Final Report

Cathy Tianjiao Zhang
Advisor: Prof. Amy Greenwald

Abstract

This is a draft of some sections of a bigger paper.

1 Introduction

In online advertising markets, search engines like Google and Yahoo! sell electronic billboards to advertisers. These billboards are spaces alongside the results of search queries where advertisers can post ads. In selling these ad spaces, search engines seek to maximize their profits; hence, they face the age-old question of how to price their goods.

Before the advent of the Internet, the most widespread advertising pricing model was "cost-per-million-impressions" (CPM).[1] This classic model, well-suited to more traditional media such as television, newspapers, and magazines, was adopted in 1994 by Internet content providers. Historically, Internet CPM contracts were large (some early contracts were valued at $30,000), and hence their negotiation was cumbersome because it necessitated extensive human intervention. This was unfortunate because there is ample opportunity for automation on the Internet.

In 1997, Overture (now Yahoo! Search Marketing) launched an innovative framework for selling advertising space on the Internet. There were (at least) two novel aspects of Overture's design: (i) payment is made on a per-click (PPC) rather than a per-impression basis, and (ii) rather than selling large chunks of advertising space, space alongside each query is sold in an *ad auction* tailored to that query. More specifically, among the three types of players in an ad auction—the search engine, the advertisers, and the users—here is how the dynamics proceed:

- the advertisers bid on the queries of interest to them (for example, Canon and Nikon might both bid on "digital camera")
- a user places a query of interest to him

[1]An *impression* is a posting of an ad to be viewed by a potential consumer.

- the search engine holds an auction in which the bidders are those advertisers who bid on that query
- the advertisers are ranked according to their bids (and possibly some other factors)
- the advertisers' ads appear alongside the search results in rank order
- the user observes the *organic* search results (i.e., those which are relevant to the query, but not ads), as well as the paid advertisements, and clicks on the links of interest to him
- the advertisers pay per click for any ads the user clicks on

In this dynamic environment, the search engines and the advertisers face difficult decision problems. At a minimum, the search engines must determine click prices, as well as where to display ads (i.e., how to rank the advertisers); the advertisers in turn must decide the maximum price they are willing to pay per click and what queries to bid on. In some settings, the search engine may also allow the the advertisers to submit a per-period budget: i.e., an upper bound on what they are willing to spend each period. In this case, the advertisers' decision includes this additional parameter. Furthermore, this complicates the search engine's picture, because it then must decide not just where to display ads, but *when* to display them as well.

In spite of the inherent complexities in ad auctions, Google AdWords, Yahoo! Search Marketing, and Microsoft adCenter are systems through which these search engines sell online advertising space via ad auctions. Indeed online advertising is the chief source of revenue for these industry giants. Nonetheless, the science of ad auctions is still not very well understood. Part of the reason for this is that the topic is highly interdisciplinary. It incorporates aspects of marketing, game theory, computer science, optimization, etc. Consequently, there is a growing interdisciplinary community of researchers studying the problems that arise in this domain, as evidenced by

2 TAC AA Overview

In the TAC AA game scenario[2], in which eight agents attempt to maximize their products over the course of D days on behalf of online advertisers in

[2]For details, visit `http://tac.eecs.umich.edu/about-the-game/`

a simulated sponsored search domain. The TAC AA scenario consists of 90,000 simulated users who submit queries, click on ads, and convert" (i.e., make purchases). Each of these users has an underlying manufac-turer and component preference which dictates which product the user will ultimately purchase. On a given day, a user may be non-searching, searching, or transacted, meaning they have made a purchase that satisfies their preferences. A searching user may be seeking information about a product or actually shopping for one. Only users that are explicitly shopping ever make purchases.

Every time a user makes a query, the publisher holds an auction that includes all agents under their budget to determine the positioning of ads on the screen. Although all eight agents may desire to have their ads shown, only five ad slots are available on the page. Slots that are higher up on the page are more likely to get clicked on by users (and hence, lead to conversions) than lower slots. The ad ranking mechanism for TAC AA varies among hybrids of the rank-by-bid and rank-by-revenue mechanisms, but is always revealed to the agents at the start of each game. After a user submits a query and is shown a page of ads by the publisher, the user's click behavior is based on a variant of the cascade model . The user considers the first ad, and clicks on it with some probability, after which there is some probability that the user will convert, meaning purchase a product from the advertiser. If the user converts, it moves to the transacted state and does not look at any other ads. If the user does not convert or did not click on the ad to begin with, there is some probability that the user will continue on to the next ad.

Each day, each agent sends the following information to the publisher: For each query, a bid representing the maximum price the advertiser is willing to pay per click for that query; an ad type describing the degree of targeting that will be performed in the ad; a per-query budget which specifies the maximum amount the advertiser is willing to spend on a given query for a single day; and an overall budget which restricts the amount that the advertiser can spend across all queries on a given day.

Each advertiser has a capacity, which is an upper bound on the number of sales that can be made without penalty over the last L days. As sales go beyond this amount, subsequent users have lower and lower conversion probabilities.

On day d, all advertisers receive reports summarizing the activity on day $d-1$. Advertisers do not receive detailed information about the bids of their opponents. But they are told, for each query: the average position of each agent, the ad type placed by each agent and the agent's own number of impressions, clicks, conversions, and average CPC.

3 Model-Light (i.e., Rule Based) Algorithms

3.1 ConstantPM, Slot, Click, ClickSlot

The various agents in this section is meant to provide some understanding of the environment and some nontrivial agents for testing.

3.1.1 ConstantPM

The first attempt is to design an agent that does not lose money in the game. Assume that the agent does not go over capacity. We can set our bid for each query so that

$$cpc_q = USP_q * pr(conv)_q * (1 - PM). \tag{1}$$

Here PM, or profit margin is a constant, representing the fixed percentage of revenue per click the agent earns as profit. We constrain $0 < PM_q < 1$, to ensure that the agent will never bid more than its true value for a query.

3.1.2 Slot

Slot attempts to stay in good slots in all auctions. We again set bids according to equation 1, but we have a different PM_q for each query.

SlotAgent adjusts PM_q, and consequently bid, by these rules:

1. **if** $slot_q <= GOOD\_SLOT$ **then** $PM_q * = INC\_PM$
2. **if** $!(slot_q < BAD\_SLOT)$ **then** $PM_q * = DEC\_PM$

Based on empirical observation, we set $GOOD\_SLOT = 3$, $BAD\_SLOT = 4$. We do not want the bids to vary too much in two consecutive days, since we want the agent to probe the auctions gradually. Therefore, we set $INC\_PM = 1.1$, $DEC\_PM = .9$.

3.1.3 Click

Click attempts to control the number of clicks on each query. It also uses equation 1 to determine bids. Since sales and clicks has the following relationship: $sales = clicks * pr(conv)$, targeting clicks is equivalent to targeting sales. We introduce another parameter, desired sales DS_q, representing the query sales our agent wishes to get from each query.

For simplicity, we fix DS_q to be the same for all queries and $\sum_q DS_q = dailyCapacity$. Hence, ClickAgent has the following rules:

1. **if** $sales_q > DS_q$ **then** $PM_q* = INC\_PM$
2. **if** $sales_q < DS_q$ **then** $PM_q* = DEC\_PM$

3.1.4 ClickSlot

ClickSlot combines the features of Slot and Click. It has the following four rules to allocate capacity among all queries and place a bid to obtain desired conversions for each query q:

1. **if** $sales_q > DS_q$ & $!(slot_q < BAD\_SLOT)$ **then** $DS_q* = INC\_DS$
2. **if** $sales_q < DS_q$ & $slot_q <= GOOD\_SLOT$ **then** $DS_q* = DEC\_DS$
3. Normalize DS_q such that $\sigma DS_q = dailyCapacity$
4. **if** $sales_q > DS_q$ & $slot_q <= GOOD\_SLOT$ **then** $PM* = INC\_PM$
5. **if** $sales_q > DS_q$ & $!(slot_q < BAD\_SLOT)$ **then** $PM* = DEC\_PM$

3.2 ClickProfitC, ClickProfitS

A direct improvement of ClickSlot is to observe that it tends to define a profitable query as the one that the agent got enough sales and was in a bad position. A natural alternative would be to look at profit directly. There are two possible ways to define profit: profit per sale (PPS), and profit per click (PPC) where

$$PPS_q = USP_q - \frac{CPC_q}{pr(conv)_q} \tag{2}$$

$$PPC_q = USP_q * pr(conv) - CPC_q \tag{3}$$

We implement ClickProfitC and ClickProfitS as two agents aware of PPC and PPS respectively. The rules for the ClickProfitC are:

1. **if** $PPC_q > avgPPC$ **then** $DS_q* = INC\_DS$
2. **if** $PPC_q < avgPPC$ **then** $DS_q* = DEC\_DS$
3. Normalize DS_q such that $\sum_q DS_q = dailyCapacity$
4. **if** $sales_q > DS_q$ & $slot_q <= GOOD\_SLOT$ **then** $PM* = INC\_PM$
5. **if** $sales_q > DS_q$ & $!(slot_q < BAD\_SLOT)$ **then** $PM* = DEC\_PM$

Here, $avgPPC$ is defined in the following way:

$$avgPPC = \frac{\sum_q revenue_q - cost_q}{\sum_q clicks_q} \tag{4}$$

Substitute $avgPPS$ for $avgPPC$, we get the rules for ClickProfitS. Similarly, $avgPPS$ can easily calculated as:

$$avgPPS = \frac{\sum_q revenue_q - cost_q}{\sum_q sales_q} \tag{5}$$

In the experiments, we set $INC\_DS = 1.25$, $DEC\_DS = 0.8$.

From these two agents, we notice that the rules tends to equate profit across queries. If a query has a good profit, then by rule 1, we will increase its DS_q, and this will likely trigger rule 5, thus decreasing its profit. Likewise, if a query has a bad profit, then by rule 2, we will decrease DS_q, and this will likely trigger rule 4, thus increasing its profit

3.3 EquateProfitC, EquateProfitS

In economic terms, profit is maximized when the marginal profit is zero. Here, since we have a 5-day capacity constraint, for simplicity, we assume that we use 1/5 of the total capacity everyday. Therefore, our marginal profit does not have to be 0. Rather, we can obtain an optimal bidding strategy by equating the marginal profit of each query. However, the real marginal profit is undefined because the slots are discrete and thus the profit function is likely to be discrete.

We can use PPC and PPS to approximate marginal profit. The advantage of this approximation is that we can design model-light agents easily. We can guess a common PPC (EquateProficC)/PPS (EquateProfitS) for all queries, and set bids for each query such that:

$$cpc_q = (USP_q * pr(conv)) - PPC \tag{6}$$

$$cpc_q = (USP_q - PPS) * pr(conv) \tag{7}$$

This bidding strategy also implies a way to allocate capacity across all queries. If a query has low USP_q and low $pr(conv)$, both equation 6 and 7 will yield low expected cpc's. Then adjust the common PPS/PPC by total conversions:

1. **if** $\sum_q sales_q > dailyCapacity$ **then** $PPC/PPS* = INC\_P$
2. **if** $\sum_q sales_q < dailyCapacity$ **then** $PPC/PPS* = DEC\_P$

In our implementation, we set $INC\_P = 1.1$ and $DEC\_P = .9$.

3.4 PPC vs. PPS

PPC captures the profit of one additional click, while PPS captures the profit of one additional conversion. The two approaches yield different bidding patterns. We illustrate the difference in a query space of two queries: q_1 and q_2 by comparing the difference between their desired cpc's, denoted Δcpc.

Scenario 1 Suppose $USP_1 = USP_2$ and $pr(conv)_1 \neq pr(conv)$. Equating PPS yields:

$$\Delta cpc = (USP - PPS) * \Delta pr(conv) \tag{8}$$

While equating PPC yields:

$$\Delta cpc = USP * \Delta pr(conv) \tag{9}$$

Scenario 2 Suppose $USP_1 \neq USP_2$ and $pr(conv)_1 = pr(conv)$. Both approaches yield:

$$\Delta cpc = \Delta USP * pr(conv) \tag{10}$$

Scenario 1 shows that when two queries have the same USP, both approaches will expect a higher CPC on the query with higher conversion probability. The desired CPC's from equating PPC has a bigger gap than from equating PPS. Scenario shows that if the USP's are different but conversion probabilities are the same, the two approaches are the same.

Empirically, equating PPS is far better than equating PPC. Equating PPC has a bigger fluctuation of capacity used even in a five day window.

3.5 Suboptimality

We have argued that neither equating PPS nor equating PPC is optimal in theory, despite the fact that EquatePPS outperforms many sophisticated agents. Here we give an insight of the suboptimality of equating PPS. For simplicity, still consider a query space of only two queries, and we assume that we always exhaust *dailyCapacity*.

The total profit gained by equating PPS is:

$$totalProfit = PPS * (sales_1 + sales_2) = PPS * dailyCapacity \tag{11}$$

Now suppose we have another bidding strategy, such that:

$$\begin{aligned} cpc'_1 &= cpc_1 + \delta_1 \\ cpc'_2 &= cpc_2 - \delta_2 \\ sales'_1 + sales'_2 &= dailyCapacity \end{aligned}$$

Then, the profit of this alternative is:

$$\begin{aligned} totalProfit' &= [PPS - \frac{\delta_1}{pr(conv)_1}]sales'_1 + [PPS + \frac{\delta_2}{pr(conv)_2}]sales'_2 \\ &= totalProfit - \frac{\delta_1}{pr(conv)_1}sales'_1 + \frac{\delta_2}{pr(conv)_2}sales'_2 \end{aligned} \tag{12}$$

Therefore, the alternative is an improvement iff:

$$\frac{\delta_1}{pr(conv)_1}sales'_1 < \frac{\delta_2}{pr(conv)_2}sales'_2 \tag{13}$$

Or,

$$\delta_1 clicks'_1 < \delta_2 clicks'_2 \tag{14}$$

Note that this analysis can be extended to more than two queries.

4 Model-Heavy Algorithms

4.1 Intro

Using the models described in section 5, we were able to use more sophisticated bidding strategies. We formulated the ad auctions bidding problem as a Multiple Choice Knapsack Problem (MCKP), and as an Integer Linear Program (ILP). We evaluated both bidding strategies in the TAC AA game and a simulated environment with perfect information.

4.2 MCKP

A Multiple Choice Knapsack Problem is when one needs to decide how to choose one item from each of several item sets while trying to maximize ones profit and not pick items that are heavier than the knapsack capacity. This problem can be adapted to the ad auctions setting by letting items be bids, item sets be queries, and the total weight that the knapsack cannot exceed be our capacity, the value of items be profit, and the weight of items be

This problem can defined as such: let m be the number of queries, $\vec{b}$ the vector of bids we are considering for each query, v_{il} be the profit associated with bidding the jth bid in the i query, w_{ij} be the weight (or the number of conversions), and C be our capacity.

$$\max \sum_{i=1}^{m} \sum_{j \in \vec{b}} v_{ij} x_{ij}$$

$$\text{Given} \quad \sum_{i=1}^{m} \sum_{j \in \vec{b}} w_{ij} x_{ij} \leq C$$

$$\sum_{j \in \vec{b}} x_{ij} = 1 \qquad 1 \leq i \leq k$$

$$x_{ij} \in \{0, 1\} \qquad 1 \leq i \leq k, j \in \vec{b}$$

We implemented two algorithms to solve the knapsack problem: an incremental item algorithm and a dynamic programming solution.

4.3 Dynamic Progamming MCKP

Observe that C is small and discrete, we can easily solve the MCKP by dynamic programming. Define function $f(i,j)$ as the maximum profit of considering queries 1 up to i fulfilling capacity j, where $0 \le j \le C$.
Boundary condition:

$$\begin{aligned} f(0,j) &= 0,\ \forall i \in \{1,...,m\} \\ f(i,0) &= 0,\ \forall j \in \{0,...,C\} \end{aligned} \tag{15}$$

Otherwise

$$f(i,j) = \max_{k \in \vec{b}} \max_{l < maxClick(k)} f(i-1, j-l) + v_{i,k,l} \tag{16}$$

Here function $maxClick(k)$ refers to the clicks given by the bid-to-click model. $v_{i,k,l}$ is the profit of bid k and get l clicks in the auction for query i. The time complexity is $O(mC\|\vec{b}\|)$.

In order to retrieve the optimal bidding strategy, we need to remember the bid k and conversions l we have chosen for each $f(i,j)$.