An Algorithm for Stochastic Multiple-Choice Knapsack Problem and Keywords Bidding

Yunhong Zhou
HP Labs
1501 Page Mill Rd
Palo Alto, CA 94304
yunhong.zhou@hp.com

Victor Naroditskiy *
Department of Computer Science
Brown University
Providence, RI 02912
victor@brown.edu

ABSTRACT

We model budget-constrained keyword bidding in sponsored search auctions as a *stochastic multiple-choice knapsack problem* (S-MCKP) and propose a new algorithm to solve S-MCKP and the corresponding bidding optimization problem. Our algorithm selects items online based on a threshold function which can be built/updated using historical data. Our algorithm achieved about 99% performance compared to the offline optimum when applied to a real bidding dataset. With synthetic dataset and *iid* item sets, its performance ratio against the offline optimum converges to one empirically with increasing number of periods.

1. INTRODUCTION

Sponsored search is an effective way of monetizing search activities where advertisers pay to place their ads on search results pages for specific user keyword queries. Using an automated auction mechanism, search engine companies alleviate themselves from the burden of pricing and placing ads and shift the burden to advertisers. On the advertisers side, large companies spend millions of dollars each year to bid on thousands of keywords, and it is important for them to automate and optimize the bidding process to achieve the best return on investment (ROI). In this work we focus on the bid optimization problem for an advertiser with budget constraints. Formally, we try to address the following problem: for each keyword and each time period, how much should the advertiser bid (i.e., which position to obtain), so as to maximize ROI of the ads given a fixed budget and a fixed time horizon?

For a given keyword, there are multiple slots in the search results page that the auctioneer needs to allocate to different ads from different advertisers taking into account the advertisers' bidding price, ad quality, and other factors. There are different ad ranking and pricing schemes most of which are variations of *rank-by-price* and *pay-per-click* [11, 24, 16], where the advertiser in the i-th position pays the bid of the $(i+1)$-th advertiser whenever its ad is clicked by a user. No matter what ranking and pricing scheme the auctioneer deploys, for a given keyword, an advertiser can bid appropriately to get its ad placed in any ad position. For each ad slot, the advertiser incurs a *cost* (the fee that the auctioneer charges for each user click), obtains a *revenue* (the expected value-per-click), and a *profit* (the difference between revenue and cost). Naturally, we can model each ad position as an item with associated weight (cost) and value (either revenue or profit). The advertiser (or the agent acting on behalf of the advertiser) has a budget constraint, and it naturally corresponds to the knapsack capacity. Furthermore, one policy most auctioneers enforce is that each advertiser can have *at most one* ad appear on each keyword results page. This corresponds to that at most one item from each item set can be taken in the Multiple-Choice Knapsack Problem (MCKP), a well-known variation of the classic Knapsack Problem (KP). Therefore we can model the budget constrained bidding problem as a MCKP.

Compared to traditional offline setting of knapsack problems, keyword bidding is by nature online and stochastic. As any keyword auction is open to all advertisers with a positive budget, advertisers can join/leave the auction at any time and change their bids arbitrarily. Bidders often exhibit strategic bidding behaviors (e.g., overbidding [10], vindictive bidding[25]) which make the market more dynamic. Furthermore, sponsored search is driven by user queries/clicks. Even though the number of user queries/clicks is statistically stable over long time horizon, periodical spikes/drops are common and quite unpredictable. All these factors contribute to the online and stochastic nature of the underlying bidding problem and this motivates us to work on the stochastic and online version of MCKP.

1.1 Our Contributions

In this work we model the budget constrained bidding problem for keyword auctions as the online multiple-choice knapsack problem, design efficient algorithms for S-MCKP and translate it back to solve the budget-constrained bidding problem. Our algorithms are simple, easy to implement, and achieve a performance ratio consistently over 90% with both synthetic and real bidding data.

Even though the Online 0/1 Knapsack Problem is a well-studied problem in Operations Research and Online Algorithms, we are aware of no prior work on S-MCKP. By utilizing previous work on stochastic Online-KP, we design a simple algorithm for S-MCKP with performance ratio approaching one empirically while the number of time periods goes to infinity.

Our algorithms for keyword bidding as well as S-MCKP assume input item sets are independent and identically distributed (iid), however our algorithms do not require any knowledge of the distribution. Our algorithms are based on

*Work was done while the author was an intern at HP Labs.

WWW2008, April 21–25, 2008, Beijing, China.
.

maintaining a threshold function, and the threshold function can be built in advance using historical training dataset, or can be built from scratch and updated overtime during the execution of the algorithm. The machine learning capability improves the bidding performance and makes our algorithm more attractive to field deployment.

The rest of the paper is organized as follows. In Section 2 we briefly discuss related work. In Section 3, we introduce terminology related to online knapsack problems and describe Lueker's algorithm for the stochastic online knapsack problem. In Section 4, we describe our algorithm for S-MCKP and prove some properties of the algorithm. Section 5 is dedicated to modeling the bidding optimization problem as S-MCKP, and Section 6 is devoted to experimental evaluation of our algorithms for S-MCKP and the keyword bidding problem. We conclude in Section 7.

2. RELATED WORK

Keyword Auctions and Bidding Optimization. Over the past few years, keyword auctions have attracted a lot of attention both from the auctioneer's perspective ([11, 24, 3, 16]) and the advertiser's perspective ([10, 4, 23, 25]). For revenue maximization for the auctioneer with budget-constrained bidders, there are a few papers with various complexities to model keywords, slots, and clicks ([6, 1, 19, 7, 2]).

For bidding optimization for the advertiser, Kitts and LeBlanc [14] describe various bidding heuristics. Borgs et al. [5] propose a bidding strategy which over time equalizes the ROI over all keywords. Rusmevichientong and Williamson [22] discuss how to learn the CTRs for various keywords over time and select keywords accordingly. Most recently, Feldman et al. [12] studied variants of the bidding optimization problem where the objective is to maximize the number of clicks, with possibly complicated interactions among many keywords, and Cary et al. [8] analyzed properties of greedy bidding strategies.

Chakrabarty et al. [9] modeled the budget-constrained bidding optimization problem as Online-MCKP and designed competitive algorithms for Online-MCKP and evaluated the algorithms using both synthetic and real datasets. Their algorithms depend on some input parameters and thus a good bidding performance depends on either knowing these parameters or tuning them appropriately. They emphasize the worst-case performance guarantee while this work focuses on the average-case performance with stochastic input. In addition, the performance of their algorithm on the same real bidding data is between 90%-95%, however ours is around 99%.

Knapsack Problems and Online Algorithms. Variants of knapsack problems were studied extensively in Combinatorial Optimization and Operations Research. For a comprehensive exposition of this topic, see the textbook by Psinger et al. [13]. Online knapsack problems were first studied by Marchetti-Spaccamela and Vercellis [18] and showed that in the general case, there exists no online algorithm achieving any non-trivial competitive ratio. Many special cases of the problem have been studied afterwards, among them the stochastic online knapsack problem [17, 21, 15], and the online partially fractional knapsack problems [20].

Among all this work, Lueker [17] designed a simple threshold based online algorithm for the classic 0/1 knapsack problem, assuming that the weight and value of all items are iid. Under the iid assumption and perfect knowledge of the distribution, Lueker's algorithm achieves an *optimal* performance ratio of $1-O(\log\log n/\log n)$ against the offline optimum. Instead of assuming that the items are iid, Chakrabarty et al. [9] assumes that all items have their value/weight ratio upper bounded by U and lower bounded by L, for two positive constants U, L. Assuming further that all items are small compared to the knapsack capacity, they designed both deterministic and randomized threshold based algorithms for the online 0/1 knapsack problem achieving an (optimal) competitive ratio $\log(U/L)+1$. They also extend the algorithm to the multiple-choice setting and obtains a competitive ratio of $\log(U/L)+2$.

3. ONLIKE KNAPSACK PROBLEMS AND LUEKER'S ALGORITHM

In this section we introduce the Online Knapsack Problem (Online-KP) and describe Lueker's algorithm to solve stochastic Online-KP.

The 0/1 Knapsack Problem (KP) is as follows: given a set of items $\{(w_i, v_i) \mid 1 \le i \le n\}$ and a knapsack capacity C, select a subset of items to maximize the total value of selected items while the total weight is bounded by C. Throughout the paper, for each item i, we call w_i its *weight*, v_i its *value*, and the ratio between value and weight its *efficiency* ($e_i = v_i/w_i$). The Online Knapsack Problem (Online-KP) is the same as the 0/1 KP except that items arrive online one at a time. At each time period t, item t arrives, and the algorithm has to decide whether to select item t or not. The Stochastic Online-KP is the same as Online-KP with an extra assumption that the (weight, value) pair of each item is randomly drawn from the same joint distribution. Naturally, we assume that the knapsack capacity is proportional to the number of items ($C = \Theta(n)$), and all items are small compared to the overall knapsack capacity ($w_t = O(1)$ and $v_t = O(1)$, $\forall\ t$).

Lueker's Algorithm [17] for the Stochastic Online-KP is based on a threshold function that is generated using the distribution of items. All items are assumed to be iid. Only items with efficiency at least the *threshold efficiency* are included in the solution. The algorithm for Online-KP is in Figure 1.

Algorithm ALG-Lueker-OKP
Input: items (w_t, v_t) for $t = 1, \ldots, n$;
knapsack capacity C; threshold function g
Output: items to take
1. for each item t from 1 to n
if $e_t \ge g(\frac{C}{n-t+1})$ and $w_t \le C$
take item t
$C := C - w_t$
2. return items taken

Figure 1: Lueker's Algorithm for Online-KP.

The Threshold Function. The main part of the algorithm is the threshold function g which maps the average remaining capacity per time period to an efficiency value, denoted *threshold efficiency*. The threshold efficiency is such that the expected weight of the remaining items with effi-

ciency at least the threshold efficiency is equal to the remaining capacity:

$$C = \mathrm{E}_{w_i,v_i}\left[\sum_{i=1}^{n} w_i \mathbf{1}_{\{\frac{v_i}{w_i}\geq e^*\}}\right] = \sum_{i=1}^{n}\mathrm{E}_{w_i,v_i}\left[w_i \mathbf{1}_{\{\frac{v_i}{w_i}\geq e^*\}}\right]$$

The second equality above uses the linearity of expectation. Since all items are iid, thus

$$C = \sum_{i=1}^{n}\mathrm{E}_{w_i,v_i}\left[w_i \mathbf{1}_{\{\frac{v_i}{w_i}\geq e^*\}}\right] = n\ \mathrm{E}_{w,v}\left[w \mathbf{1}_{\{\frac{v}{w}\geq e^*\}}\right]$$

$$\frac{C}{n} = \mathrm{E}_{w,v}\left[w \mathbf{1}_{\{\frac{v}{w}\geq e^*\}}\right]$$

Let

$$f(e) \equiv \mathrm{E}_{w,v}\left[w \mathbf{1}_{\{\frac{v}{w}\geq e\}}\right], \quad (1)$$

then the threshold function is $g = f^{-1}$, the inverse of f. f maps the efficiency e to the expected item weight among items with efficiency at least e, while g maps the average capacity per item to the efficiency.

4. APPROXIMATION ALGORITHMS FOR S-MCKP

In this section we describe our algorithm for S-MCKP. Before we introduce the algorithm, we first define the problem briefly. The Multiple-Choice Knapsack Problem is a generalization of the 0/1 KP: Given a collection of item sets $\{N_t \mid t = 1,\ldots,n\}$ where $N_t = \{(w_{ti}, v_{ti} \mid 1 \leq i \leq n_t\}$ for each t and a knapsack capacity C, select at most one item from each item set to maximize the total value of selected items while the total weight of selected items is bounded by C. The Online MCKP is the online version of MCKP where item set N_t arrives at time t and the algorithm needs to select at most one item from N_t. Stochastic MCKP is the same as Online MCKP with an extra assumption that all item sets are iid random variables. Naturally we assume $C = \Theta(n)$, $w_{ti} = O(1), v_{ti} = O(1)\ \forall\ t, i$.

Our algorithm for the S-MCKP is based on Lueker's Algorithm for Stochastic Online-KP (described in Section 3) and an approximation for MCKP [13]. We first describe the approximation for MCKP (Section 4.1), then an approximation for the threshold function (Section 4.2), and finally the overall algorithm (Section 4.3).

4.1 Converting Item Sets to Incremental Items

Approximation for MCKP modifies the items from each item set so that taking multiple items is equivalent to taking one original item ([13], p.320). An item i is *dominated* by another item j if $w_j \leq w_i$ and $v_i < v_j$. An item i is *LP-dominated* by items j and k if i is dominated by a convex combination of j and k. Equivalently, if $w_j < w_i < w_k$ and $v_j < v_i < v_k$, then i is LP-dominated by j, k if

$$\frac{v_k - v_i}{w_k - w_i} \geq \frac{v_i - v_j}{w_i - w_j}.$$

The algorithm to remove all dominated and LP-dominated items and generate incremental items is described in Figure 2. The algorithm consists of two steps, first sorting items in increasing weight order, then removing dominated and LP-dominated items repeatedly. The second step clearly takes linear time, thus the total running time is dominated by the first step of sorting, thus $O(n \log n)$ time.

Algorithm ALG-Gen-Incr-Items
Input: an item set $N_t = \{(w_{ti}, v_{ti}) \mid i = 1,\ldots,n_t\}$
Output: incremental items
1. sort items according to increasing weights
2. /** remove dominated and LP-dominated items **/
   let Q be a queue with initially one element $(0,0)$
   for i from 1 to n_t
     push element i into the queue
     (ℓ always denote the last element of Q)
     if $w_\ell = w_{\ell-1}$
       remove from Q either item ℓ or $\ell-1$
       with smaller value
     while $\ell > 2$ and $\frac{v_{\ell-1}-v_{\ell-2}}{w_{\ell-1}-w_{\ell-2}} \leq \frac{v_\ell - v_{\ell-1}}{w_\ell - w_{\ell-1}}$
       remove item $\ell-1$ from Q
3. /** create incremental items from items in Q **/
   let $\{(w_i, v_i) \mid 1 \leq i \leq \ell\}$ denote the items in Q
   $\bar{w}_1 = w_1$, $\bar{v}_1 = v_1$
   $\bar{w}_i = w_i - w_{i-1}$, $\bar{v}_i = v_i - v_{i-1}$, $\forall\ 2 \leq i \leq \ell$
4. return $\{(\bar{w}_i, \bar{v}_i) \mid 1 \leq i \leq \ell\}$

Figure 2: Algorithm to generate incremental items from an item set

Once all dominated and LP-dominated items are removed, and remaining items are sorted in increasing weight order, then for three adjacent items $i-1, i, i+1$, we have

$$\frac{v_i - v_{i-1}}{w_i - w_{i-1}} = \frac{\bar{v}_i}{\bar{w}_i} = \bar{e}_i > \frac{v_{i+1} - v_i}{w_{i+1} - w_i} = \frac{\bar{v}_{i+1}}{\bar{w}_{i+1}} = \bar{e}_{i+1}.$$

Thus the efficiency of incremental items are monotone decreasing: $\bar{e}_1 > \bar{e}_2 > \ldots > \bar{e}_\ell$. Taking incremental items $1,\ldots,i$ from the set Q is equivalent to taking item i from the set N_t.

4.2 Approximating the threshold function

To compute an approximate solution for S-MCKP, we first convert each item set into a set of incremental items, and try to apply Lueker's Algorithm for Online-KP to these incremental items. Lueker's algorithm requires requires as an input the threshold function, which is not available to us. In this section we discuss how to compute an approximate threshold function using sample item sets, and how to update the threshold function over time.

Generating threshold function from a sample. Given a set of training item sets, we can transform them into a collection of incremental items. The distribution of incremental items may not be known or have a closed-form representation, however we can approximate it if we have a reasonably large sample size.

Given a sample set of m incremental items, we can approximate the threshold function given by Eq. 1 with the average over all the sample points. Formally, we can use $\tilde{f}$ to approximate f where

$$\tilde{f}(e) \equiv \frac{1}{m}\sum_{i=1}^{m} w_i \mathbf{1}_{\{e_i \geq e\}} \quad (2)$$

Assuming that the incremental items are sorted in decreasing order of efficiency, then $\forall\ e \in (e_{i+1}, e_i]$, $\tilde{f}(e)$ is equal to $\tilde{w}_i \equiv (w_1 + \ldots + w_i)/m$. Therefore $\tilde{f}$ is a piecewise constant function, and it can be represented as a sorted list of pairs

$\{(e_i, \tilde{w}_i) \mid 1 \le i \le m\}$ with $\{e_i\}$ monotone decreasing and $\{\tilde{w}_i\}$ monotone increasing. The threshold function can be computed using the algorithm in Figure 3.

Algorithm ALG-Gen-Threshold
Input: set of incremental items $\{(w_j, v_j) \mid j = 1, \ldots, m\}$
Output: threshold function f
1. sort items in decreasing order of efficiency.
   let $e_j = v_j/w_j, \forall\ j$, then $e_1 \ge e_2 \ge \ldots \ge e_m$.
2. $f(e_1) = \frac{w_1}{m}$
3. $f(e_i) = f(e_{i-1}) + \frac{w_i}{m}, \quad \forall\ 2 \le i \le m$
4. return f

Figure 3: Algorithm for generating the threshold function.

Update Threshold Function Online. We can update the threshold function as we are presented with new sets of incremental items. It is convenient to represent the threshold function by a collection of efficiencies $e_1 > e_2 > \ldots > e_k$ sorted in decreasing order and a collection of corresponding weights $w_1 < w_2 < \ldots < w_k$ in increasing order where $w_i = f(e_i)$. Initially the collections can be empty in which case the threshold function is generated using the first item set. See the algorithm in Figure 4.

Algorithm ALG-Update-Threshold
Input: threshold function $\tilde{f} = \{(e_i, w_i) \mid 1 \le i \le k\}$,
a set of incremental items $\{(\tilde{w}_j, \tilde{v}_j) \mid 1 \le j \le m\}$
Output: updated $\tilde{f}$
1. /** normalize weights **/
   $w_i = w_i \frac{k}{k+m} \qquad 1 \le i \le k$
   $\tilde{w}_j = \tilde{w}_j \frac{1}{k+m} \qquad 1 \le j \le m$
2. /** update weights **/
   $w_i = w_i + \sum_{\tilde{e}_j \ge e_i} \tilde{w}_j \qquad 1 \le i \le k$
3. /** create a list of sorted (e, w) pairs **/
   for j from 1 to m
   if there is no pair in f with efficiency $\tilde{e}_j = \tilde{v}_j/\tilde{w}_j$
   $i = \arg\max_i \{e_i \ge \tilde{e}_j\}$
   add $(\tilde{e}_j, w_i + \tilde{w}_j)$ to the list of new pairs
4. linearly merge the new list and $\tilde{f}$ to get the updated $\tilde{f}$

Figure 4: Algorithm for updating the threshold function.

4.3 An Approximation Algorithm for S-MCKP

We are now ready to describe our algorithm for S-MCKP. For each item set arriving online, we use ALG-Gen-Incr-Items given in Figure 2 to generate incremental items for the item set and use the approximate threshold function to select incremental items for the current time period. Since we described how to generate the approximate threshold function and update it in Section 4.2, we are now ready to describe the whole algorithm.

The algorithm for S-MCKP is in Figure 5. It consists of two phases, where the first is optional, and it depends on whether training item sets are available. For the second phase, the algorithm decides whether or not to take an item at time period t using the threshold function, and updates the threshold function if necessary.

Algorithm ALG-S-MCKP
Input: item set N_t for $t = 1, \ldots, n$;
knapsack capacity C;
(optional) training item sets
Output: items to take
1. (optional) /** generate threshold function f from training items sets **/
   create incremental items from training item sets using ALG-Gen-Incr-Items
   r is the average number of incremental items per set
   generate f using ALG-Gen-Threshold with these incremental items as input
2. for t from 1 to n
   create incremental items from item set N_t usin ALG-Gen-Incr-Items
   (optional step)
   update f (using ALG-Update-Threshold) and r
   $e = f^{-1}(\frac{C}{r(n-t+1)})$
   /** $r(n-t+1)$ is the expected number of remaining incremental items **/
   select incremental items with efficiency at least e
   $\bar{w}$, $\bar{v}$ are the total weight and value of selected incremental items
   if $\bar{w} \le C$
   take item $(\bar{w}, \bar{v})$.
   $C := C - \bar{w}$

Figure 5: Algorithm for S-MCKP.

5. KEYWORD BIDDING AS S-MCKP

Sponsored search auctions are used by search engine companies to sell ad positions to advertisers on search results page, where popular query terms are treated as "keywords". An auction is set up for each keyword where advertisers submit bids and compete for different ad positions. The auction mechanism determines how to rank and price ads, using factors like the bidding prices and ad qualities, or even budgets of different advertisers. Among many variations of ad ranking and pricing schemes, most are based on *rank-by-price* and *pay-per-click*. In this mechanism, assuming that bidding prices are sorted in decreasing order ($b_1 \ge b_2 \ge \ldots \ge b_n$), bidder i obtains position i, and is charged a fee $p_i = b_{i+1}$ whenever a user clicks on its ad. [1] No matter what ranking and pricing scheme the auctioneer deploys, for a fixed avertiser and a fixed keyword, the advertiser can obtain any position with an appropriate bidding price. For each ad slot, the advertiser incurs a *cost* (the fee that the auctioneer charges for each user click), obtains a *revenue* (the expected value-per-click), and a *profit* (the difference between revenue and cost). Naturally, we can model each ad position as an item with associated weight (cost) and value (either revenue

[1]For the popular rank-by-revenue scheme, the charge of the i-th position is p_{i+1} times a coefficient which is related to the quality scores of ads at both i and $i+1$. We can easily incorporate this case into our consideration. For simplicity, we assume all ads are equally good for this work.

or profit). Without loss of generality, we focus on profit.

A typical advertiser has a budget for some time horizon (e.g. daily, weekly, quaterly or annually) and wants to purchase a certain set of keywords to maximize its total ROI. The profit of the advertiser is equal to the total amount of expected revenue from search marketing minus the total amount of marketing cost. We can discretize the time horizon into small time periods and assume that the bidding prices of all advertisers do not change over each small time period. Formally, we can model the bidding optimization problem as a multiple-choice knapsack problem as follows. Given multiple keywords $k \in K$, multiple time periods when the advertiser places bids $t \in \{1, \ldots, T\}$, and multiple positions $s \in \{1, \ldots, S\}$, the item set N_t^k consists of items (w_{ts}^k, v_{ts}^k) for all ad positions s. Formally w_{ts}^k and v_{ts}^k are defined as follows:

$$\begin{aligned} w_{ts}^k &= p_{ts}^k \alpha^k(s) X^k(t), \\ v_{ts}^k &= (V^k - p_{ts}^k)\alpha^k(s) X^k(t), \quad \forall\ s, t, k. \end{aligned} \tag{3}$$

Here V^k denote the expected value-per-click for keyword k, $X^k(t)$ denote the number of user queries for keyword k at time period t, and $\alpha^k(s)$ denote the click-through rate (CTR) of position s (the ratio between total user clicks on the ad at s-th slot and the total number of impressions). $p_{ts}^k = b_{t,s+1}^k$, i.e. the cost-per-click is equal to the next highest bid. Since most auctioneers enforce a policy that each advertiser can have *at most one* ad appear on each keyword results page, this corresponds to that at most one item can be taken from N_t^k. If we treat each N_t^k as an item set, then this consists of an instance of MCKP where the knapsack capacity C is equal to the advertiser's total budget B.

6. EXPERIMENTAL RESULTS

We run two sets of experiments. The first set evaluates the performance of the algorithm ALG-S-MCKP on synthetic datasets when items are generated from various probability distributions. The second set of experiments uses a real dataset we manually collected from the (now defunct) Yahoo!/Overture view bids webpage.

6.1 Experiments With Known Distributions

In this set of experiments, we generate items with weights and values drawn independently from one of the following distributions: Uniform with support between 1 and 10, Normal with mean 10, and Exponential with mean 10. The normal distribution was truncated at zero to avoid negative weights and values. The number of items per set received each time period is 5. We express the budget (capacity) as a fraction of the mean weight of an item times the total number of time periods, i.e., $C = \lambda \times n \times \text{mean}(w)$, where n is the total number of time periods, mean(w) is the mean of the weight of a random item. For example, mean(w) is 5.5 when item weights are uniformly distributed between 1 and 10, $B = 5.5 \times \lambda \times n$. The value of λ indicates whether the budget is large compared to the expected overall spending if you pick an item randomly from each set. We tested the algorithms on various problem instances with various budget levels and report the following λ values in our experiments: 0.05, 0.2, 0.5, 0.9, 1.1.

We evaluate the performance of the algorithm based on the ratio of the value obtained by the algorithm and an upper bound on the optimal value of the solution to MCKP. The upper bound of the MCKP can be calculated by solving the fractional version of the MCKP as described in section 4.1.

We test two versions of the algorithm. The first one uses the threshold function generated based on 80 sampled item sets and does not update the threshold function. The other version does not generate the threshold function beforehand but uses the items received each time period to generate/update the threshold function. We refer to the first version as offline-training and to the second one as online-training. The results are in Figure 6. Each data point on the graphs is the average of 100 runs on random problem instances.

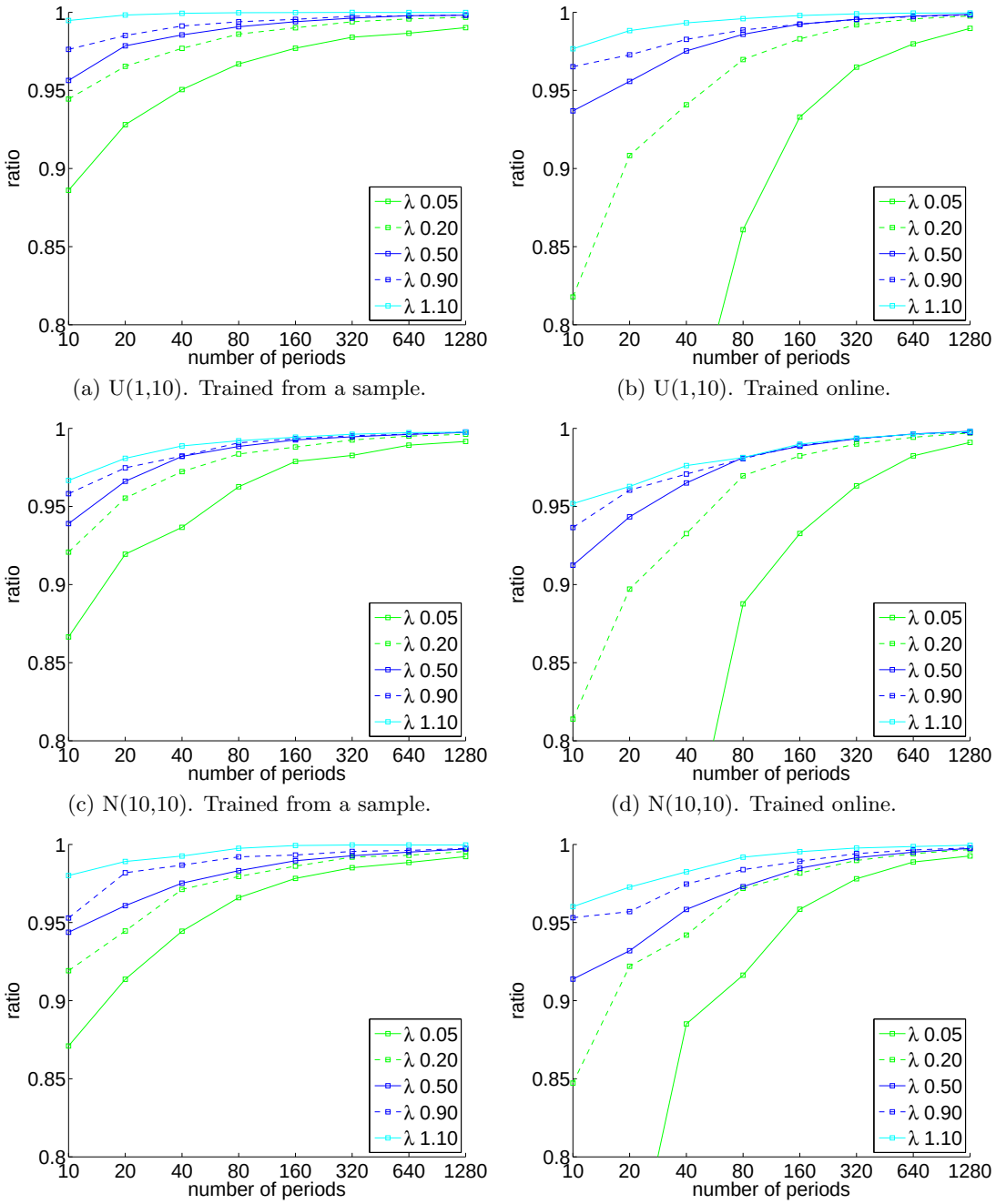
Graphs for the algorithm with offline-training are on the left and graphs for the algorithm with online-training are on the right. For both versions of the algorithm, the performance is almost always within 10% of the optimal when the number of periods is 20 or higher and approaches the optimal as the number of periods increases. The algorithm with online-training performs worse than the algorithm with offline-training when the number of periods is small. During the first few periods the threshold function of the online algorithm is very unreliable (based on very few samples) and the decisions made during early periods are prone to mistakes. These mistakes are especially costly when the budget is small because taking a wrong item may exhaust a significant part of the budget. However as the number of periods increases the performance of the online-training algorithm comes close to that of offline-training. The results are similar across different distributions.

6.2 Experiments With Real Bidding Data

For these experiments we use a keyword bidding dataset that was manually collected by [9] from the Overture view bids website over the period of two weeks. The bids are for the single keyword "auto insurance." There are totally 1842 distinct time periods. For each time period, the data contains the bidding prices for all top-40 positions. Each time period is about one minute. The data do not contain any information about the number of clicks.

For one experiment, following [9], we assume that $\alpha(s) = 1 - s/40$ is the CTR of position s and $X(t) = 1$ over all time periods. We use the algorithm that trains online without any pre-training. Each time period, the current set of bids is converted into the set of items as described in Eq. (3), for three distinct values $V = 8, 10, 12$. The algorithm's solution is within 99% of the offline optimum for all three values of V. Note that the algorithm performs better than the one described in [9], which achieves a ratio between 90%-95%. Even though both algorithms are based on using some kind of efficiency threshold for item selection, there is an important distinction: our algorithm's bids remain relatively constant over time close to the value which is optimal in expectation, while the algorithm from [9] bids higher at the beginning and gradually reduces its bid, and at some time it starts to increase its bids again based on a sniping heuristic.

For another experiment, we use an exponential function to model the CTR each periods: $\alpha(s) = 0.9^s$, while the number of clicks is $X(t) = 1$ as in the first experiment. The expected value-per-click in this experiment is set to the single value $V = 12$. The performance of the algorithm is shown in Figure 7. The algorithm finds a solution that is within 6% of the optimal for all budget levels and number



(a) U(1,10). Trained from a sample.

(b) U(1,10). Trained online.

(c) N(10,10). Trained from a sample.

(d) N(10,10). Trained online.

(e) E(mean=10). Trained from a sample.

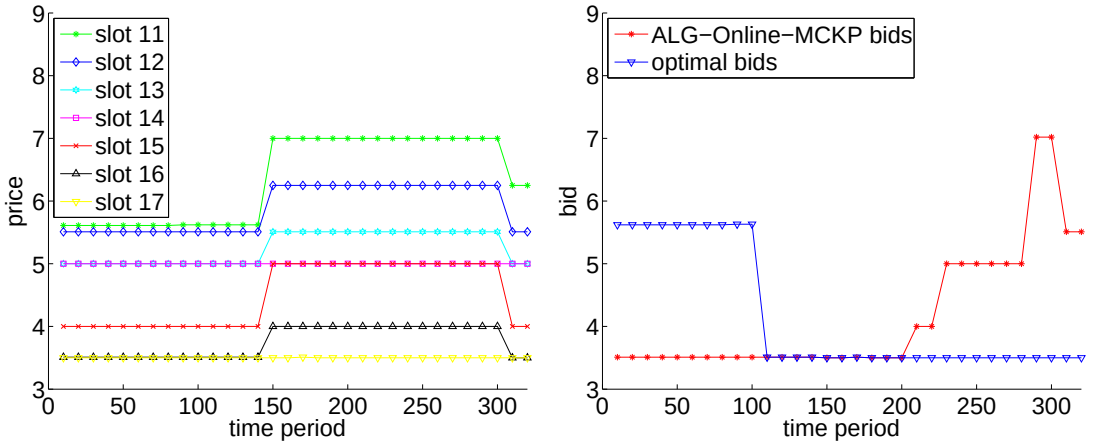
(f) E(mean=10). Trained online.

Figure 6: Performance of ALG-S-MCKP with various item distributions

of periods. Unlike the experiments with known distribution, the ratio is not monotonically increasing with the number of periods. Increased number of periods does not result in improved performance because the prices in the data set are not identically distributed each time period.

For instance, the ratio for $\lambda = 1.1$ drops from .97 to .95 when the number of periods increases from 160 to 320. We plot prices of the slots that are targeted by the algorithm during the first 320 periods to explain why this happens. Figure 8(a) shows that prices for slots 11-17 are constant during the first 140 periods. Around period 140, prices for slots 11-16 increase and stay at a higher level for most of the periods through 320. Intuitively, it is optimal to target a higher slot in order to get more clicks before period 140, i.e., when the prices are lower. We plot optimal bids and algorithm's bids in Figure 8(b) to illustrate this.

As expected, most optimal bids are higher during the first 140 periods in order to buy a lot of clicks when they are cheap. Optimal bids win slot 11 for \$5.7 for the first 100 periods and are lowered to \$3.5 to win slot 17 in periods



(a) Prices for Slots 11-17

(b) Bids for $\lambda = 1.1$ and $V = 12$

Figure 8: Keyword bidding data set with 320 periods and exponential CTR.

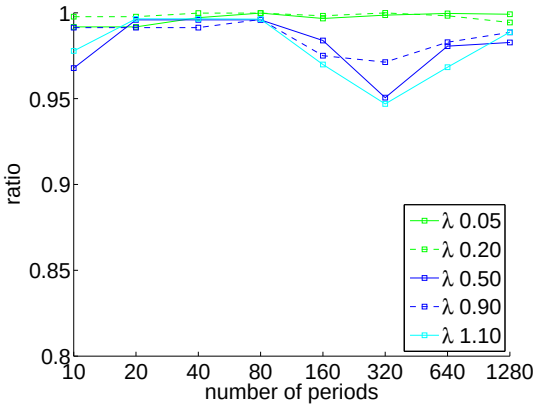


Figure 7: Performance of ALG-S-MCKP on keyword bidding data set with 1842 periods, exponential CTR, and fixed number of clicks $X(t) = 1$.

100-300. In contrast, the algorithm has no way of knowing that the prices are going to increase, and its bid ($3.5 to win slot 16) during the first 200 periods is at the level that is optimal assuming the prices will remain the same as in the first 140 periods. After period 140, the algorithm starts seeing higher prices and spending less as its bid of $3.5 results in position 17 instead 16 (i.e., fewer clicks). At period 200, the algorithm has seen enough periods with higher prices and has underspent enough to decide to raise its bids. Unfortunately, clicks are more expensive then and the algorithm performs worse than the optimal. However the algorithm's performance is still around 95% of the optimal.

The final experiment is a variant of experiment 2 where the number of clicks each time period is uniformly distributed between 1 and 20. The results are in Figure 9. The added randomness from the number of clicks makes the threshold function less reliable when it is based on few samples. This results in mistakes in early periods which are relatively more costly when the total number of periods is small. This is evidenced in a relatively bad performance when the number of periods is 10 and 20. However additional uncertainty about the number of clicks does not prevent the algorithm from perfoming well on problems with more periods. The algorithm is within 7% of the optimal when the number of period is 40 or more.

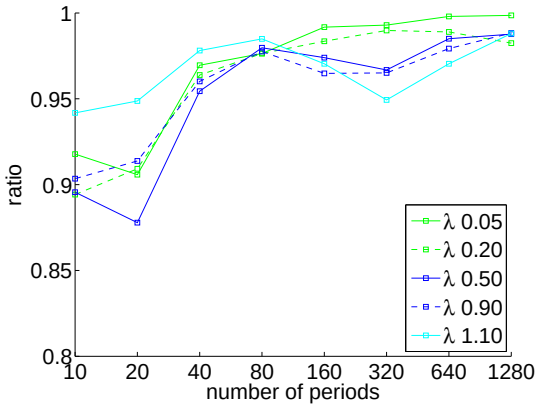


Figure 9: Performance of ALG-S-MCKP on keyword bidding data set with 1842 periods, exponential CTR, and random number of clicks.

7. CONCLUSION

We propose an algorithm for S-MCKP that combines an approximation to MCKP with an algorithm for Online-KP. Our algorithm is based on the idea that MCKP can be converted to KP, which can then be solved using the well-known greedy KP approximation, and the solution to KP can be mapped back to the solution to MCKP. Our main contribution is the algorithm that accomplishes this for the online version of MCKP. At the heart of the algorithm is the threshold function for KP which filters out the items of insufficient efficiency. We adapt the process of computing the threshold function to the online setting where no information about the items needs to be available a priori. Instead, the thresh-

old function is updated online. We apply the algorithm to problem instances generated with different distributions and to a real data set. In all of our experiments the performance is within 10% of the offline optimum, and it approaches the offline optimum when the number of periods is sufficiently large.

For future work, one direction is to model trends in data explicitly. The current algorithm assumes that there are no trends in data, as items are identically distributed across time periods. However it works relatively well in the presence of trends because the threshold function is updated over time. Another direction that we are currently pursuing function over time. Another direction that we are currently pursuing is building/deploying a keyword bidding agent based on the above algorithm. Yet another direction is to prove theoretical guarantees about the performance of the S-MCKP algorithm with general assumptions on the distribution of items.

8. REFERENCES

[1] Z. Abrams. Revenue maximization when bidders have budgets. In *SODA*, pages 1074–1082. ACM Press, 2006.

[2] Z. Abrams, O. Mendelevitch, and J. A. Tomlin. Optimal delivery of sponsored search advertisements subject to budget constraints. In *ACM EC*, pages 272–278, 2007.

[3] G. Aggarwal, A. Goel, and R. Motwani. Truthful auctions for pricing search keywords. In *Proc. ACM EC*, pages 1–7, June 2006.

[4] K. Asdemir. Dynamics of bidding in search engine auctions: An analytical investigation. In *Proc. 2nd Workshop on Sponsored Search Auctions*, 2006.

[5] C. Borgs, J. Chayes, O. Etesami, N. Immorlica, K. Jain, and M. Mahdian. Dynamics of bid optimization in online advertisement auctions. In *Proc. WWW*, pages 531–540, 2007.

[6] C. Borgs, J. T. Chayes, N. Immorlica, M. Mahdian, and A. Saberi. Multi-unit auctions with budget-constrained bidders. In *Proc. ACM EC*, pages 44–51, 2005.

[7] N. Buchbinder, K. Jain, and J. S. Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Proc. ESA*, pages 253–264, 2007.

[8] M. Cary, A. Das, B. Edelman, I. Giotis, K. Heimerl, and A. R. Karlin. Greedy bidding strategies for keyword auctions. In *ACM EC*, 2007.

[9] D. Chakrabarty, Y. Zhou, and R. Lukose. Budget-constrained bidding in keyword auctions and online knapsack problems. In *Proc. SSA*, May 2007.

[10] B. Edelman and M. Ostrovsky. Strategic bidder behavior in sponsored search auctions. In *Proc. 1st Workshop on Sponsored Search Auctions*, Vancouver, Canada, June 2005.

[11] 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*, 2007.

[12] J. Feldman, S. Muthukrishnan, M. Pal, and C. Stein. Budget optimization in search-based advertising auctions. In *ACM EC*, pages 40–49, 2007.

[13] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[14] B. Kitts and B. Leblanc. Optimal bidding on keyword auctions. *Electronic Markets*, 14(3):186–201, 2004.

[15] A. J. Kleywegt and J. D. Papastavrou. The dynamic and stochastic knapsack problem. *Operations Research*, 46(1):17–35, 1998.

[16] S. Lahaie. An analysis of alternative slot auction designs for sponsored search. In *Proc. 7th ACM Conference on Electronic Commerce*, pages 218–227, Ann Arbor, MI, June 2006.

[17] G. S. Lueker. Average-case analysis of off-line and on-line knapsack problems. *J. of Algorithms*, 29(2):277–305, 1998.

[18] A. Marchetti-Spaccamela and C. Vercellis. Stochastic on-line knapsack problems. *Mathematical Programming*, 68:73–104, 1995.

[19] A. Mehta, A. Saberi, U. V. Vazirani, and V. V. Vazirani. Adwords and generalized on-line matching. In *Proc. FOCS*, pages 264–273, 2005.

[20] J. Noga and V. Sarbua. An online partially fractional knapsack problem. In *Proc. 8th Sym. Parallel Architectures, Algorithms and Networks*, pages 108–112, 2005.

[21] J. D. Papastavrou, S. Rajagopalan, and A. J. Kleywegt. The dynamic and stochastic knapsack problem with deadlines. *Management Science*, 42(12):1706–1718, 1996.

[22] P. Rusmevichientong and D. P. Williamson. An adaptive algorithm for selecting profitable keywords for search-based advertising services. In *Proc. 7th ACM conference on Electronic commerce*, pages 260–269, 2006.

[23] B. K. Szymanski and J.-S. Lee. Impact of ROI on bidding and revenue in sponsored search advertisement auctions. In *Proc. 2nd Workshop on Sponsored Search Auctions*, Ann Arbor, MI, 2006.

[24] H. R. Varian. Position auctions. *International Journal of Industrial Organization*, 2007, to appear.

[25] Y. Zhou and R. Lukose. Vindictive bidding in keyword auctions. In *Proc. 9th Int. Conf. Electronic Commerce*, pages 141–146, Minneapolis, MN, August 2007.