Scalable Winner Determination in Advertising Auctions

David J. Martin
Cornell University
djm@cs.cornell.edu

Johannes Gehrke
Cornell University
johannes@cs.cornell.edu

Joseph Y. Halpern
Cornell University
halpern@cs.cornell.edu

Abstract

Internet search results are a growing and highly profitable advertising platform. Search providers auction advertising slots to advertisers on their search result pages. Due to the high volume of searches and the users' low tolerance for search result latency, it is imperative to resolve these auctions fast. Current approaches restrict the expressiveness of bids in order to achieve fast winner determination, which is the problem of allocating slots to advertisers so as to maximize the expected revenue given the advertisers' bids. The goal of our work is to permit more expressive bidding, thus allowing advertisers to achieve complex advertising goals, while still providing fast and scalable techniques for winner determination. We also discuss the application of our framework to advertising in massively multiplayer online games.

1. Introduction

With the number of Internet searches performed every day, search result pages have become a thriving advertising platform. The results of a search query are presented to the user as a web page that contains a limited number of slots, typically between four and twenty, for advertisements. On each search result page, the major search engines, like Google and Yahoo, sell these slots to advertisers via an auction mechanism that charges an advertiser only if a user clicks on his ad. Most of Google's multi-billion dollar revenue, and more than half of Yahoo's revenue, comes from these so-called sponsored search auctions [8]; and this market is growing quickly – by 2008, spending by US firms on sponsored search is expected increase by $3.2 billion from 2006 and will exceed $9.6 billion, the amount spent on all of online advertising in 2004 [9]. With the increasing market size in mind, it is natural to approach sponsored search auctions from a database perspective in order to tackle issues of scalability and expressiveness. Our paper is a first step in this direction.

Sponsored search auctions currently work as follows:

1. **Bid submission.** Advertisers submit bids on clicks for certain keywords offline.
2. **User search.** A user submits a search query.
3. **Winner determination.** Slots are assigned to advertisers by the search provider based on the advertisers' bids.

4. **User action.** The search result page is returned to the user who may now click on one or more of the sponsored links.
5. **Pricing and payment.** The search provider charges an advertiser according to some pricing rule if the user clicked on the advertiser's sponsored link.

The speed of the winner determination in Step 3 is crucial because it contributes to the user-experienced latency since the winning ads are displayed on the search result page returned to the user. In current sponsored search auctions, this winner determination can be done quickly because advertisers are limited to submitting a single bid on whether or not the user clicks on their ad.

1.1. The Need for Expressive Auctions

Unfortunately, as we now point out, the limited bidding in current sponsored search auctions is insufficient to meet advertisers' needs in two respects.

Bidding on Multiple Features. Once the advertisers' ads are displayed on the search results page, the user who submitted the query may click on the ad and may even make a purchase as a result. Advertisers clearly value purchases because they represent immediate revenue. They also value clicks on their ads because they indicate potential customers. However, even if the user does not click on or buy something, advertisers might place value on having their ads displayed simply because this increases their chance to make an impression on the customer. Advertisers who value brand awareness may wish their ads to be placed in prominent positions. Such advertisers may prefer their ads to be displayed near the top or bottom of the list, but not in the middle. Other advertisers whose goals are to be perceived as the leaders in their markets may wish their ads to be displayed in the topmost slot or not displayed at all. Thus it is clear that advertisers have valuations on clicks, purchases, and slot positions.

Unfortunately, in current search advertising platforms, advertisers are restricted to bidding only on whether they receive a click on their ad. We call this a *single-feature auction* since the advertisers can express their valuations on only one feature, namely, receiving a click. Our goal is to support *multi-feature auctions* that would allow advertisers to express valuations on multiple features, namely, clicks, purchases, and slot positions. Extending bidding to multiple features is non-trivial; whereas previously the advertiser submitted a single value as depicted in Figure 1, now the advertiser can submit a whole table of values for the different combinations of features, as depicted in Figure 2. The fast algorithms for winner determination that are currently used by Google and Yahoo! do not extend to non-trivial multi-feature auctions. Moreover, even for single-feature auctions, these algorithms can correctly deal with only a restricted situation, namely, one where the expected number of clicks on an ad is "separable" into the product of an advertiser-specific factor and a slot-specific factor.

Dynamic Bidding Strategies. The language that search providers, such as Google and Yahoo, currently use to let advertisers express bidding preferences in is rather limited. While the language does allow advertisers to specify a limited number of parameters to constrain their bids (such as a daily budget, and geographic targets), the language is often insufficiently expressive for serious advertisers to express their preferences and how they change over time. To deal with this, advertisers employ the services of various third-party search engine management companies (such as iProspect, SureHits, Atlas, etc.) that monitor the outcomes of auctions and periodically resubmit bids on behalf of the advertiser in an attempt to approximate the advertisers' preferences as much as possible. The kinds of goals that they try to achieve include maintaining a specified slot position during certain hours of the day, maintaining a

slot position above a specified competitor, and equalizing the return on investment (ROI) across multiple keywords.[1] The success of such search engine management companies demonstrates the desire among advertisers for more complex expressive bidding in search auctions. Again, advertisers want these, but can only pick from a set of pre-defined strategies that these companies provide.

1.2. Our Framework

With the increasing market size in mind, our goal is to design a framework that allows huge numbers of advertisers to bid on a richer set of features using dynamic bidding strategies while simultaneously allowing the search provider to determine winners quickly so as not to detract from the user experience [4].

Bidding Language. In this paper, we propose a simple but rich language for bidding that allows advertisers to express their high-level strategies directly; we allow users to submit their dynamic strategies as *bidding programs* that can bid on multiple features of the auction outcome, such as purchases and slot positions, in addition to clicks. Programs take as input the search query and various statistics about auction history and performance, and they output bids on clicks, purchases, and slot positions. Using this language gives advertisers direct and fine-grained control over their advertising strategies instead of simply picking from a menu of pre-defined goals, as is currently done. Thus, in our framework, the search auctions work as follows:

1. **Program submission.** Advertisers submit a bidding program to bid on their behalf.
2. **User search.**
3. **Program evaluation.** The programs are run and place bids on clicks, purchases, and slot positions.
4. **Winner determination.**
5. **User action.**
6. **Pricing and payment.**

Scalable Algorithms. In addition to proposing our language, we propose an algorithm to perform fast winner determination for multi-feature auctions where the advertiser can bid on clicks, purchases, and slot positions simultaneously, under the assumption that the conditions that determine an advertiser's preferences satisfy a condition that can be viewed as a generalization of separability; moreover, we prove that this requirement is in a sense necessary. We then propose techniques for reducing the amount of work that needs to be done when evaluating dynamic strategies of many advertisers. This results in a scalable infrastructure for multi-feature auctions with dynamic strategies.

Application to Advertising in Games. Our algorithms are applicable to more than just web search. Massively multiplayer online games are becoming are starting to become a very successful advertising platform. Games have unique benefits over web search for advertising because of the sense of immersion that the player feels, and because of the well-defined and self-contained social networks present in games that can allow advertisers to target specific market segments. Furthermore, we describe how to leverage existing technology built into most games to accurately track ad prominence and exposure,

[1] In Section 2.3, we revisit the dynamic ROI equalizing strategy and illustrate how we can implement such a complex real-world bidding strategy in our framework.

and to implement a novel ad "bookmarking" system. We discuss these issues and describe how to our advertising framework for web search can be adapted to advertising in massively multiplayer games.

Summary of our contributions. In this paper, we approach sponsored search auctions from a database perspective, and tackle issues of scalability and expressiveness. Our main contribution is an efficient and scalable infrastructure that permits much more expressive bidding than is currently available. Our framework consists of:

- A language for expressing dynamic bidding strategies for multi-feature sponsored search auctions (Section 2),
- An efficient, scalable, and parallelizable algorithm to solve winner determination for bids in our language (Section 3),
- Techniques to reduce the amount of work necessary for evaluating dynamic strategies for multiple advertisers (Section 4).

We evaluate our techniques experimentally in Section 5, we discuss the application of advertising auctions to massively multiplayer games in Section 6, and we conclude in Section 7.

2. Bidding Strategies as Programs

In this section, we formalize the notion of bidding on multiple features, and we propose a simple language for dynamic strategies that bid on these features.

2.1. Multiple Features

Recall that traditionally an advertiser could only bid on one property of the outcome, namely, whether his ad received a click. Now we would like to allow advertisers to bid on additional properties as well, namely whether a purchase was made, and whether his ad was displayed within a desired set of slots. To each advertiser, we make available the following predicates that indicate whether or not the outcome has one of these desired properties.

1. Slot_j, indicating that the advertiser gets slot j, for $j \in \{1, \ldots, k\}$, with k being the number of slots.
2. Click, indicating that the user clicked on the advertiser's ad.
3. $\mathrm{Purchase}$, indicating that the user made a purchase via a link from the advertiser's ad.

Conceptually, the advertiser associates a value with each truth assignment to these predicates, as depicted in Figure 2. However, the size of such a representation is exponential in the number of predicates. So we represent bids as OR-bids on Boolean combinations of predicates instead. That is, we let the advertiser fill in a *Bids table* where each row corresponds to a Boolean formula of predicates and the amount that he is willing to pay should that formula be true. If multiple formulas are true, the advertiser can be charged the sum of the corresponding amounts. For example, the Bids table depicted in Figure 3 indicates that the advertiser is willing to pay 5 cents if he gets a purchase; 2 cents if his ad is displayed in either positions 1 or 2; and 7 cents if he gets a purchase *and* his ad is displayed in positions 1 or 2.

Click	value
Y	3

Figure 1. Single-feature valuation

Purchase	Click	Slot_1	Slot_2	Slot_3	value
Y	Y	Y	N	N	7
N	Y	Y	N	N	2
⋮	⋮	⋮	⋮	⋮	⋮
Y	Y	N	N	Y	5
N	Y	N	N	Y	0
⋮	⋮	⋮	⋮	⋮	⋮

Figure 2. Multi-feature valuation

2.2. Dynamic Strategies

As we said, we are interested in designing a programming language that lets advertisers express more complex preferences, which may change over time. Instead of providing advertisers with a pre-defined selection of advertising strategies, we let the advertisers submit their bidding strategies as programs for the search provider to run. Conceptually, each time a user submits a search query to the search provider, these programs are triggered. The main purpose of these programs is to output bids on clicks, purchases, and slot positions that may result from displaying their ad on the search result page. In order to do so, each program creates a Bids table as described in Section 2.1 each time there is a sponsored search auction. These programs have access to several variables pertinent to the current auction and to the advertiser, such as the keywords in the search query, the time of day, the advertiser's remaining budget, the current return on investment for the keywords that the advertiser is interested in, and so on. These variables are stored in tables, some of which are read-only shared between all advertisers (such as the time and location of the search) and some of which are private to each advertiser (such as information about the keywords that the advertiser is interested in). The programs can then be written using simple SQL updates without recursion and side-effects. SQL triggers can be used to activate programs when an auction begins and to notify programs if they received a slot, click, or purchase. Programs can modify their private tables, although commonly used variables, such as amount spent, budget remaining, return on investment for various keywords, etc. can be automatically maintained for each program by the search provider. For example, the advertiser-specific variables related to keywords are stored in a Keyword table, as depicted in Figure 4 that is private to each advertiser. Each tuple in the Keyword table corresponds to a bid for a keyword that the advertiser is interested. The attributes of the tuple contain, among other things, the formula for the bid, keyword's relevance score in the search query, the return

formula	value
Purchase	5
$\text{Slot}_1 \vee \text{Slot}_2$	2

Figure 3. Bids table

text	formula	maxbid	roi	bid	relevance
boot	$\text{Click} \wedge \text{Slot}_1$	5	2	4	0.8
shoe	Click	6	1	8	0.2

Figure 4. Keywords table

on investment that this keyword has provided the advertiser, the maximum amount that the advertiser is willing to bid on a click by a user who searched for this keyword, and the amount of money that the advertiser is currently bidding for the keyword. The search provider updates the return on investment for a keyword each time a user searches for the keyword and then clicks on the advertiser's ad. The bidding program should be stored with its private tables to increase locality. Since the bidding programs use their own private tables and read-only shared tables, they do not interact with each other when they are triggered by a new search query. Hence they can be distributed across several machines and run in parallel if necessary.

2.3. An Example: Equalizing ROI

We now give a concrete example of a dynamic bidding strategy that bids on multiple features. Our example combines the dynamic ROI equalizing heuristic mentioned in Section 1 with bidding on two features – clicks and the top slot; the advertiser is interested in receiving clicks for two keywords, "boot" and "shoe", but also wants to be perceived as the leading supplier of boots and so would be willing to pay extra to be shown in the top slot if the search query is highly relevant to boots. In order to control his spending, the advertiser has a target spending rate that he wishes to maintain. The ROI equalizing heuristic, as suggested in [5], tries to dynamically allocate spending across the different keywords and bids so as to maximize the advertiser's "bang for the buck". If the advertiser is underspending (i.e., his current spending rate is lower than his target spending rate), then the advertiser increases the bids on keywords that have been most profitable for him (i.e., those with the highest return on investment). If the advertiser is overspending (i.e., his current spending rate is higher than his target spending rate), then the advertiser decreases the bids on keywords that have been least profitable for him (i.e., those with the lowest return on investment). Return on investment of a bid is the total value gained from the keyword (e.g., number of clicks received in the top slot times the amount the advertiser values a click in the top slot) divided by the amount spent so far on it.

```
 1 CREATE TRIGGER bid AFTER INSERT ON Query
 2 {
 3   IF amtSpent / time < targetSpendRate
 4   THEN
 5     UPDATE Keywords
 6     SET bid = bid + 1
 7     WHERE roi =
 8       ( SELECT MAX( K.roi )
 9         FROM Keywords K )
10       AND relevance > 0 AND bid < maxbid
11   ELSEIF amtSpent / time < targetSpendRate
12   THEN
```

formula	value
$\text{Click} \wedge \text{Slot}_1$	4
Click	0

Figure 5. Bids table for Example Program

```
13     UPDATE Keywords
14     SET bid = bid - 1
15     WHERE roi =
16       ( SELECT MIN( K.roi )
17         FROM Keywords K )
18       AND relevance > 0 AND bid > 0;
19   ENDIF;
20
21   UPDATE Bids
22   SET value =
23     ( SELECT SUM( K.bid )
24       FROM Keywords K
25       WHERE K.relevance > 0.7
26         AND K.formula = Bids.formula )
27 }
```

A detailed explanation of this program follows. Line 1 creates a trigger that waits for a new query to be inserted into the Query table, indicating that a new auction is taking place. If the advertiser notices that he has been underspending (line 3), he increases his tentative bids for all relevant keywords that have provided him with the highest return on investment, taking care not to increase the bid past its maximum value (lines 5 - 11). Similarly, if he notices that he has been overspending (line 12), he decreases his bids for all relevant keywords with the lowest return on investment, taking care not to decrease the bid below zero (lines 14 - 20). Next, he updates the values in the Bids table with the sum of his tentative bids for the corresponding formulas for all sufficiently relevant keywords, namely, those with a relevance score higher than 0.7 in the user-submitted search query (lines 21 - 26). For example, if the Keywords table is as depicted in Figure 4 after running lines 1 - 20, then the output Bids table will be as depicted in Figure 5.

3. Winner Determination

Having empowered the advertisers with a language for expressing dynamic bidding strategies to bid on a rich set of features, we now seek efficient and scalable techniques for the search provider to perform winner determination.

All sponsored search auction mechanisms currently in use (see, for example, [1, 2, 8, 22]) first solve the winner determination problem, then assign slot positions according to the winning allocation, and finally use some method of charging prices for the positions, such as charging each advertiser their social opportunity cost (this is known as Vickrey pricing [7, 13, 23]), or charging advertiser in the kth slot the amount bid by the next-highest bidder (this is known as generalized second-pricing [8]). Note that with most pricing schemes, a provider's revenue is *not* the revenue that is computed in the winner

determination problem. Different pricing schemes lead to different behavior of the auction in terms of revenue, stability, and other economic and game-theoretic properties. For example, Vickrey pricing leads to theoretically stable truthful auctions [23], while generalized second pricing leads to locally envy-free equilibria [8]. Nevertheless, the first step in all these auctions is to do winner determination. Furthermore, given winner determination as a subroutine, the pricing schemes used in these auctions (i.e., Vickrey pricing, generalized second-pricing, etc.) can all be expressed as very simple computations. In our work, therefore, we focus on optimizing the winner determination computation.

3.1. How Winner Determination Works

The *winner determination problem* is to compute the allocation of slots to advertisers that results in the highest expected revenue for the search engine provider, under the assumption that advertisers actually pay what they bid. In keeping with Google and Yahoo policy, we restrict the slot allocations to those in which no advertiser gets assigned more than one slot. This prevents extremely wealthy advertisers from monopolizing all the available slots.

In order to compute the expected revenue resulting from an allocation, we need the advertisers' bids on clicks, purchases, and slot positions as specified in their Bids tables. For now, let us assume, that we actually run all of the advertisers' bidding programs to get their resulting Bids tables. In Section 4, we will give techniques that will require us to run only a small subset of programs under certain conditions.

In order to compute the expected revenue resulting from an allocation, we also need the probabilities that the formulas in the Bids tables are true in the final outcome. We thus consider the set of all possible outcomes that describe which slot was allocated to which advertiser together with which advertisers received clicks and purchases. The probabilities of clicks and purchases depend on the search provider's allocation of slots to advertisers. For example, ads placed near the top are more likely to be noticed and clicked on than those placed in the middle of the page [19]. As a reasonable first order of approximation, we assume that, for each advertiser, the probability that the advertiser gets a click depends only on the slot allocated to him, and that the probability that he gets a purchase depends only on whether he got a click and on the slot allocated to him. Furthermore, we assume that the search provider has (or can estimate, using data it has collected) these click and purchase probabilities for each advertiser and each slot allocation to that advertiser.[2]

3.2. Complexity

Given the assumptions on slot allocations and distributions above, we look at the complexity of solving the winner-determination problem given bids in our language. Recall that a bidding program's output is an OR-bid represented by a Bids table whose rows contain bids of the form "Pay \$$d_1$ for E_1", ..., "Pay \$$d_m$ for E_m", where $E_1, \ldots, E_m$ are Boolean combinations of the Slot_j, Click, and $\mathrm{Purchase}$ predicates. Recall that, in addition, we assume that for any allocation, we have a distribution on outcomes, conditional on that allocation. Each formula E_i can be identified with an *event* on the set of possible outcomes, namely, the set of outcomes in which E_i is true. Thus bidding on formulas can be interpreted

[2]Note that a complete representation of the probabilities of all possible formulas for each advertiser would be exponential in the number of features. Although this is not too large in our setting, the complete set of probabilities should be stored in a database separate from the run-time system which itself should only store probabilities for the formulas mentioned in the bidding programs and Keyword tables since these are the only probabilities that are used. Furthermore, the probabilities can be partitioned by advertiser and should be stored with the advertiser's bidding program and private tables to improve locality.

as bidding on events. Toward proving that winner determination is tractable for bids in our language, we introduce the following definition.

Definition 1 (m-dependent event) *An event is* m-dependent *if there are at most* m *advertisers such that probability of the event given any allocation depends only on the placement of those* m *advertisers.*

That is, an event is m-dependent if it is independent of the slots assigned to all but m advertisers. For example, the event that a given advertiser gets a click is 1-dependent since we assumed that the probability of an advertiser getting a click depends only on the slot position of that advertiser. Similarly, the event that a given advertiser is in either the top slot or the bottom slot is 1-dependent since it depends only on the slot assigned to that advertiser. However, given two advertisers, the event that one gets the top position and the second gets the bottom is 2-dependent since it depends on the slots assigned to both those advertisers.

We assume that the representation of each m-dependent event includes the labels of the m advertisers on whose slot assignment the event depends. The following theorem says that winner determination is tractable for 1-dependent events.[3]

Theorem 2 *For OR-bids on collections of* 1*-dependent events, the winner determination problem is in polynomial time.*

As a corollary, we see that, winner determination for the bids represented by our Bids tables can be solved in polynomial time since it follows from our assumptions in Section 3.1 that any Boolean combination of predicates for an advertiser (i.e., of the form $\mathrm{Slot}_1, \ldots, \mathrm{Slot}_k, \mathrm{Click}, \mathrm{Purchase}$) is 1-dependent.

A natural question to ask is whether we can extend our tractability results to a language that allows advertisers to bid on m-dependent events, for $m \geq 2$. The next result says that winner determination is *APX-hard* if we allow bids to be placed on 2-dependent events, such as the event that one advertiser is displayed above another. APX is the class of NP optimization problems that have polynomial-time constant-factor approximation algorithms [14].

Theorem 3 *For OR-bids on collections of* 2*-dependent events, the winner-determination problem is APX-hard.*

In the remainder of this section, we take the reader on a quest for an efficient and scalable winner determination algorithm for our bidding language.

3.3 Existing Allocation Algorithms

The allocation algorithms used by Google and Yahoo, as well as those studied in the literature [2, 1, 8, 22], deal with the issue of scalability by assuming that the probability of a click resulting from assigning a slot to an advertiser is *separable*, that is, it can be written as the product of an advertiser-specific factor and a slot-specific factor. To illustrate this notion of separability, we provide examples of separable and non-separable click probabilities in Figures 7 and 6 respectively. The matrix in Figure 7 is separable because the entries in the matrix can be split into the product of advertiser-specific factors (namely, 4 for Nike and 3 for Adidas) and slot specific-factors (namely, 0.2 for slot 1, and 0.1 for slot 2).

	Slot_1	Slot_2
Nike	0.7	0.4
Adidas	0.6	0.3

Figure 6. Non-separable click probabilities

	Slot_1	Slot_2
Nike	0.8	0.4
Adidas	0.6	0.3

Figure 7. Separable click probabilities

When the click probabilities are separable, it is easy to see that winner determination can be performed by assigning the advertisers with jth highest advertiser-specific factor to the slot with the jth highest slot-specific factor. This can be done in time $O(n \log k)$.

Note that the assumption of separability implicitly assumes that the event that an advertiser gets a click is 1-dependent. Indeed, it assumes the event that an advertiser gets a click depends on only that advertiser's slot assignment. But separability requires much more 1-dependence: it requires that the ratio of the expected number of clicks on one advertiser in a slot and the expected number of clicks on another advertiser in the same slot is the same for all slots.

Not only is separability a much stronger requirement than 1-dependence, but the techniques for fast winner determination using this assumption do not suffice to deal with our bidding language. In particular, they cannot deal with the types of situations described in Section 1 where one advertiser wants to be displayed in the top slot or not displayed at all, while another advertiser wants to be displayed in either the top or bottom slots but not in the middle slots. (Bids representing these preferences can be easily expressed in our language.)

3.4. Maximum-Weight Bipartite Matching

We proved Theorem 2 by showing that winner determination in this case is equivalent to maximum-weight bipartite matching between advertisers and slots, where the edge-weight between an advertiser and a slot is the expected revenue obtained by assigning that slot to that advertiser. The fastest known (non-parallel) algorithm to solve this is the Hungarian algorithm, invented by Kuhn [16] (also known as the Kuhn-Munkres algorithm after being revised by Munkres [17]); it finds the best matching in time $O(nk(n+k))$ where n is the number of advertisers and k is the number of slots. Since this is quadratic in n, this will not scale well. We want to deal with situations where n can be quite large (possibly in tens to hundreds of thousands). To make the problem scalable, we need it to be *linear* in n, the number of advertisers. There are parallel algorithms for maximum-weight matching [11], but these require prohibitively large numbers (typically $\Omega(n^2)$) of processing units in order to achieve linear running time.

3.5. Our Algorithm

We now give a scalable winner-determination algorithm that takes advantage of the fact that k, the number of slots, is quite small (say less than 20) compared to n, the number of advertisers. Indeed, n

[3] See appendix for proofs.

	Slot_1	Slot_2
Nike	**9**	5
Adidas	**8**	**7**
Reebok	7	**6**
Sketchers	7	4

Figure 8. Revenue matrix

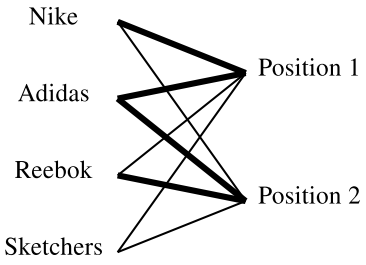

Figure 9. Bipartite graph

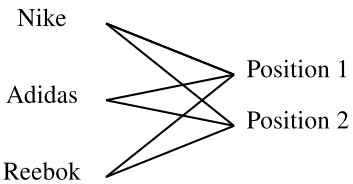

Figure 10. Reduced graph

is growing rapidly every year while k remains the same. We can modify the Hungarian algorithm to get a $O(nk \log k + k^5)$ algorithm by considering only those advertisers whose values are in the top k highest for some slot. That is, for each slot, we consider the k advertisers who would produce the top k expected revenue if placed in that slot. We take the union of these advertisers over all the k slots, and consider the bipartite subgraph containing only these advertisers along with all the k slots. We then solve maximum-weight bipartite matching problem for this reduced bipartite graph. As an example, consider the expected revenue matrix as depicted in Figure 8. There are two slot positions available and four advertisers. The top two expected revenues for the first slot come from Nike and Adidas, while the top two expected revenues for the second slot come from Adidas and Reebok. The corresponding edges in the original bipartite graph between advertisers and slots have been depicted in bold in Figure 9. This bipartite graph is then reduced to contain only those advertisers with an adjacent bold edge as depicted in Figure 10. We observe that the maximum matching for the original problem must occur for this smaller problem since if an maximum matching in the original problem assigned a slot to an advertiser who was not in the top k highest bidders for that slot, we can simply reassign that slot to one of these top k bidders who is not assigned any slot. Note that since there are only $k - 1$ other slots, at least one advertiser in the top k is guaranteed to remain unassigned.

Finding the relevant advertisers takes time $O(nk \log k)$ because, for each slot, we can find the top k bidders for that slot in time $O(k + n \log k)$ by maintaining a priority heap of size at most k. There are at most k^2 such advertisers since in the worst case we will have a distinct set of k advertisers for each of the k slots. Hence running the Hungarian algorithm on the reduced graph takes time $O(k^5)$ for a total running time of $O(nk \log k + k^5)$ for our algorithm.

Parallelization. Our technique lends itself very well to parallelization. Note that in our setting there is typically already a high amount of parallelized infrastructure present since the bids are collected from advertisers in a distributed way. We construct k networks of computers each in the form of a binary tree of height $O(\log n)$ with n leaves. We can compute a maximum matching in time $O(k \log n + k^5)$ as follows. For each slot j, we consider the jth binary tree network, which will ultimately compute the top k bidders for that slot at the root:

1. The ith leaf node in the jth network starts out with the expected revenue from assigning slot j to advertiser i.
2. Each internal node gathers the top k bidders (along with their corresponding bids) from its two children, and combines them into a single list of top k bidders. This takes time $O(k)$ for each of the $O(\log n)$ levels of the tree since each level of the tree works in parallel.

3. The root nodes in each of the j-networks take the union of their lists of bidders and compute the maximum-weight matching of these bidders with the k slots using the Hungarian algorithm. This takes time $O(k^5)$ since there are k slots and at most k^2 bidders considered.

Note that we can mix sequential processing with parallel processing by running more than one program sequentially on each machine, computing the top k bids, and then aggregating using a tree network as before. If we have a binary tree network with p nodes, then the total running time becomes $O(\frac{n}{p}k \log k + k \log p + k^5)$.

Finally, if necessary, the $O(k^5)$ part of the algorithm (i.e., the part resulting from running the Hungarian algorithm on the reduced bipartite graph) can be reduced to $O(k^2)$ using a parallel algorithm, such as in [11]. The number of parallel nodes processing units required would be $O(k^5)$, which is independent of n.

3.6. Beyond 1-dependence

So far, our results have assumed that the probability that an advertiser receives a click or a purchase depends only on the slot to which that advertiser was assigned. However, it is easy to think of situations where this assumption might not be true. For example, if the slot assigned to an advertiser for a small company is just below a very large and popular competitor, then it is likely that the competitor will receive a substantial portion of user clicks that might otherwise have gone to the smaller advertiser had the competitor not been present. Thus the probability of receiving a click (or a purchase) would depend on who else displays an ad and in what position. In the worst case, the probability would depend on the entire slot assignment. The representation of such a general probability distribution would be quite large ($O(kn^k)$), and, conceptually, winners can be determined by a brute force algorithm that considers each of the possible $\binom{n}{k}k!$ assignments. This would also lead to advertisers to value two assignments differently even if both assignments may give the advertiser the same slot. For example, consider two assignments, both of which assign an advertiser slot 2. However, in the first assignment, slot 1 is given to a very famous company, while in the second assignment, slot 1 is given to a relatively unknown company. Then the advertiser in slot 2 would naturally prefer the second assignment to the first, since the famous company poses a serious threat to the advertiser in terms of diverting away clicks. Representing such general valuations would again require large space ($O(kn^{k-1})$) in general.

Motivated by these concerns, but keeping in mind that we cannot store such huge distributions and valuations (since n can be very large), we propose the following model. For a given search auction, we assume that the advertisers are classified into either *heavyweights* (the famous advertisers) and *lightweights* (the relatively unknown advertisers). One way for the search provider to decide which advertisers are heavyweights is to select those advertisers with the most clicks so far. We now allow the probability that an advertiser gets a click (or a purchase) on the advertiser's slot position as well as on which slot positions have heavyweight advertisers and which slot positions have lightweight advertisers. We also allow advertisers to place bids on which slots get heavyweights and which clots get lightweights, in addition to placing bids on click, purchases, and slot positions as before. Thus an advertiser might bid 3 cents if he gets slot 2 and if there is a lightweight advertiser in slot 1. Advertisers could even place more complex bids, such as bidding on having no heavyweights within 3 slot positions above or below his slot in addition to having no more than 2 heavyweights appear anywhere else. The representation of the probability distributions and valuations now become $O(k2^{k-1})$ which does not depend on n anymore.

In order to solve the winner determination problem, we have to find an assignment of slots to advertisers to maximize expected revenue (assuming advertisers pay what they bid) given these new valuations and distributions. Suppose we knew exactly which slots get heavyweight advertisers in such a revenue maximizing assignment. We call these slots *heavyweight slots*, and we call the remaining slots *lightweight slots*. Then we can solve the winner determination problem by simply solving two disjoint maximum-weighted bipartite bipartite matching problems: one matching the heavyweight advertisers to the heavyweight slots, and the other matching the lightweight advertisers to lightweight slots. So if we do this for each possible way to choose heavyweight slots, we can find the assignment that maximizes expected revenue over all possible assignments. Moreover, the maximum-weight bipartite matching problems for different choices of heavyweight slots can be solved independently and in parallel. Therefore, since there are 2^k ways to choose heavyweight slots, we can solve the winner determination in time $O(2^k(n \log k + k^5))$ in series, and in time $O(n \log k + k^5)$ in parallel using 2^k processing units, each of which solves the lightweight and heavyweight bipartite matching for a different choice of heavyweight slots. Note that the number of processing units required in the parallel algorithm is independent of the number of advertisers n.

4. Reducing Program Evaluation

We have shown how to solve the winner-determination program given the bids output by programs. However, getting these bids for a given search query requires, in the worst case, running each advertiser's program for that query. This itself can be quite expensive. An obvious step toward alleviating this problem is for search providers to use their proprietary keyword matching algorithms to prune away advertisers who are not interested in the search keywords for the current auction. However, this is not enough if the search query contains a very popular keyword, such as "music" or "book", where the set of interested advertisers can still be large. In this section, we show that we can further reduce the amount of work by taking advantage of knowledge of the structure of, and relationships between, the advertiser's programs. To simplify exposition, we assume that advertisers' programs output bids on only the formulas $\mathrm{Click} \wedge \mathrm{Slot}_1, \ldots, \mathrm{Click} \wedge \mathrm{Slot}_k$. It is easy to incorporate bids on other complex formulas involving Click and $\mathrm{Purchase}$ since both Click and $\mathrm{Purchase}$ are assumed to be 1-dependent events.

4.1 Threshold Algorithm

We start by considering a situation where the only difference between the programs used by different advertisers is in the values of certain advertiser-specific parameters. More precisely, for each slot $j \in [k]$, suppose that each advertiser's bids depends on a set of (numeric) parameters X_j in a monotonic way. That is, there is a monotonic function $f_j : X_j \to \mathbb{R}_+$ that takes as input a value for each parameter in X_j and outputs a bid for a click in slot j. We allow some subset of the parameters Y_j to be advertiser-specific: these can vary from advertiser to advertiser (e.g., the amount that they value a particular keyword, the amount of budget remaining, etc.). Suppose further that these parameters Y_j are only updated by programs that win the auction.[4] The rest of the parameters $Z_j = X_j \setminus Y_j$ can be thought of as public global parameters and are the same for all advertisers (e.g., the keyword scores associated with the user's

[4] In Section 4.2, we will visit the case where all programs can update their state; nonetheless, restricting updates to winning programs is not unreasonable since most useful advertiser-specific quantities (such as number of auctions won, amount spent so far, return on investment for a given keyword, etc.) only change when the advertiser wins an auction.

search query, the time and date, the number of times the keywords in search query have appeared today). A simple example of such a situation is where advertisers all use the same general strategy of starting each day by bidding low and then gradually increasing their bids as the end of the day approaches. However, they might each start with a different amount and might increase their bids at different rates. Then the starting amounts and the rate of increase would be advertiser-specific parameters in Y_j, and the time of day would be a global parameter in Z_j.

For each advertiser i and each slot j, we let the edge weight between advertiser i and slot j be $w_{i,j} \times f_j(y_{i,j}, z_j)$ where $w_{i,j}$ is the probability of advertiser i getting a click in slot j, and $y_{i,j} \in Y_j$ are the values of the advertiser-specific parameters and $z_j \in Z_j$ are the values of the global parameters. We previously showed that we can solve the maximum-weight matching in time $O(nk \log k + k^5)$. Under the assumptions above, we can further reduce the $O(nk \log k)$ portion which finds the top k bidders for each slot as follows. For a given slot j, we also store a list of bidders sorted by $w_{i,j}$ and we incrementally maintain $|Y_j|$ lists of bidders, each sorted by one of the parameters in Y_j. We can then run the *threshold algorithm* [10] with these lists as input to find the top k advertisers with the highest values of $w_{i,j} \times f_j(y_{i,j}, z_j)$. Note that we do not need to maintain lists for the parameters in Z_j since all advertisers have the same value for these parameters. Since f_j was monotonic, the threshold algorithm is instance optimal for the class of algorithms that find the advertisers with the top k values of $f_j(x_{i,j})$ without making "wild guesses" (i.e., the algorithms must not access an advertiser until that advertiser is encountered via a sequential scan of one of the lists). Instance optimality means that, for all inputs, the threshold algorithm finds the top k values within a constant factor of the time it takes the fastest algorithm that avoids wild guess for that particular input. Given these sets of k advertisers per slots, we take $O(k^5)$ further time running the Hungarian algorithm for just these advertisers to compute the winners. To maintain the sorted lists, once the k winners have been computed, we remove them from the sorted lists, update their Y_j parameters, and reinsert them into the sorted lists, which takes only $O(|Y_j|k \log n)$ time.

4.2. Logical Updates

We now consider the case where all program update their state, not just the winners. In certain situations, it is possible to reduce the amount of work done in this case as well. Consider a situation where many programs update their state using an operation that maintains their relative bid ordering. For example, suppose that many bidders are using the ROI heuristic described in Section 2.3, each with possibly different target spending rates and maximum bids. As long as certain conditions hold (namely, the bid is above zero and the spending rate is above the target spending rate), the heuristic will decrement its bid for a given keyword. Thus, if we can maintain a *decrement list* – that is, a list of programs, sorted by their bid, that are currently decrementing their bid for a given keyword – we can avoid explicitly decrementing each program's bid, by instead performing a single *logical* decrement in constant time. That is, the decrement list is associated with a single *adjustment variable*, initially zero. A program's bid is then the sum of the adjustment variable and the program's original bid. Then, in order to decrement the bids of all programs in the list, we simply decrement the adjustment variable. The sorted order is maintained because all programs in the list adjust their bids by the same amount.

Of course, eventually the ROI heuristic will stop decrementing its bid and start to increment it (if its spending rate drops below the target spending rate) or keep its bid constant (if its bid reaches zero) instead. At this point we must move the program to an *increment list* or a *constant list* as appropriate

(similar to a decrement list, except that the adjustment variable respectively increments or remains constant). At first glance, this would seem to involve checking checking the conditions for each program at every auction. However, we observe that such conditions can often be reduced to waiting for a shared monotonic variable (such as time, or the number of times a given keyword has occurred) to reach a *critical value*. For example, in the ROI heuristic, the spending rates of losing programs decreases with time since their amount spent remains constant. We can thus compute the earliest "critical" time in the future that a program would have to stop decrementing and start incrementing assuming it continued to lose. Similarly, we can compute the minimum number of auctions containing a given keyword necessary before its bid would be decremented to zero and it would have to remain constant at zero. We can thus maintain a list of triggers, sorted by critical value, that when activated move a bidding program to either the increment, decrement, or constant lists as appropriate and insert the appropriate new triggers. In this way, we only have to do any work for those programs that win an auction and for those losing programs whose critical values have triggered. We evaluate this technique in our experiments section.

5. Experiments

To evaluate our fast winner determination algorithm, we compare the performance of four methods for solving the winner determination problem. The first method (LP) solves the linear program formulation of the winner determination problem. We can prove that this linear program is guaranteed to have an integer optimum using a theorem of Chvátal [6] by showing that the rows of the constraint matrix represent the maximal cliques of a perfect graph. The second method (H) uses the Hungarian algorithm in a straightforward way to compute the maximum-weight bipartite matching in the bipartite graph with advertisers on the left and slots on the right, where the weight of an edge from an advertiser to a slot is the expected revenue from assigning that slot to that advertiser. The third method (RH) is our winner determination technique from Section 3.5, which first reduces the bipartite graph before using the Hungarian algorithm. The fourth method (RHTALU) augments the third with the techniques for reducing program evaluation from Section 4 using the threshold algorithm together with logical updates with triggers. We used 15 slots in all cases and ran 100 auctions. For simplicity, search queries were generated at a constant rate, each containing one keywords chosen uniformly at random out of 10 keywords. That chosen keyword was given a relevance score of 1 for that query, while other keywords had a relevance score of 0. All bidders used the ROI heuristic described in Section 2.2. For each keyword, the bidders' value for a click for was generated uniformly at random between 0 and 50, conditioned on the bidder having at a non-zero value for at least keyword. The target spending rates were chosen uniformly at randomly between 1 and the bidder's maximum value over all keywords. The interval $[0.1, 0.9]$ was partitioned into 15 disjoint intervals, with the $(j+1)$-highest interval associated with slot j. The probability of a given advertiser getting a click in a given slot was generated uniformly at random within that slot's interval. We used a slight generalization of generalized second-pricing to charge the advertisers who received clicks. The entire auction system, as well as the bidder's programs, were implemented in C++. We used the GNU Linear Programming Kit to solve the linear program via the simplex method.[5] We ran the experiments on an AMD Athlon 64 3800+ processor with 1GB of RAM.

Figure 11 shows, for each of the four methods, the time average taken per auction, over a sequence of 100 auctions, as we increase the number of bidders. We observed roughly an order of magnitude improvement of the Hungarian method over naive linear programming solution, and further order of

[5]We found that the library's interior point method was much slower than the simplex method for our workloads.

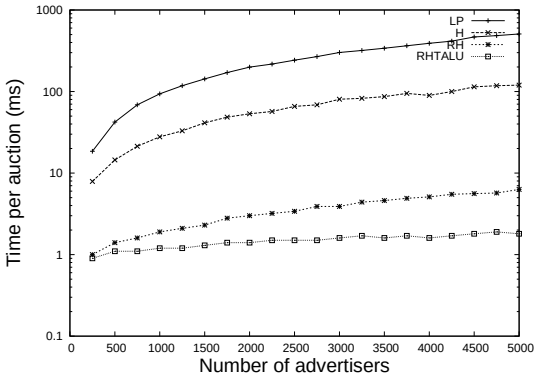

Figure 11. Winner Determination Performance

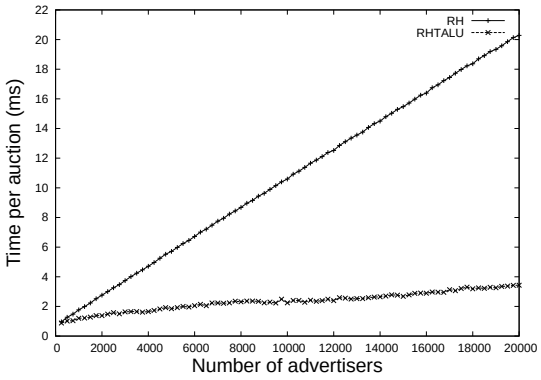

Figure 12. Reducing Program Evaluation

magnitude improvement using our reduced bipartite graph technique. Figure 12 compares the performance of methods RH and RHTALU in more detail. It plots the average time taken per auction, over a sequence of 1000 auctions, as we increase the number of bidders. We observe that our techniques for reducing program evaluation from from Section 4 give a significant further improvement in performance.

6. Advertising in MMOGs

Beyond web search, another setting to which sponsored ad auctions can be applied is massively multiplayer online games (MMOGs). In-game advertising in these games could prove to be a highly effective advertising platform – recent studies that included eye tracking have shown that 75% of gamers engage with at least one ad per minute across most, but not all, game types; 81% of gamers engage at least every other minute [12]. Advertising in the form of product placement and in-game billboards is already making their way into the current generation of games and in-game ad spending could reach $1.8 billion by 2010 [20]. We now describe how one could adapt our sponsored search auction framework to the auction of in-game billboards to advertisers. Product placement advertisement would work in a similar fashion.

The billboard ads shown to a player do not have to be generated statically. The game can select which ad to display dynamically (via an auction) just so long as it does so before the billboard is rendered on the player's screen. There are a number of ways to determine *when* to run the ad auction. For example,

it can be done just before drawing the first frame in which the billboard is visible. Or it can be done by having the level-designer manually place *trigger areas* on the game map that activates the routine for ad selection. Ideally, the trigger areas would be placed on the map so that the player would have to cross the trigger area before the billboard comes into view. Well-developed techniques such as binary space partitioning [21] can help to automate the process of identifying trigger areas. No matter which method is used to determine when to run the auction, there is a requirement of fast winner determination in order to keep the game running in real-time.

Player profile. Since the advertisements do not affect the gameplay, it would be acceptable for two players looking at the same virtual billboard simultaneously to see different ads. Thus the game could display different ads for different players even in a shared environment. This opens the door to targeted advertising. Advertisers can bid differently for different player profiles. Accurate statistics about players' in-game activities are already maintained by game servers in order to track players' progress. These statistic can be used to get a picture of the type of player. For example, Bartle types [3] propose four prototypes: explorers, killers, achievers, and socializers. A player is given scores, called Bartle quotients, in each of the four types. These Bartle quotients can be used by advertisers to distinguish target market segments. For example, advertisers selling fiction books might bid higher for their ad to be displayed to players who have high explorer quotients. Even more useful, MMOGs contain well-defined social networks such as guilds (large groups of players who share similar goals or virtual professions), parties (smaller groups of players who go on quests together), and personal contact lists (other players who are friends of a player and who often socialize with the player in the game). These social networks can be mined to predict whether or not a new player falls into a certain market segment based on whether or not his friends and fellow guild-members do. Furthermore, all in-game chat is logged and so can be mined for keywords that indicate potential interest in the products that the advertisers sell. Of course, the extent to which the social networks and chat transcripts can be used to build the player profile for advertisers is subject to the privacy policies of the game. However, one could well imagine ad-supported versions of the game that allow players to play for free provided they agree to sharing their in-game social data with advertisers.

Prominence. One of the most important sources of value of virtual billboard advertising to an advertiser comes from impressions. The location of the billboard within the virtual world can affect the amount of impact its ads can have. Billboards places at eye-level within the gameworld tend to have greater impact. Too many billboards cluttered together can reduce the amount of impression that an ad makes. Beyond spatial positioning of the billboard, gameplay-related distractions present at the location can also affect impressions. For example, if a billboard is placed in an area where there is a lot of intense and immersive gameplay (such as combat with a monster), then the player is not likely to pay much attention to the ad displayed on that billboard. Thus, we propose to assign each billboard a *prominence score*, based on the visibility of its spatial location and on the amount of distractions present at the location (e.g., other billboards, enemies, etc.). The prominence score can be calculated just before the auction for a billboard (or set of billboards) begins based on the number of enemies near the billboard at the time the player enters a trigger area, and based on the visibility the billboard would have for a player approaching from the trigger area.

Exposure. Even if an ad is placed on a prominent billboard with few distractions around it, the player may still not see the ad because he just happened to be facing the wrong way. In determining the amount of exposure an ad has to a player, games have a great advantage over web search. It is very easy to accurately measure and record various properties that directly affect an ad's exposure. For example, one

can measure how long the ad is in the player's field of vision, whether or not the player's view of the ad was obstructed by another object, what angle the ad was viewed from, whether the player was engaged in some other activity (e.g., cycling through his inventory) while the ad was in view, etc. We propose to combine these measurements into a single exposure score that is accumulated over the course of the game. Note that the exposure score is known only after the player has quit the game and is therefore uncertain at the time of the ad auction.

Engagement. Beyond measuring the exposure that virtual billboard ads provide, one may be tempted to implement a mechanism analogous to clicks in sponsored search auctions take the player to the advertiser's homepage in a separate browser window. However, care must be taken so as to minimize the intrusiveness of such a mechanism on the gameplay. The mechanism should allow players to express interest in an ad, but should not entail a substantial distraction from the immediacy of gameplay. We propose to use the aiming/targeting system already built into these games for such a non-intrusive mechanism. The idea is to allow players to *bookmark* an ad by "shooting" at the ad. Player can "shoot" ads more or less to indicate how much the ad interests them. Upon quitting the game, the player is then presented with a splash screen containing the list of all the ads he bookmarked, sorted by the amount of he engaged the ad by shooting it.

Adapting our auction framework. In this new setting of advertising on in-game billboards, the billboards are analogous to slots. The exposure and engagement scores are similar to clicks and purchases in that they are unknown at auction time, and therefore the game must maintain distributions of exposure and engagement scores for each billboard. These distributions can be based on historical data. An auction for a set of billboards is run when a player enters their trigger area.

As before, advertisers submit programs to bid on their behalf, and these programs are given access to variables relevant to the player's profile (e.g., Bartle types, guild, etc.) and to the current prominence scores for the set of billboards. Now, instead of of bidding on slots, clicks, and purchases, these programs output bids on billboards as well as on intervals of exposure and engagement scores. For examples, an advertiser can bid 3 cents for the second-most prominent billboard if exposure ends up being greater than 0.8 and engagement ends up being greater than 0.6. This would be represent by a bids table as shown in Figure 13.

formula	value
...	...
$\text{Billboard}_2 \wedge \text{Exp}_{(0.8,1]} \wedge \text{Eng}_{(0.6,1]}$	3
...	...

Figure 13. Bids table

The game then computes winners so as to maximize the expected revenue assuming that advertisers pay what they bid. We assume once again that the only billboard that affects the exposure and engagement scores for an advertiser is the billboard to which he is assigned. Then we can use our algorithm from Section 3 to solve winner determination efficiently. Moreover, our techniques for reducing program evaluation from Section 2.2 still apply in this setting.

7. Conclusion

Our paper is a first step toward applying database principles to the exciting and important problems arising in advertising auctions. In this paper, we highlight the need for more expressive bidding in sponsored search auctions. To address this, we propose a framework that empowers advertisers with an expressive bidding language, and we provide efficient, scalable, and parallelizable techniques for performing winner determination given bids expressed in our language. We also described how our techniques could be adapted to advertising in massively multiplayer online games. We believe that the database community has much to offer this area given its vast experience with the trade-offs between expressiveness and scalability; and providing advertisers with more expressive bidding languages while retaining the scalability of these sponsored search auctions is crucial to the continued growth of this multi-billion dollar industry.

References

[1] G. Aggarwal, A. Goel, and R. Motwani. Truthful auctions for pricing search keywords. In *EC '06: Proceedings of the 7th ACM Conference on Electronic Commerce*, pages 1–7. ACM Press, 2006.

[2] G. Aggarwal, S. Muthukrishnan, and J. Feldman. Bidding to the top: VCG and equilibria of position-based auctions. In *WGOA '06: Proceedings of the 4th Workshop on Approximation and Online Algorithms*, September 2006.

[3] R. A. Bartle. Hearts, clubs, diamonds, spades: Players who suit MUDs. In *Comms Plus!*, October/November 1990. Available at http://www.mud.co.uk/richard/hcds.htm.

[4] M. S. Borella, A. Sears, and J. A. Jacko. The effects of Internet latency on user perception of information-content. In *GLOBECOM '97: Proceedings of 40th annual IEEE Global Telecommunications Conference*, volume 3, pages 1932–1936. IEEE Computer Society, November 1997.

[5] C. Borgs, J. Chayes, O. Etesami, N. Immorlica, K. Jain, and M. Mahdian. Dynamics of bid optimization in online advertisement auctions. In *WWW '07: Proceedings of the 16th International World Wide Web Conference*, 2007.

[6] V. Chvátal. On certain polytopes associated with graphs. *Journal on Combinatorial Theory Series B*, 13:138–154, 1975.

[7] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[8] B. G. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. NBER Working Paper No. W11765, November 2005.

[9] eMarketer. The unstoppable surge of search advertising. http://www.emarketer.com/Article.aspx?1004811, April 2007.

[10] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS '01: Proceedings of the 20th ACM Symposium on Principles of Database Systems*, pages 102–113. ACM Press, New York, NY, USA, 2001.

[11] M. Fayyazi, D. Kaeli, and W. Meleis. An adjustable linear time parallel algorithm for maximum weight bipartite matching. *Information Processing Letters*, 97(5):186–190, 2006.

[12] K. Graft. Study: In-game ads actually work. Next Generation, July 2007. Available at http://www.next-gen.biz/index.php?option=com_content&task=view&id=6533&Itemid=2.

[13] T. Groves. Incentive in teams. *Econometrica*, 41:617–631, 1973.

[14] V. Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, Royal Institute of Technology, Stockholm, 1992.

[15] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.
[16] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics*, 2:83–97, 1955.
[17] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society of Industrial and Applied Mathematics*, 5(1):32–38, March 1957.
[18] A. Newman. The maximum acyclic subgraph problem and degree-3 graphs. In *APPROX '01/RANDOM '01: Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems and 5th International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 147–158. Springer-Verlag, 2001.
[19] Nielsen//NetRatings. Interactive advertising bureau (IAB) search branding study. Commissioned by the IAB Search Engine Committee, August 2004. Available at http://www.iab.net/resources/iab_searchbrand.asp.
[20] M. Shields. In-game ads could reach 2 bil. Adweek, April 2006. Available at http://www.adweek.com/aw/national/article_display.jsp?vnu_content_id=1002343563.
[21] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 61–70, New York, NY, USA, 1991. ACM Press.
[22] H. R. Varian. Position auctions. UC Berkeley Working Paper, 2006.
[23] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

A. Proofs

Proof of Theorem 2 Consider any bid of \$d on event E where E is a 1-dependent event which depends on the slot assigned to only one advertiser, say i. If advertisers pay what they bid, then in all outcomes this bid contributes exactly the same amount to the revenue as the OR-bid of \$d on $E \wedge \mathrm{Slot}_1^i$, \$d on $E \wedge \mathrm{Slot}_2^i$, ..., \$d on $E \wedge \mathrm{Slot}_k^i$, and \$d on $E \wedge (\wedge_j \neg \mathrm{Slot}_j^i)$, where Slot_j^i is the event that advertiser i gets slot j. This is because $\mathrm{Slot}_1^i, \ldots, \mathrm{Slot}_k^i$ are mutually exclusive events since the allocations are restricted to at most one slot per advertiser. We can thus fill out a table of advertisers versus slots where the entry for the ith advertiser and the jth slot is the sum of the total expected revenue from bids on events of form $E \wedge \mathrm{Slot}_j^i$ assuming advertisers pay what they bid. If we interpret this table as the edge-weight matrix of a bipartite graph between advertisers and slots, then the winner determination problem is the problem of finding a maximum-weight bipartite matching for this graph, which can be done in polynomial time [16]. □

Proof of Theorem 3 We will reduce the winner determination problem to the maximum-weighted feedback arc set problem by using bids on 2-dependent events to encode the edges in a given weighted directed graphs on advertisers. Consider any weighted directed graph on n advertisers. Let $w_{i,i'}$ be the weight of the edge from advertiser i to advertiser i'. Let Slot_j^i be the event that advertiser i gets assigned slot j. For two advertisers i and i', let $E_{i>i'}$ be shorthand for $\vee_j(\mathrm{Slot}_j^i \wedge ((\vee_{j'>j}\mathrm{Slot}_{j'}^{i'}) \vee (\wedge_{j'}\neg\mathrm{Slot}_{j'}^{i'}))$, which is the event that advertiser i gets a slot and is placed above advertiser i' who may or may not get a slot. Then $E_{i>i'}$ is a 2-dependent event since it depends on the slots assigned to advertisers i and i'. Let each advertiser i place the following bids: for each $i' \neq i$, bid $w_{i,i'}$ on $E_{i>i'}$. Then, assuming advertisers pay what they bid, revenue of $w_{i,i'}$ will be generated if and only if advertiser i is placed above advertiser i'. Then winner determination is equivalent to the problem of finding the maximum-weighted feedback arc set over all size-k subgraphs, which is APX-hard in n and k [14]. In fact, even when the directed

graphs are restricted to degree-3, the feedback arc set problem is still NP-hard [18, 15]. This means that winner determination is NP-hard even when each bid is restricted to the events mentioning at most three advertisers. So it is infeasible to allow advertisers to bid on being placed above even two or more competitors. □