible complex activities. Early examples included models where the column represented a path through a network or a cutting pattern for cutting up stock sizes of material. These particular models require that small auxiliary models be solved to generate relevant columns—shortest path problems in the former case, and a "knapsack" problem in the latter.

Using a conventional column-generation approach [Dietrich and Forrest 2001; Lubbecke and Desrosiers 2005], we do not attempt to generate every slate L_i^k a priori, but to generate an initial subset (say the L_i) and then generate columns as needed using the dual values of the linear program. Considering to begin with the revenue maximizing objective, let π_j be the marginal value for bidder j's budget, i.e. the simplex multipliers [Dantzig 1963] for the j^{th} constraint *(5)* and let γ_i be the marginal value for the i^{th} keyword, i.e. the simplex multipliers for the i^{th} constraint *(6)*, then a column corresponding to slate L_i^k (and hence to variable x_{ik}) can be profitably introduced into the model if:

$$r_{ik} - \sum_{j \in L_i^k} c_{ijk} \pi_j - \gamma_i > 0 \tag{7}$$

For each keyword i we seek to maximize $r_{ik} - \sum_{j \in L_i^k} c_{ijk} \pi_j$ (or equivalently, minimize $\sum_{j \in L_i^k} c_{ijk} \pi_j - r_{ik}$) over all legal slates L_i^k. If a slate is found such that *(7)* is satisfied, the corresponding slate and its variable are introduced into the problem. If no such slate exists (for any i) then an optimal solution has been obtained.

[2] Although this is not always true, we believe it is a reasonable assumption in this context, when actual values are unavailable.

Looking at the structure of the coefficients in *(7)* and the definitions of r_{ik} and c_{ijk} from *Equations (1) and (4)*, respectively, we see that the subproblem is to maximize (for the given π_j):

$$F_{ik}(\pi) = \sum_{p=1}^{P_i^k} T_{i,j_p^k p} \cdot A_{i,j_{p+1}^k} \cdot \frac{Q_{i,j_{p+1}^k}}{Q_{i,j_p^k}} \cdot (1-\pi_{j_p^k}) \tag{8}$$

over all legal slates L_i^k. When the number of budgeted bidders for a query is not too large this may be done by enumerating all legal subsets of L_i^k, evaluating *(8)* on the fly, until a L_i^k is found for which $F_{ik} > \gamma_i$. The corresponding column is then added to the problem. If no new column satisfying this condition is found, the present solution is optimal. If there are more than a few such legal subsets, we may need to algorithmically generate columns (slates) which maximize *(8)* and test whether they satisfy $F_{ik} > \gamma_i$. If the maximizing slate does not satisfy this inequality we may pass on to the next query, since no improving slate can be found for the present set of π_j values. If this is true for all queries, then the current solution is optimal.

The overall algorithm proceeds by generating improving sets of slates in this way, then re-solving the *LP*, until no further improvement is possible, or some other heuristic termination criterion is met (such as percentage improvement in the objective).

Column generation extends to the alternate objective functions we have suggested. In particular, if the maximum value objective is chosen, we see that the objective function coefficients are:

$$\sum_p A_{i,j_p^k} \cdot T_{i,j_p^k,p} \tag{9}$$

which must replace r_{ik} in *(7)*. The function we wish to maximize in this case is then:

$$F_{ik}(\pi) = \sum_{p=1}^{P_i^k} T_{i,j_p^k p} \cdot (A_{i,j_p^k} - A_{i,j_{p+1}^k} \cdot \frac{Q_{i,j_{p+1}^k}}{Q_{i,j_p^k}} \cdot \pi_{j_p^k}) \tag{10}$$

In much the same way, we may generate columns for the maximum clicks objective by replacing A_{i,j_p^k} by 1 in *(10)*.

The generation of the slate that maximizes the utility function in *(8)* or the one in *(10)* is a form of cardinality constrained knapsack problem[deFarias 2003], complicated by the ordering requirement and the fact that only certain items can be omitted. This problem is amenable to a dynamic programming approach which given the dimensions of the problems we are considering here (perhaps a dozen ads chosen from a few dozen candidates) is fast enough to be solved many thousands of times in the column generation process. Below we describe network path optimization algorithm for this approach. Fuller details as well as a backward recursion dynamic programming algorithm and extensions of these algorithms to other application settings are given in [Keerthi 2006].

3.1. An Optimal Path Approach

To simplify notations we take one query i and let m, n to be the numbers of positions and bidders for that query. Define the utility of having bidder j in position p and bidder $j_{\text{next}} > j$ in position $p+1$ as

$$u_{j,j_{\text{next}},p} = T_{i,j,p} \cdot A_{i,j_{\text{next}}} \cdot \frac{Q_{i,j_{\text{next}}}}{Q_{i,j}} \cdot (1-\pi_j) \tag{11}$$

For the alternate objective in *(10)* we just need to redefine u as $u_{j,j_{\text{next}},p} = T_{i,j,p} \cdot (A_{i,j} - A_{i,j_{\text{next}}} \cdot \frac{Q_{i,j_{\text{next}}}}{Q_{i,j}} \cdot \pi_j)$. For the sake of dealing with the last n^{th} bidder we define a dummy $(n+1)^{th}$ bidder, and it is the only allowed value of j_{next} for $j = n$. Then the determination of the optimal slate for query i to optimize the function in *(8)* (or *(10)*) can be rewritten as

$$\max \sum_p u_{j_p,j_{p+1},p} \tag{12}$$

The problem in *(12)* can be cast in the form of a longest path problem on a network with nodes $N_{j,p}$ for $p = 1,...,m$ and $j = p,...,n+1$, with $u_{j,j_{\text{next}},p}$ denoting the utility of going from $N_{j,p}$ to $N_{j_{\text{next}},p+1}$. We also define terminal nodes $N_{0,0}$ and $N_{n+1,m+1}$ with appropriately appended zero utility edges so that the problem transforms to the problem of finding the longest path from $N_{0,0}$ to $N_{n+1,m+1}$. See *Figure 1* and *Figure 2* for examples. Very efficient polynomial time algorithms are known for solving this problem. In our case the problem size is not big and so a simple

implementation suffices.

It is also useful to write down a backward recursion dynamic programming algorithm for solving *(12)*. Let $U(j,p)$ denote the optimal total utility of going from node $N_{j,p}$ to the terminal node, $N_{n+1,m+1}$. Then we can write the recursion,

$$U(j,p) = \max_{j_{\text{next}} > j} u_{j,j_{\text{next}},p} + U(j_{\text{next}}, p+1). \tag{13}$$

We can start the algorithm by setting $U(n+1,m+1) = 0$ and recurs backwards in j using *(13)*. Finally, when we get $U(0,0)$ we get the solution of *(12)* as well as the optimal slate.

3.2. Restricted Omissions

The longest path algorithm that we have just described assumes that any appropriate ordered subset of the ads may be chosen which fits within the slate size. In practice this may not always be true. It is normal to allow budgeted bidders to be held out of the notional auction—that is excluded from the slate. However, it is not obvious that this option should extend to unbudgeted bidders. A business decision may require that unbudgeted bidders not be excluded. We therefore require a means of specifying which ads (bidders) can be excluded. We accomplish this by specifying a *mask* or bit vector, which has a 1 if the ad can be excluded and a zero otherwise. We then modify the path optimization algorithm as follows. Since each arc in the network gives the utility of including a particular ad i in position p followed by ad j, we consider only arcs such that:

1. For each position p we allow i to assume the values from p up to the first ad in rank order which has a zero mask bit. Any subsequent ads are ignored for this p. This ensures that the unmasked ad with the highest rank is not excluded, but that lower ranked ads which are masked are not considered for the position p.
2. For each i chosen as above, the second index j shall only run from $i+1$ through the next unmasked ad. This ensures that if an ad i can be followed by an unmasked ad it will be the next in rank order.

This scheme corresponds to removing certain arcs from the network, or more simply implemented by modifying the longest path algorithm with the rules we have itemized. In *Figure 3* we show the reduced network obtained from *Figure 1* when we specify that $mask = (1,0,1,0,1)$. In practice however, we use the second technique of modifying the algorithm.

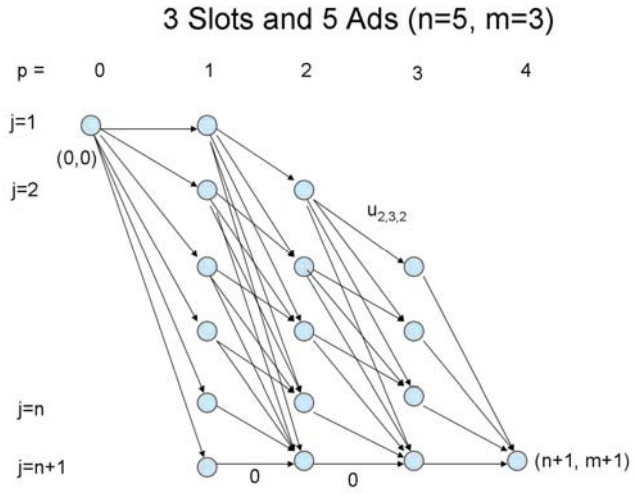

Figure 1: Network with $n > m$

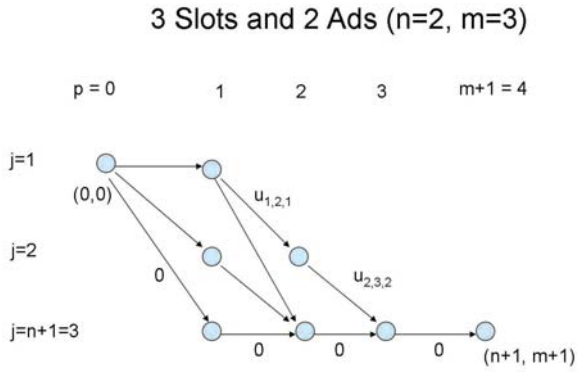

Figure 2: Network with $n < m$

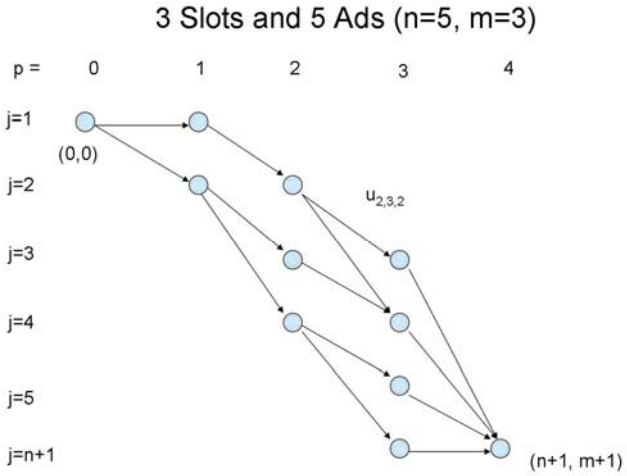

Figure 3: Reduced network with $mask = (1,0,1,0,1)$

4. Model Extensions

Although we manage to capture a great deal of the complexity of the system, there are still additional aspects of the problem which are not present in the proposed formulation. In this section, we expand the formulation to include some additional aspects from the real-world setting, and to accommodate alternative pricing functions.

4.1. Keeping Advertisers Happy

Thus far, the solution proposed does not adjust on behalf of the experience of any individual advertiser. Here, we present an approach for limiting the amount of change experienced by any single bidder.

Consider we want to ensure that bidder j receives a price-per-click of at most l_j. Then we can add the following constraint to our LP:

$$\sum_{i:\, j \in L_i} \sum_{k:\, j \in L_i^k} (c_{ijk} - l_j) x_{ik} \leq 0$$

This approach is easily adapted to constrain other aspects of the advertiser experience such as clicks received and utility. To receive at least c_j clicks, we add the constraint:

$$\sum_{i:\, j \in L_i} \sum_{k:\, j \in L_i^k} T_{i, j_p^k, p} x_{ik} \geq c_j$$

To ensure some amount of total utility u_j, if we assume the bid is a bidder's true value for a click, we add the constraint:

$$\sum_{i:\, j \in L_i} \sum_{k:\, j \in L_i^k} (A_{ij_p^k} - c_{ijk}) T_{i, j_p^k, p} x_{ik} \geq u_j$$

Of course, all these additional constraints may lead to an infeasible LP. To avoid this situation, the l_j, c_j, or u_j

values could be scaled up or down, respectively, based on their values obtained from some feasible solution (such as the solution for the greedy baseline algorithm). Alternatively, the requirements could be increasingly relaxed until a feasible solution is found.

4.2. Campaign and Account Budgets

It is often the case that a single advertiser is managing many different advertising initiatives simultaneously. For instance, a food vendor might have a catering service and also a mail order gift basket service. For each type of service the vendor has different advertisements, and different bids and different keywords. We refer each unit of advertisement combined with bids on keywords as a campaign. When many campaigns are connected to a single advertiser, we refer to the collection of campaigns as an account. An advertiser may have a budget requirement on its account, campaigns, or any combination of account and campaigns. The original model has only one kind of budget—the d_j —which we can consider as the lowest level of budget, i.e. a campaign budget. We now wish to superimpose account level budgets, which may constrain disjoint sets of campaigns. To do this, we introduce new sets:

$$C_l = \{ j_1^l, ..., j_{k_l}^l \} \quad l = 1, 2, ... \tag{14}$$

which are the sets of campaigns belonging to account l.

We then add account level budget constraints of the form:

$$\sum_{j \in C_l} \sum_i \sum_k c_{ijk} x_{ik} \le d_l \quad \forall l \tag{15}$$

where d_l is the budget for account l. With some matrix and variable manipulation, we can generate the variables and constraints in *Equation (15)* initially, then add on columns as needed.

4.3. Column Generation with VCG Pricing

In Section 1, an algorithm is described that finds a violated constraint in the *LP* from Section 1 in strongly polynomial time. This dynamic programming algorithm is based on the GSP pricing scheme [Edelman 2006]. Alternatively, a truthful pricing scheme might be used, such as VCG [Vickrey 1961; Clarke 1971; Groves 1973; Edelman 2006] or the laddered auction [Aggarwal 2006]. In VCG, the weighted price (meaning the price per click times the clickability) bidder j pays for position p in slate k for query i is:

$$V_{jpk} = (T_{ij_{p+1}^k p} - T_{ij_{p+1}^k (p+1)}) Q_{j_{p+1}^k} A_{j_{p+1}^k} + V_{j_{p+1}^k (p+1) k}$$

For more details on VCG pricing, see [Aggarwal et al., 2006; Clarke 1971; Edelman et al., 2006; Groves 1973; Vickrey 1961]. Using these prices, the maximization subproblem from *Equation (8)* becomes

$$F_{ik}(\pi) = \sum_{p=1}^{P_i^k} T_{i, j_p^k p} \cdot V_{jpk} \cdot (1 - \pi_{j_p^k}) \tag{16}$$

The slate that maximizes $F_{ik}(\pi)$ can be found using dynamic programming. In the following recursive formula, $U(j, s, V)$ is the maximum possible revenue if bidder j is placed in position s and pays price V:

$$U(j, s, V) = max_{h > j} (1 - \pi_j) T_{ijs} V + U(h, s+1, V - (T_{ihs} - T_{ih(s+1)}) Q_h A_h)$$

We can start the algorithm by setting $U(j, P+1, 0) = 0, \forall j$ and then recurs backwards, filling in each successively smaller position, for all bidders and prices that could potentially fit in that position. If every price is unique, the running time of the algorithm will be $\Theta(n^P)$. Or, the recursion produces a pseudo-polynomial algorithm with running time $\Theta(n^2 m \frac{\max_h Q_{ih} A_{ih}}{\delta})$, where δ is the smallest precision of any value in matrices A, T and Q. We could alternatively have running time $\Theta(n^2 m \frac{\max_h Q_{ih} A_{ih}}{\beta})$, where we can set β to any value and assume β-truthfulness VCG (i.e. advertisers don't have any value in gaming the system for a price differential of less than β).

5. Dynamic Environment

Sponsored search operates in a dynamic environment. Bidders come and go, and their bid and budget values can fluctuate as well. Query volumes are not fixed, and although they can be predicted with a high level of accuracy, it is likely that the traffic stream will not occur exactly as predicted. In this section, we present three enhancements to the problem formulation from Section 2. We start with a relatively simple enhancement that assumes future changes are known in the current time step. When the assumption that the future is known in advance is removed, we give an enhancement that is more complex and more computationally involved, but capable of addressing uncertainty. Finally, we describe techniques for managing uncertainty that are less computationally intensive.

5.1. Multiple Time Periods

So far, we have used query volumes to guide our solution, but have not used the *order of arrival* for these queries. It is not realistic to think we might know the exact order of query arrivals in advance. However, there is a temporal nature to queries. For instance, there are more queries for *pizza delivery* at 6pm than at 6am. The temporal aspect of queries can play an important role in helping to determine which advertisements to show when. This section presents a technique for improving the slate frequency solution by incorporating predictable, time dependent fluctuations in query volumes into the *LP* formulation.

Here, we assume that bids and bidder landscapes change over time, but these changes are known in advance or can be predicted from historical data with a high degree of accuracy. For instance, an advertiser may specify that they will pay $\$1$ for a particular keyword search between 11am and 4pm, and during all other hours, they want to withdraw their bid from that keyword.

To describe and implement this extension, we first expand our notation by adding a t superscript to indicate the time period ($t = 1,...,T$). Thus, for time period t, we let v_i^t be the query frequencies. The time t now determines the matrix of bids (A^t), clickabilities (Q^t), and clickthrough rates (T^t). The matrices are not simplified to constants over all time as in Section 2. The revenue (r_{ik}^t) and cost per bidder for each slate (c_{ijk}) are constructed using *Equations (1) and (4)*, respectively, except now the corresponding A^t, Q^t, and T^t for time t are used; let us call these new constants r_{ik}^t and c_{ijk}^t. We rewrite the variables, constraints and objectives from Section 1:

Variables

x_{ik}^t Number of times slate k is shown in time period t for keyword i

Budget Constraints

$$\sum_t \sum_i \sum_k c_{ijk}^t x_{ik}^t \le d_j \ \forall j$$

Query Volume Constraints

$$\sum_k x_{ik}^t \le v_i^t \quad \forall i,t$$

Revenue Objective

$$\text{Maximize} \sum_t \sum_i \sum_k r_{ik}^t x_{ik}^t$$

5.2. Periodic Readjustment

Our assumption in Section 1 that all changes are known ahead of time is not always true. Despite the best predictive tools, the system parameters may change in unpredictable ways, and even if forecasting was 100% accurate, the randomness due to choosing a slate randomly, creates some amount of variation.

To accommodate for unpredictable variation, we can resolve the *LP* solution throughout the course of the day, and update the slate frequencies accordingly. One can either apply an update only at time points when there has been significant inaccuracies in the forecast, or, at every time period, say at every 1 hour interval. Although this re-computation requires more computational resources, it allows the slate frequencies to adjust to changes in the environment as the day progresses.

More precisely, in hour t, we set $d_j^{\text{rem}}(t) = d_j - d_j^{\text{done}}(t)$, where $d_j^{\text{rem}}(t)$ is the budget remaining at time t and $d_j^{\text{done}}(t)$ the amount of budget that has already been spent. Similarly, let $v_i^{\text{rem}}(t)$ be the query volume remaining in the day for query i. This can be calculated by keeping track of $v_i^{\text{done}}(t)$ the query volume completed till time t and setting $v_i^{\text{rem}}(t) = v_i - v_i^{\text{done}}(t)$. A more accurate approach, which we use in our simulations, would be to estimate query volume for each hour of the day and to set $v_i^{\text{rem}}(t)$ to be the sum over hours left in the day, of the anticipated volume of query i during that hour. Now, at hour t, we use slate frequencies obtained by solving the following modified *LP*:

Variables

x_{ik} Number of times slate k is shown for keyword i

Budget Constraints

$$\sum_i \sum_k c_{ijk} x_{ik} \le d_j^{\text{rem}}(t) \ \forall j$$

Query Volume Constraints

$$\sum_k x_{ik} \le v_i^{\text{rem}}(t) \ \forall i$$

Revenue Objective

$$\text{Maximize} \sum_i \sum_k r_{ik} x_{ik}$$

We find that in simulations, resolving the linear program every hour, as opposed to once every day, leads to a 5% increase in revenue gains, where the percent increase is over the revenue gain when computing frequencies once a day. For more details, see *Figures 5 and 4* in Section 4.

5.3. Approximate Solution via Decoupling

The approach from Section 2 can be expensive, e.g., if we want to re-solve the full LP frequently. Here, we give a simpler sub-optimal strategy that is cheap to implement. In very simple terms, we use the original *LP* solution that was solved to get the portion of the budget that each advertiser should spend on each query. Then, periodically throughout the day, we run an *LP* for each query, to determine which slates to use just for that particular query, given the budget remaining of the portions allotted to that query by the original *LP* .

Let us now reformulate an equivalent *LP* by bringing in new variables for $D_{ij} = \sum_{k: j \in L_i^k} x_{ik}$, representing the portioned budget of bidder j for query i . It is necessary to assign these variables only for $j \in L_i$. The constraints from the LP in Section 1 can be equivalently written as:

Query-level Budget Constraints

$$\sum_{k: j \in L_i^k} c_{ijk} x_{ik} \le D_{ij} \ \forall i, j \tag{17}$$

High-level Budget Constraints

$$\sum_i D_{ij} \le d_j \ \forall j \tag{18}$$

Query Volume Constraints

$$\sum_k x_{ik} \le v_i \ \forall i \tag{19}$$

Suppose optimal values for the D_{ij} 's that satisfy this revised *LP* are all known. Then the constraints from *Equation (18)* can be eliminated, and the LP rewritten as a large number of separate, smaller, *LP*s, one for each query. The decoupled *LP* for a single query i uses *Equations (17) and (19)*, except D_{ij} are now constants:

Query-level Budget Constraints

$$\sum_{k: j \in L_i^k} c_{ijk} x_{ik} \le D_{ij} \ \forall i, j$$

Query Volume Constraints

$$\sum_k x_{ik} \le v_i \ \forall i$$

Revenue Objective

$$\text{Maximize} \sum_k r_{ik} x_{ik}$$

The solution for the above decoupled LP (i.e., the generation of optimal slates and their frequencies for given D_{ij}) is relatively simple, fast, and easy to implement.

We can use this decoupling of the LP to achieve the benefits of periodic adjustment from Section 2 without the high computational demands. Let t denote the time point in the day at which some disruption occurs and we want to repair the *LP* solution. Let us say we have kept track of $D_{ij}^{\text{done}}(t)$, which is the amount of money spent by bidder j on query i till time point t in the day. The remaining budget is $D_{ij}^{\text{rem}}(t) = D_{ij} - D_{ij}^{\text{done}}(t)$. We rewrite the *Query-level Budget Constraints*:

$$\sum_{k: j \in L_i^k} c_{ijk} x_{ik} \le D_{ij}^{\text{rem}}(t) \ \forall i, j$$

As bidders come, go and change their values, this decoupling isolates the impact of their changes. Updating $D_{ij}^{\text{rem}}(t)$ to best accommodate the dynamic environment is left as future work (see Section 8).

6. Practical Considerations for Slate Frequencies

The column generation algorithm produces fractional values for the variables x_{ij}. In practice, there is no such thing as a fractional query. Therefore, we divide slate occurrences by the query volume to produce slate frequencies, which we denote as $\theta_{ik} = \frac{x_{ik}}{V_i}$. Every time a query i arrives and must be served, we choose to use slate k with probability θ_{ik}.

6.1. Optimality of Slate Frequencies

The optimality of our solution is lost when we use the *LP* variables as slate frequencies[3]. Still, we find our algorithm works well in simulations because, typically, budgets are significantly higher than bids and query volumes are relatively large and predictable. Randomly rounding the x_{ik} variables to integers, according to their fractional remainders and such that they still sum to the query volumes, becomes more accurate as the budgets become larger, compared with the largest amount spent on any query. And, if the query volumes are also large, then choosing slates randomly from a distribution is not too different from choosing precisely that distribution (due to the law of large numbers). Also, the use of randomness here creates a more opaque system, which has other advantages such as being more difficult for bidders to manipulate.

6.2. Translating Slate Frequencies into Throttle Rates

Depending on the architecture of the system, implementing slate frequencies could potentially be involved and complex. In some architectures it may be easier to implement a *throttle rate* (i.e. some rate per query at which eligible ads are not considered as candidates for that query), which we denote as α_{ij}. In other words, once an advertiser j has been matched to query i, there is some probability (the throttle rate) that they will actually be withdrawn from the set of ads that are returned as matches for that query.

Slate frequencies are not equivalent to throttle rates. In fact, slate frequencies are strictly more descriptive than throttle rates. To see this, let us begin with a probabilistic tree model defined by throttle rates and derive corresponding slate frequencies. Recall the notations of section 2 and define $\hat{L}_i^k$ to include *all* bidders that are not throttled for a particular slate, including bidders that are not displayed because they are ranked below the cutoff position. More precisely,

$$\hat{L}_i^k = \begin{cases} L_i^k \cup \{ j_p \in L_i : p > P_i^k \} & \text{if } P_i^k = P \\ L_i^k & \text{otherwise} \end{cases}$$

For a given set of throttle rates, we create a slate for every possible subset of bidders and set $\theta_{ik} = \prod_{j \in L_i^k} (1-\alpha_{ij}) \prod_{j \notin \hat{L}_i^k} \alpha_{ij}$. Using this distribution over slates produces exactly the same outcome as having used the given throttle rates.

However, there is not always a transformation in the other direction. For instance, suppose the slate distribution shows bidders ranked 1 and 3 together with probability .5 and advertisers 2 and 4 together with probability .5. These are the only two slates in the distribution, so advertisers 1 and 2 are *never* shown together on a slate. However, any throttling scheme with a non-positive probability of showing ad 1 and also a non-positive probability of showing advertisement 2, as would have to be the case to replicate this slate distribution, would also have some strictly positive probability of showing ads 1 and 2 together. Therefore it is impossible to achieve every slate frequency distribution using only throttle rates. Since an exact translation from slate frequencies to throttle rates is not always possible, we next explore several heuristics for determining useful throttle rates given slate frequencies.

Set Throttle Rate = *LP* Probability of Display- One possible heuristic for obtaining throttle rates from slate frequencies is to simply set the throttle rate equal to one minus the probability that the advertisement is displayed using slate frequencies. We have:

---

[3] In fact, as the number of bidders grows, even an integral optimum can be arbitrarily less than the fractional optimum. Let $\frac{C}{d}$ be the number of bidders each with value C for being placed in the highest slot for query i and with budget $d << C$. There is only one query with one slot. The objective function of the optimum fractional solution has value C, obtained by setting each $x_{ik} = \frac{C}{d}$, where for every bidder there is a slate k containing only that bidder. In contrast, any integral solution has value 0.

$$\alpha_{ij} = 1 - \frac{\sum_{k: j \in L_j^k} x_{ik}}{v_i} \tag{20}$$

In practice, this approach works well, providing an increase in revenue that is roughly 98% of the increase when slates are used (see section 7 for more details).

There is the possibility that a simple modification could improve performance even more. The idea behind the modification is illustrated by the following example. Consider there are two bidders A and B with A ranked higher than B. There is only one query i that displays one advertisement. The *LP* displays slate A half the time and slate B the other half of the time. Using *Equation (20)*, $\alpha_{iA} = .5$ and $\alpha_{iB} = .5$, giving inaccurate corresponding slate frequencies. The problem here is that sometimes both advertisements will be throttled. If we use the slate to extrapolate which advertisers are throttled, as opposed to which advertisers are displayed, then we can avoid the pitfall presented in this example. Using $\hat{L}_i^k$ in place of L_i^k, we have:

$$\alpha_{ij} = 1 - \frac{\sum_{k: j \in \hat{L}_i^k} x_{ik}}{v_i}$$

This new definition sets $\alpha_{iA} = .5$ and $\alpha_{iB} = 1$. The resulting slate frequencies now exactly replicate the *LP* slate frequencies.

Minimizing Distance to Slate Frequencies- Another alternative heuristic is to find throttle rates that minimize some objective function that measures the distance to the *LP* slate frequencies. We assume the θ_{ik} are given (using the *LP*), and solve for the α_j. Two possibilities include:

Least Squares

$$min \sum_{ik} \left(\prod_{j \in L_i^k} (1 - \alpha_j) \prod_{j \notin L_i^k} \alpha_j - \theta_{ik} \right)^2$$

K-L Divergence

$$min \sum_{ik} \theta_{ik} \log \frac{\theta_{ik}}{\prod_{j \in L_i^k} (1 - \alpha_j) \prod_{j \notin \hat{L}_i^k} \alpha_j}$$

These nonlinear optimization problems can be solved using suitable nonlinear programming techniques. Since the optimization problems across queries are decoupled, their solution is not very expensive.

7. Simulation Results

In order to test our approach, we measure its performance in simulations. This section describes our results, as well as a number of considerations which must be handled by the simulation[4].

7.1. Query Selection

Since there are tens of millions, perhaps hundreds of millions, of queries, we are necessarily limited to working with a subset of them. Clearly, we wish to deal with a manageable subset which captures a large part of the benefits we hope to gain from our optimization algorithm. This is aided by the typical distribution of query volumes, where the head queries capture a disproportionate share of the revenue from sponsored search. In confirmation of this, we found that in one sample, the top 5000 queries (in terms of revenue) captured a significant fraction of the overall revenue. This indicates that even a modest gain for the head queries can lead to a significant overall gain.

This "cherry picking" can only be achieved at some cost. While it is easy to segment the queries into the head and the rest, it is not so obvious how to segment the bidders. We must somehow isolate the bidders associated with these chosen queries. Unbudgeted bidders present no problem, but we must take into account the fact that some budgeted bidders may have bid on queries in both the chosen head set, and the remainder. We must therefore partition the budgets of these bidders into two parts—that spent on our chosen set of queries, and the rest. As a practical matter, this is not too difficult; we may base the division on historical data. However, this obviously introduces a measure of uncertainty that we would wish to minimize.

Fortunately, this sort of problem has been considered before. Carrasco et al. [2003] consider the problem of clustering a query-bidder bipartite graph corresponding to a sponsored search market. Even more appropriate, [Anderson 2007] finds clusters that are isolated (little spend leaves the cluster) *and* rich (contains a large amount of spend). We may consider this a generalization of our present problem of isolating a lucrative head market. Using a

[4] The results we report here are based on simulation only and the algorithm presented is not in operation on Yahoo!'s production system.

variant of the algorithms in [Anderson 2007] we may hope to choose a set of head queries minimally connected to the remainder. Having done so, we may then compute adjusted budgets for the budgeted bidders which straddle the chosen/non-chosen query set. While this a desirable property, it is not essential to our approach.

7.2. Column Generation Implementation

Our prototype system is implemented in C++ to run under Linux (or Cygwin) on an Intel-based work station. We use the open source COIN-OR library [Lougee-Heimer 2001] and its *LP* code Clp [COIN-OR], which allows efficient implementation of the column-generation framework and fast updating of the model without the need for any external interface. The column generation code itself—both the initial enumeration and subsequent dynamic programming subproblem solution—are also implemented in C++. This is economical and avoids compatibility problems, while at the same time allowing us, through the COIN-OR interface, to easily use a commercial *LP* code if we should ever find Clp inadequate for our models. So far this has been far from the case. The models on which we carry out most of our experiments, using real data on approximately 5,000 queries and about 50,000 bidders of whom about 60% are budgeted, are solved to optimality in less than half a minute on a 32-bit Linux box with a 2.8 GHz Xeon processor, and in less than 1 minute, even when doubled in size. We therefore expect the algorithm to scale well, even if we re-solve at short intervals of, say, 15 minutes.

Note that the algorithm naturally produces results which are suitable directly for serving with organic query results, a task normally performed by the *ad server*. The ordered slates of ads, and their serving frequencies in response to queries, can be used to relieve the ad server of the need to execute the auction process for our chosen set of queries. Since we choose from the head queries, this can lead to a significant reduction in workload at the ad server itself.

7.3. Simulation Methodology

We measure the algorithm's performance against a greedy "baseline" algorithm, which allocates to a query all bidders who have any remaining budget.

For this evaluation we use a fixed set of 5000 queries, and capture hourly (over an eleven-day period) the bidders, bids and budgets for each of the queries in this set. We use predictors of the click-through rates that use historical click data for their prediction (the exact details of these predictors is beyond the scope of this paper).

We used an impression predictor to predict v_{it}, the number of times query i will appear at hour t of the day. To compensate for the dynamics in impression volumes, we adapt our algorithm as follows: we convert the variables x_{ik} to frequencies $f_{ik} = \frac{x_{ik}}{v_i}$. Then, for each query instance we use a coin-toss, weighted by the frequencies f_{ik}, to decide which slate to show for this query.

For each algorithm we evaluate, we perform the following steps every hour:

1. Simulate the keyword auction mechanism for all queries each hour in arbitrary order (the order will not matter) according to the algorithm chosen. For the greedy algorithm the auction is performed with all candidates that have any budget remaining, while for the *LP* based algorithms, a coin is tossed to decide which slate to show based on the frequencies f_{ik}.
2. Compute metrics such as revenue, efficiency, clicks and PPC.
3. Check if any bidder has exceeded their adjusted daily budget. If so, we reimburse those bidders the amount they are owed and remove them from participating in future auctions.

There are several inaccuracies in the simulation. First, due to the hourly granularity, bidders who exceed their budget in the middle of a given hour may cause illegitimate price hikes for other bidders. Second, we assume at each showing of the slate that we receive exactly the expected number of clicks on each advertisement. In reality, there will be some amount of variance and inevitable inaccuracies in the click-through estimates. However, we believe that the above inaccuracies are not significant to our results; furthermore, the inaccuracies will have similar affects on both of the algorithms, so in terms of measuring relative performance these inaccuracies have little impact.

7.4. Results

Our simulation results are quite promising. Whether we optimize for revenue or efficiency, in both cases our results show a significant increase in both values. As we would anticipate, we see better performance in efficiency when that is the objective function, and similarly for revenue. However, the revenue appears to be more volatile in response to the objective function, with a percent increase that roughly doubles, whereas the percent increase for efficiency sees roughly a 30% increase. It is also interesting to note that the gains in revenue and efficiency using slates, for each day of the simulation, follow similar patterns, peaking and dipping on the same days (as seen by the lines labeled *revenue/slates* and *revenue/throttling* in *Figures 4 and 5*). This suggests that the ability to maximize in either case is based on similar properties of the problem. We see that revenue and efficiency are closely tied together and that gains along one objective imply similar gains along the other.

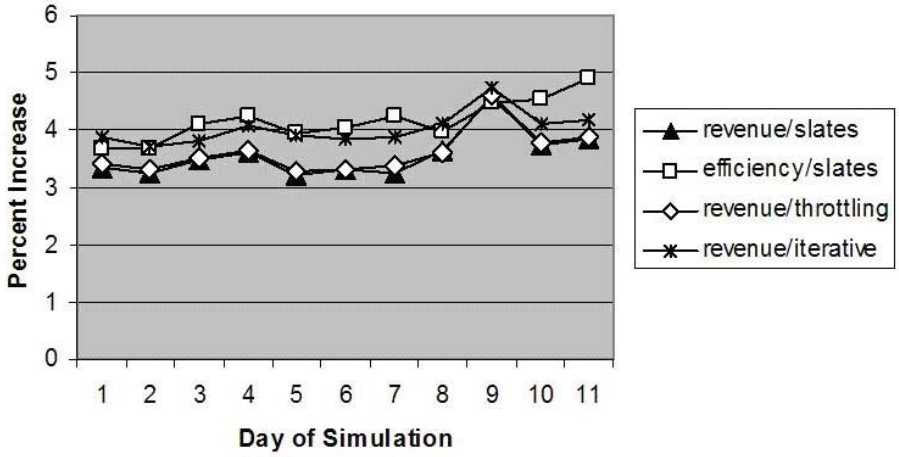

Figure 4: Gains when efficiency is maximized

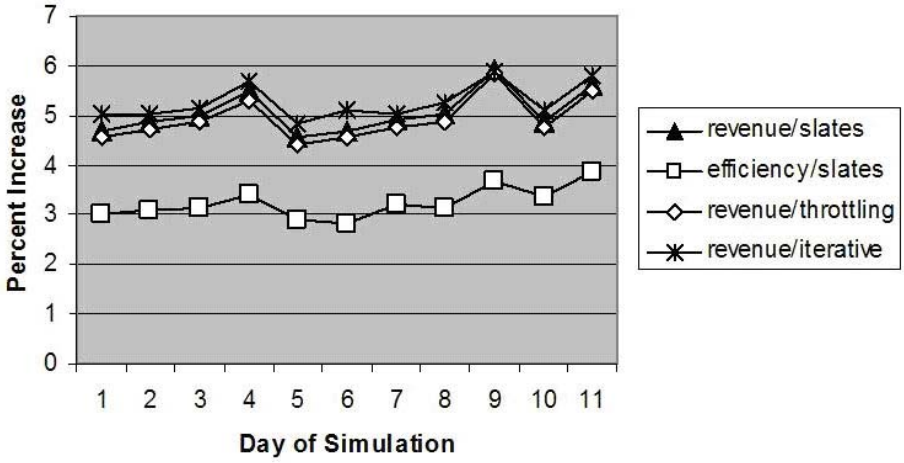

Figure 5: Gains when revenue is maximized

The percent increase in revenue using throttle rates as described in Section 2 is shown by the line labeled *revenue/throttling* in *Figures 4 and 5*. We observe that the percent increase in revenue is almost exactly the same when using throttle rates instead of slates. Interestingly, throttle rates perform slightly worse (in terms of revenue) when revenue is maximized but slightly better when efficiency is maximized. This suggests there is a small gain based on slates that are carefully and precisely constructed for the objective function. In the case of maximizing efficiency, this precision in attaining a small increase in efficiency is to the detriment of revenue.

The iterative approach shows increased revenue whether maximizing for efficiency or revenue. However, the line labeled *revenue/iterative* in *Figures 4 and 5* shows that the increase is significantly more when efficiency is maximized.

Figure 6 shows that the impact on advertisers is favorable, but differs between budgeted and unbudgeted advertisers. We see here an interesting distinction: budgeted bidders get a steep increase in clicks with low prices, whereas the unbudgeted advertisers receive little increase in overall clicks and a slightly higher PPC. The value also increases slightly, although not graphed here. This does not mean that if unbudgeted bidders specify budgets that will lead to much increased clicks and lowered prices for them too. It is just that the tightness of the budgets associated with the budgeted bidders allows a much larger scope for our approach to improve their values over simple strategies for tackling budget specifications. Overall, the impact on bidders appears positive, and is therefore more likely to be sustainable.

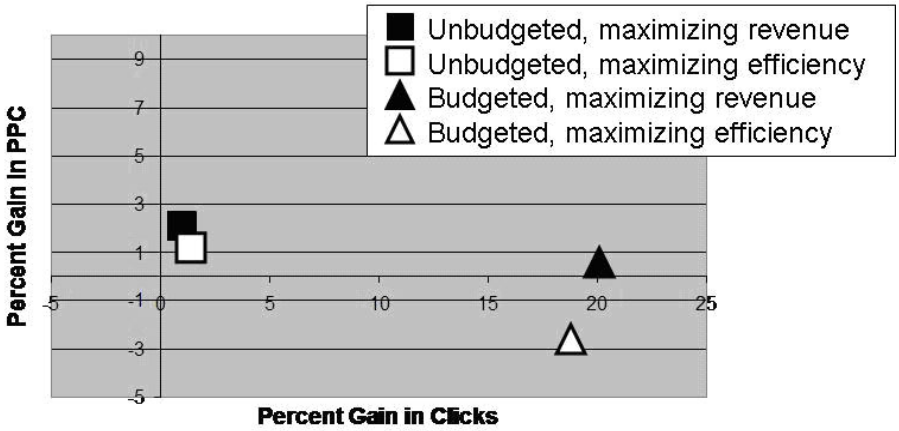

Figure 6: Impact of the optimization on bidders

8. Conclusion

Our simulations show that (irrespective of the objective function used) the better scheduling resulting from our approach leads to improved aggregate values for the advertisers. Hence their overall response is expected to be positive. Still, it is important to understand how individual advertisers might react to our approach, and specifically how they will change their bids and budgets in response. While the constraints of section 1 can ensure guaranteed values for each individual advertiser, it is not clear if enforcing these constraints would seriously inhibit the *LP*'s ability to increase the objective function. We cannot simply address advertiser incentives by using a truthful pricing scheme, such as VCG as described in Section 3, because optimizing over separate truthful auctions does not maintain truthfulness of the overall system. An interesting area for future work is to maintain significant increases in revenue (or efficiency) while accounting for individual advertiser incentives.

Another parameter that lies outside our control are query frequencies. We focus on queries at the head of the query distribution that can be forecasted with a high degree of accuracy. However, there is also a heavy tail in this distribution, and many of these queries are not as easily forecasted as our subset. As described in section 1, when query frequencies are unknown, previous research has proposed online algorithms with provable worst case performance guarantees. This previous research does not incorporate pricing and ranking into their solution. There may be a way to extend the work in [Mahdian 2007] to use our *LP* as a subroutine. Another approach that does not focus on worst case analysis, would be to explore methods such as the one presented in Section 2, to compensate for variation in query traffic. It is not yet well understood whether such methods might be able to accommodate changing bids and budgets, as well as variable query frequencies. We are also exploring other approaches when query frequencies are unknown, including machine learning and stochastic programming.

Concerning the implementation challenges described in Section 6, it may be possible to find optimal throttle rates directly. We do not know of an optimization program for throttle rates that is tractable, and leave this as an open problem.

We may also consider using parallel processing to scale up the approach even further. In particular, instead of dividing queries into our chosen set and the rest, we may use an algorithm such as that proposed in [Carrasco 2003] to partition the query-bidder graph into multiple submarkets, and apply the budget adjustment method discussed in Section 1 to partition the affected budgets and allow parallel solution.

Acknowledgment

This research was funded by Yahoo!, Inc. The authors would like to thank Andrei Broder, Arik Motskin, Kevin Lang, Ananth Nagarajan, and Jan Pedersen for helpful discussions and advice related to this research.

REFERENCES

Abrams, Z., "Revenue Maximization When Bidders Have Budgets," *Proc. Symposium on Discrete Algorithms*, pp. 1074-1082, 2006.

Aggarwal, G., A.Goel, and R.Motwani, "Truthful Auctions For Pricing Search Keywords," *Proc. 7th ACM conference on Electronic Commerce*, pp.1-7, 2006.

Alczak, S., D. Gregg, and J. Berrenberg, "Market Decision Making for Online Auction Sellers: Profit Maximization or Socialization," *Journal of Electronic Commerce Research*, 7, 2006.

Anderson, R. and K. J. Lang, "Finding Rich Isolated Submarkets in a Sponsored Search Spending Graph," *Unpublished Manuscript*, 2007.

Borgs, C., J.Chayes, N.Immorlica, M.Mahdian, and A. Saberi, "Multi-unit Auctions with Budget-constrained Bidders," *Proc. 6th ACM Conference on Electronic Commerce*, pp. 44-51, 2005.

Carrasco, J. J., D.Fain, K.Lang, and L. Zhukov, "Clustering of Bipartite Advertiser-keyword Graphs," *Workshop on Large Scale Clustering at IEEE International Conference on Data Mining*, 2003.

Chen, M., A.Chen, and B. Shao, "The Implications and Impacts of Web Services to Electronic Commerce Research and Practices," *Journal of Electronic Commerce Research*, 3, 2003.

Clarke, E., "Multipart Pricing of Public Goods," *Public Choice*, 11:17-33, 1971.

Dantzig, G. B., *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

de Farias, I. and G. Nemhauser, "A Polyhedral Study of the Cardinality Constrained Knapsack Problem," *Mathematical Programming (Ser. A)*, 96:439-467, 2003.

Dietrich, B. and J. J.Forrest, "A Column Generation Approach for Combinatorial Auctions," *Workshop on Mathematics of the Internet: E-Auction and Markets Institute for Mathematics and its Applications*, 2001.

Edelman, B., M.Ostrovsky, and M. Schwarz, "Internet Advertising and the Generalizaed Second Price Auction: Selling Billions of Dollars Worth of Keywords," *Second Workshop on Sponsored Search Auctions, Ann Arbor, MI. June*, 2006.

Groves, T., "Incentives in Teams," *Econometrica*, 41:617-631, 1973.

Kim, C. and R. D.Galliers, "Deriving a Diffusion Framework and Research Agenda for Web-based Shopping Systems," *Journal of Electronic Commerce Research*, 5, 2004.

Lougee-Heimer, R., F. Barahona, B. L.Dietrich, J. Fasano, J. J. Forrest, R. Harder, L.Ladanyi, T. Pfender, T. Ralphs , M. Saltzman, and K. Scheinberg, "The COIN-OR Initiative: Accelerating Operations Research Progress through Open-source Software," *ORMS Today*, Vol.28, No.5, 2001.

Lubbecke, M. E. and J.Desrosiers, "Selected Topics in Column Generation," *Operations Research*, Vol.53, No.6:1006–1027, 2005.

Mahdian, M., H.Nazerzadeh, and A. Saberi, "Allocating Online Advertisement Space with Unreliable Estimates," *ACM Conference on Electronic Commerce*, 2007.

COIN-OR Foundation:http://www.coin-or.org.

Sathiya Keerthi, S. and J. A. Tomlin, "Constructing an Optimal Slate of Advertisements," *Yahoo! Research Report*, 2006.

Schrage, L., "Solving Multi-object Auctions with LP/IP," *University of Chicago, Unpublished Manuscript*, 2001.

Silverstein, C., H.Marais, M. Henzinger, and M. Moricz, "Analysis of a Very Large Web Search Engine Query Log," *SIGIR Forum*, Vol.33, No.1:6–12, 1999.

Vazirani, U., A.Mehta, A.Saberi, and V.Vazirani, "Adwords and the Generalized Bipartite Matching Problem," *Proceedings of the Symposium on the Foundations of Computer Science*, pp.264-273, 2005.

Vickrey, W., "Counterspeculation, Auctions, and Competitive Sealed Tenders," *Journal of Finance*, 16:8-37, 1961.