Sponsored Search Auctions with Conflict Constraints

Panagiotis Papadimitriou
Stanford University
Stanford, CA, USA
papadimitriou@stanford.edu

Hector Garcia-Molina
Stanford University
Stanford, CA, USA
hector@cs.stanford.edu

ABSTRACT

In sponsored search auctions (SSA) advertisers compete for ad slots in the search engine results page, by bidding on keywords of interest. To improve advertiser expressiveness, we augment the bidding process with *conflict* constraints. With such constraints, advertisers can condition their bids on the non-appearance of certain undesired ads on the results page. We study the complexity of the *allocation* problem in these augmented SSA and we introduce an algorithm that can efficiently allocate the ad slots to advertisers. We evaluate the algorithm run time in simulated conflict scenarios and we study the implications of the conflict constraints on search engine revenue. Our results show that the allocation problem can be solved within few tens of milliseconds and that the adoption of conflict constraints can potentially increase search engine revenue.

1. INTRODUCTION

In sponsored search auctions advertisers compete for m ad slots next to the organic search results. The participation of each advertiser in the auction is determined by the relevance of the submitted query to the advertiser interests. These interests are expressed through keywords that should match the submitted queries. For example, an advertiser with an online movies web site may specify that he is interested in queries that contain the word "movie". The advertiser also provides a *bid*, the maximum amount the he is willing to pay for a click on his ad.

Although this auction model is simple, it lacks some necessary expressiveness. For example, there is no way for an advertiser to express that his value for a click, and consequently his bid, depends on the identity of other advertisers that appear in the m slots. In the movie advertiser example, he may not want his ad to appear when other slots advertise adult movies. Similarly, the advertiser may not want his ad to appear when certain competitors appear.

The improve expressiveness in SSA, we allow an advertiser to specify, in addition to keywords of interest, a set of *conflict* advertisers, i.e., advertisers that should not appear in the top m slots when his ad is displayed. The advertiser can express a different valuation for a click if his conflict constraints are satisfied and a different bid if his conflict constraints are violated.

What makes the adoption of conflicts attractive is that they can benefit all parties of sponsored search. The advertisers who declare conflicts can more effectively control the context where their ads get clicked and avoid the cost of "undesired" advertising. The search engines can potentially increase their revenues from the new service (as we will show) and users will enjoy the benefits of improved ad targeting. The only concern about the support of conflicts comes from its computational cost. However, we believe that this papers addresses effectively this concern.

The paper is organized as follows. In Section 2 we introduce our formal model and describe the problems that need to be solved to support conflict constraints: (a) the *allocation problem*, i.e., how will the ad slots be allocated to advertisers in the new auction mechanism, and (b) the *payment problem*, i.e., how will advertisers be charged. Then, in Section 3 we provide two solutions to the allocation problem. The first solution uses a depth-first-search (DFS) branch-and-bound algorithm and the second solution (just summarized due to space constraints) uses the combinatorial auctions framework. In the same section we provide a solution for the payment problem using the *Vickrey-Clarke-Groves* mechanism. In Section 4 we evaluate the run time performance of the proposed solutions and we quantify the impact of conflicts on the search engine revenue. We present the related work in Section 5 and we conclude in Section 6.

2. PROBLEM DEFINITION

In a sponsored search auction the *search engine* is the auctioneer, the *advertisers* are the bidders and the ad *slots* are the auctioned items.

Let $A = \{a_1, \ldots, a_j, \ldots, a_n\}$ denote the set of n advertisers that participate in a sponsored search auction. Without loss of generality we assume that each advertiser participates in the auction with a single ad. Thus, in the rest of the paper we will use the terms "advertiser" and "ad" interchangeably. The advertisers participate in the auction to get their ads clicked by the users. Advertiser a_j participates in the auction with:

- the private valuation v_j for a click;
- the bid b_j, i.e., the maximum that he is willing to pay for a click; and

- the set of *conflict* advertisers $C_j \subseteq A$. Advertiser a_j's bid is conditioned upon the constraint that none of his conflict advertisers C_j wins a slot.

Note that in our model each advertiser specifies his conflicts explicitly. Although this convention simplifies the presentation, it does not restrict the generality of our model. In practice, we expect advertisers to specify their conflicts by describing conflict advertiser properties. For example, an advertiser can declare conflicts against advertisers that use certain keywords in their ad messages or against advertisers whose landing pages belong to a certain domain. In any case the advertiser preferences can eventually be translated to specific conflicts at the auction time.

Although the advertisers are interested in user clicks, the search engine cannot auction such clicks directly. The search engine auctions instead m ad slots $S = \{s_1, \ldots, s_i, \ldots, s_m\}$, where the ads can be displayed and potentially get clicked by users. These slots are not identical, because each slot s_i has its own click probability ρ_i. The click probabilities are estimated through historic click through rates (CTRs) and they are presumed to drop as the slot index i increases [9], i.e., $\rho_i \geq \rho_{i+1}, \forall s_i \in S$. Note that the click probability of an ad does not depend only on the hosting slot but also on the ad itself. However, the most popular search engines such as Google and Bing take the advertiser impact into account through a multiplicative factor that is applied to the advertiser bid. We follow this convention and in the rest of the paper we assume ad independent click probabilities. However, the algorithms that we present could be applied to ad-dependent click probabilities with minor changes.

Given the click valuation v_j and the click probabilities ρ_i, the expected *utility* of advertiser a_j who wins slot s_i is $v_j \rho_i$. If advertiser j does not win any slot, then his utility is 0.

The auctioneer, i.e., the search engine, has to allocate the auctioned slots to the advertisers and charge them accordingly. Thus, the auctioneer needs to define an *allocation rule* and a *payment rule*. In the following paragraph we discuss the desired properties of these rules.

The objective of the allocation rule is to maximize the *social welfare* [12], i.e., to maximize the sum of the advertisers' utilities. If the binary variable $x_{ij} \in \{0, 1\}$ indicates whether advertiser a_j wins slot s_i, then the social welfare is:

$$\sum_{j=1}^{n} \sum_{i=1}^{m} v_j \rho_i x_{ij}. \tag{1}$$

Apart from maximizing the social welfare, the allocation rule must also satisfy a series of constraints that we show in the following Integer Linear Program (ILP):

$$\begin{aligned}
\text{max.} \quad & \sum_{j=1}^{n} \sum_{i=1}^{m} v_j \rho_i x_{ij} && \text{(2a)}\\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} \leq 1, \qquad \forall s_i \in S && \text{(2b)}\\
& \sum_{i=1}^{m} x_{ij} \leq 1, \qquad \forall a_j \in A && \text{(2c)}\\
& \sum_{i=1}^{m} (x_{ij} + x_{ik}) \leq 1, \quad \forall a_j \in A \wedge a_k \in C_j && \text{(2d)}\\
& x_{ij} \in \{0, 1\}
\end{aligned}$$

We refer to this ILP as the *allocation problem*. Note that the objective contains the secret valuations v_j instead of the known bids b_j, since the search engine desires to maximize the actual welfare. With the appropriate allocations scheme (to be discussed), advertisers will use their v_j values as their bids. Therefore, we will assume that the b_j are indeed the v_j values, and we can use the known b_j values to compute utility in our allocation scheme.

The constraints of Equation 2b guarantee that every slot is allocated to at most one advertiser. The constraints of Equation 2c guarantee that every advertiser wins at most one slot. Finally, the constraints of Equation 2d guarantee that the conflict constraints of the advertisers are also satisfied. A *feasible allocation* is an allocation of slots to advertisers that satisfies all the constraints of the ILP. The *optimal allocation* x^* is a feasible allocation that maximizes the objective.

For example, say that there are three advertisers $A = \{a_1, a_2, a_3\}$ with conflicts $C_1 = C_3 = \emptyset$ and $C_2 = \{a_1\}$ and two ad slots. If we denote the allocation of advertisers to slots as an ordered advertiser list, the allocation $[a_1]$, i.e., advertiser a_1 wins the first slot, or the allocation $[a_2, a_3]$, i.e., advertiser a_2 wins the first slot and advertiser a_3 wins the second slot, are feasible allocations. However, the allocation $[a_1, a_2]$ is not feasible because the advertiser a_2 has a conflict with advertiser a_1 and the constraint of Equation 2d is violated.

Note that in our formulation each advertiser bids only upon the condition that his conflict constraints are satisfied. However, an advertiser may want to specify two different bids: one for the case that his conflict constraints are satisfied and a different one for the case that his constraints are violated. Althouh we do not dicuss such bids in the rest of the paper, our formulation can support them in the following way. For advertiser a_j that wants to submit two bids we create a virtual advertiser a'_j and we make sure that $a_j \in C'_j$ and $a'_j \in C_j$ (so that the advertiser does not win two slots). Then, the advertiser can submit two different bids with different conflict sets.

Given the allocations $x = \{x_{ij}\}$ and the bid vector $b = \{b_j\}$, the payment rule defines the price $p_j(b, x)$ that advertiser j has to pay. A desired property of the payment rule is to motivate *truthful bidding*, i.e., to motivate advertisers to report their real click valuations ($b_j = v_j$). An advertiser has such an incentive, if his *payoff* is maximized when $b_j = v_j$, regardless of the other advertisers' valuations, and regardless of how they decide to bid [12]. The payoff of an advertiser is defined as his utility minus his payment: $\sum_{i=1}^{m} v_j \rho_i x_{ij} - p_j(b, x)$. Thus, if $b, \hat{b}$ are two bid vectors that differ only in the j-th position with $b_j = v_j$ and $\hat{b}_j \neq v_j$, and x^*, $\hat{x}^*$ are the respective allocations by some allocation rule, then the payment rule ensures truthful bidding if and only if for all $b_1, \ldots, b_{j-1}, b_{j+1}, \ldots, b_n$:

$$\sum_{i=1}^{m} v_j \rho_{ij} x^*_{ij} - p_j(b, x^*) \geq \sum_{i=1}^{m} v_j \rho_{ij} \hat{x}^*_{ij} - p_j(\hat{b}, \hat{x}^*).$$

Without truthful bidding the advertisers have incentive to report untrue evaluations that may result in inefficient allocations [12, 5].

In the next sections of the paper we address the following questions: (a) how can we efficiently solve the allocation problem, (b) which payment rule ensures truthful bidding, and (c) how are the advertiser utilities and consequently the search engine revenue affected by the ability to declare conflicts.

3. SSA WITH CONFLICTS

In this section we study the allocation and the payment rules in SSA with conflict constraints. In Section 3.1 we provide a proof that the allocation problem is NP-hard to solve. Then, in Section 3.2 we present Debbassac, a DFS algorithm that can solve the allocation problem efficiently by reducing the search space with branch-and-bound pruning. In Section 3.3 we provide an alternative formulation of the allocation problem in the context of combinatorial auctions. Then, in Section 3.4 we present a payment rule that ensures truthful bidding. Finally, in Section 3.5 we study the implications of the ability to specify conflicts on the advertiser utilities and bids.

Throughout the section we use the following auction as an example: there are four advertisers $A = \{a_1, a_2, a_3, a_4\}$ with click valuations $v_1 = \$10$, $v_2 = \$8$, $v_3 = \$5$ and $v_4 = \$2$ that have the following conflicts: $C_1 = C_4 = \emptyset$ and $C_2 = C_3 = \{a_1\}$. The auctioned slots are $S = \{s_1, s_2\}$ and their click probabilities are $\rho_1 = 0.2$ and $\rho_2 = 0.1$.

3.1 Problem hardness

We represent the advertiser conflicts and valuations with the weighted *undirected conflict graph* $G = (V, E)$. Each node $a_j \in V$ corresponds to advertiser a_j and weight of the node is equal to the advertiser click valuation v_j. The graph has an edge (a_j, a_k) if $a_k \in C_j$ or $a_j \in C_k$. Note that the graph is undirected, since no matter whether $a_k \in C_j$ or $a_j \in C_k$, the advertisers a_j and a_k cannot appear in the ad slots together.

In our NP-hardness proof we use a reduction from the *Maximum Weight Independent Set* or MWIS problem. An *independent set* is set of nodes in a graph, no two of which are adjacent. The MWIS in a weighted graph is the independent set with the maximum sum of node weights. The problem of finding the MWIS in a weighted graph is NP-hard.

THEOREM 1. *It is NP-hard to solve the allocation problem of Equation 2.*

PROOF. Sketch: Use reduction from the MWIS. Take any instance of the MWIS problem on a graph $G = (V, E)$ and create an allocation problem with one advertiser per node of V and one conflict per edge of E. Then say that there are $m = |V|$ slots with equal click probabilities $\rho_1 = \rho_2 = \cdots = \rho_n$. The set of nodes that correspond to the advertisers of the optimal allocation is the MWIS of graph G. □

3.2 Debbassac algorithm

A brute-force algorithm for the allocation problem is the following:

1. Enumerate all the possible allocations of advertisers to ad slots. Note that we can consider only the allocations that either fill all the slots or they leave some bottom slots empty. Allocations that leave a top slot empty can be discarded, since we can always shift the advertisers up-wards to fill the slot and improve the social welfare. The number of possible allocations is $n + \binom{n}{2} + \cdots + \binom{n}{m}$, since n is the number of allocations that fill just the top slot, $\binom{n}{2}$ is the number of allocations that fill the two top slots, and so on.
2. Remove allocations that are not feasible, i.e., allocations that contain advertisers with conflicts among them.
3. Find among the remaining allocations the one that yields the maximum social welfare.

Algorithm 1 Debbassac: **De**pth-first-search **b**ranch-and-**b**ound **a**lgorithm for **s**ponsored **s**earch **a**uctions with **c**onflict constraints.

```
Input: set of advertisers $A$ with conflict set $C_j$ and valuations
       $v_j, \forall a_j \in A$, slots $S$ with click probabilities $\rho_i, \forall s_i \in S$.
Output: optimal allocation $x^*$ and social welfare $w^*$.

 1: Global variable $w^* \leftarrow 0$                      ▷ Optimal welfare
 2: Global variable $X^* \leftarrow \emptyset$   ▷ Sparse representation of the
    optimal allocation
 3: AllocRem($\emptyset, 0, A$)
 4: $x^*_{ij} \leftarrow 0, \forall a_j \in A, s_i \in S$
 5: for all $(s_i, a_j) \in X^*$ do
 6:     $x^*_{ij} \leftarrow 1$
 7: end for
 8: return $x^*, w^*$

 1: procedure AllocRem($X_{in}, w_{in}, A_{rem}$)
    Input invariants: $\{a_j \mid (s_i, a_j) \in X_{in}\} \cap A_{rem} = \emptyset$
 2:     $i \leftarrow |X_{in}| + 1$
                                      ▷ Check for the end of recursion.
 3:     if $i = m + 1 \vee A_{rem} = \emptyset$ then
 4:         if $w_{in} > w^*$ then
 5:             $w^* \leftarrow w_{in}$
 6:             $X^* \leftarrow X_{in}$
 7:         end if
 8:         return
 9:     end if
                        ▷ Calculate $w_j$, $h_j$ and $A_{j,rem}$ for $a_j \in A_{rem}$.
10:     for all $a_j \in A_{rem}$ do        ▷ $A_{rem}$ sorted by $v_j$ desc.
11:         $w_j \leftarrow w_{in} + \rho_i v_j$
12:         $C'_j \leftarrow \{a_k \mid a_k \in A_{rem} \wedge a_j \in C_k\}$
13:         $L_j \leftarrow \{a_k \mid a_k \in A_{rem} \wedge v_k > v_j\}$
14:         $A_{j,rem} \leftarrow A_{rem} - (\{a_j\} \cup C_j \cup C'_j \cup L_j)$
15:         $h_j \leftarrow$ UpperBound($m - i, A_{j,rem}$)
16:         if $(C_j \cup C'_j) \cap A_{rem} = \emptyset$ then
17:             break
18:         end if
19:     end for
                                  ▷ Explore allocations recursively.
20:     Remove un-examined advertisers from $A_{rem}$.
21:     Sort $A_{rem}$ by $w_j + h_j$ desc.
22:     for all $a_j \in A_{rem}$ do
23:         if $w_j + h_j \leq w^*$ then          ▷ Pruning condition
24:             break
25:         else
26:             AllocRem($X_{in} \cup \{(s_i, a_j)\}, w_j, A_{j,rem}$)
27:         end if
28:     end for
29: end procedure

 1: function UpperBound($m_{rem}, A_{rem}$)
 2:     $h \leftarrow 0$
 3:     while $m_{rem} > 0 \wedge A_{rem} \neq \emptyset$ do
 4:         $i \leftarrow m - m_{rem} + 1$
 5:         $a_j \leftarrow \arg\max_{a_k \in A_{rem}} v_k$
 6:         $h \leftarrow h + \rho_i v_j$
 7:         $m_{rem} \leftarrow m_{rem} - 1$
 8:         $A_{rem} \leftarrow A_{rem} - \{a_j\}$
 9:     end while
10:     return h
11: end function
```

Such an algorithm is exhaustive and it finds the optimal allocation. However, the algorithm is expensive, since it examines all possible allocations.

We can substantially improve upon the brute-force algorithm with a more "careful" search in the space of possible advertiser allocations. In the Debbassac algorithm that we present in Algorithm 1 we show (a) how we can restrict the search within the space of feasible allocations, and (b) how we can prune allocations that have no way beating allocations that have already been explored.

The algorithm input consists of the advertisers A (along with their click valuations and their conflict sets) and the ad slots S (along with their click probabilities). The algorithm explores the space of feasible allocations in a DFS fashion and uses a branch-and-bound approach to prune the search space. The algorithm returns the optimal allocation x^* and the corresponding social welfare w^*. In the pseudo-code we also use variable X^* as a sparse representation of the optimal allocation. X^* is a set of pairs such as (s_i, a_j) that indicates that advertiser a_j wins the slot s_i.

The search in the allocation space is performed in the procedure $\text{ALLOCREM}(X_{in}, w_{in}, A_{rem})$. The input of the procedure is the allocation of advertisers to the first $i-1$ slots X_{in}, the corresponding social welfare so far w_{in} and the set of advertisers A_{rem} that can fill the remaining $m-(i-1)$ slots. The procedure tries to fill the i-th slot with one of the A_{rem} advertisers and then it calls itself recursively to fill the remaining $m-i$ slots. The procedure does not have a return value, but it updates the global variables w^* and X^* when it finds an allocation that yields higher social welfare than the current maximum.

In lines 2-9 the procedure checks whether the input allocation is *complete* and compares its social welfare with the current maximum. The allocation is complete if either all m slots are filled or there are no advertisers left ($A_{rem} = \emptyset$) to fill the remaining slots. If the input social welfare w_{in} is higher than the current maximum, then global variables w^* and X^* are updated in line 5-6. The procedure stops in line 8.

If the input allocation is not complete, then the algorithm evaluates the allocations that augment X_{in} with an advertiser $a_j \in A_{rem}$ in the slot s_i (lines 10-19). In particular, for each such allocation the algorithm calculates (a) the new social welfare w_j, (b) the advertisers $A_{j,rem}$ that will be considered to fill the remaining $m-i$ slots, and (c) an upper bound h_j of the social welfare that the remaining slots can contribute. The calculation for w_j in (a) is straight-forward. For $A_{j,rem}$ in (b) the algorithm removes from A_{rem} the advertiser a_j himself, his conflicts C_j, the advertisers C'_j that have a_j in their conflicts, and the advertisers L_j that have a higher valuation than a_j. The first three removals ensure that the algorithm will explore only feasible allocations while trying to fill the remaining $m-i$ slots. The L_j advertisers are removed, because we do not need to consider an advertiser a_k with higher valuation than a_j for the remaining slots. At some other point in search the algorithm will consider a scenario with a_j and a_k roles reversed and that alternative scenario dominates the one where a_j wins slot s_i and a_k wins a lower slot s'_i with $i < i'$. To obtain an upper bound in (c) we can assume that the advertisers with the top $m-i$ click valuations in $A_{j,rem}$ will win the remaining $m-i$ slots. This calculation yields an upper bound, because it ignores possible conflicts within these $m-i$ advertisers.

The algorithm stops the loop execution in lines 16-18 if the current advertiser a_j has no conflicts with the remaining advertisers. The lack of conflicts ensures that no restriction is imposed on the allocation of the remaining slots. Moreover, since the advertisers are examined in descending order of click valuation, the allocation of slot s_i to a_j dominates the allocation of this slot to any of the un-examined advertisers.

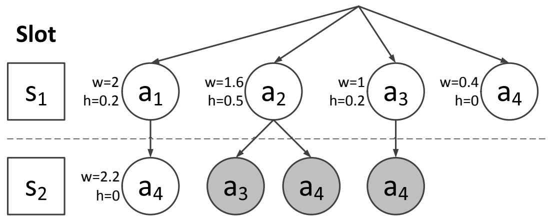


Figure 1: The search space that corresponds to the running example.

In lines 20-28, the algorithm attempts to allocate slot s_i to advertiser a_j and allocate the rest of the slots in a recursive fashion. First, the algorithm sorts the advertisers A_{rem} by the sum $w_j + h_j$ to explore the most promising allocations first. Note that the sum $w_j + h_j$ provides a social welfare upper bound of any complete allocation with the advertiser a_j in the i-th slot and the first $i-1$ slot allocated as in X_{in}. In lines 22-28 the procedure calls itself to explore the allocation with the slot s_i to advertiser a_j, only if such an allocation can yield a higher social welfare than the current maximum (line 23). If this condition does not hold the procedure breaks the loop and returns. Note that it does not need to consider the remaining advertisers, since they are sorted in decreasing values of $w_j + h_j$. If the condition holds, the procedure calls recursively itself with an updated input allocation $X_{in} \cup \{(s_i, a_j)\}$, social welfare w_j and remaining advertisers $A_{j,rem}$.

In Figure 3.2 we illustrate how the algorithm would be applied in the auction of our running example. Each path in the figure (from root to leaf) represents a possible allocation, e.g., the $a_1 \rightarrow a_4$ path represents the allocation of the slot s_1 to a_1 and the slot s_2 to a_4. Note that the algorithm considers all four advertisers as candidates to fill the first slot. If advertiser a_1 is selected, then the algorithm will consider only advertiser a_4 for the second slot, because advertisers a_2 and a_3 have a_1 in their conflict sets. If advertiser a_2 is selected, then the algorithm will consider only a_3 and a_4 for the second slot, since a_2 has advertiser a_1 in his conflict set. If the advertiser a_3 is selected, then the algorithm will consider only a_4 for the second slot, since a_4 has advertiser a_1 in his conflict set and a_2 has a higher click valuation than a_3. Finally, if advertiser a_4 is selected there will be no advertiser to consider for the second slot, since the rest of the advertisers have higher click valuations than him. The algorithm also calculates the w and h values for each node. For example, if advertiser a_1 is selected for the first slot, then $w = v_1\rho_1 = \$10 \times 0.2 = \$2$ and $h = v_4\rho_2 = \$2 \times 0.1 = \$0.2$. Based on the social welfare upper bound $w + h$, the algorithm will choose to explore first the allocation with advertiser a_1 in the first slot. Then the algorithm allocates the second slot to advertiser a_4 and updates the maximum welfare $w^* = \$2.2$. Since none of the upper bound values is higher than \$2.2 the algorithm will terminate and it will never explore the gray nodes of the search tree.

3.3 Combinatorial auction formulation

In this section we describe how to formulate the sponsored search auctions with conflicts in the combinatorial auctions framework [3]. A bidder in a combinatorial auction can place his bid on a *bundle*, i.e., a combination of the auctioned items, and he wins either all or none of the bundle items. For example, if the auctioned items are $\{\alpha, \beta, \gamma\}$ then a bidder can place a bid on the bundle $\{\alpha, \beta\}$ and another bidder on the bundle $\{\beta, \gamma\}$. Since the two bundles have item β in common, only one of the bidders can win. If the first bidder wins, then the item γ will be unassigned and if the second bidder wins, then the item α will be unassigned.

We use the notion of shared items to represent the conflicts between advertisers. In addition to the real ad slots S we introduce the set of *dummy* slots

$$D = \{d_{jk} \mid a_j \in A \wedge (a_k \in C_j \vee a_j \in C_k) \wedge j < k\}. \quad (3)$$

The dummy slot d_{jk} corresponds to the conflict between advertisers a_j and a_k, i.e., a_j is in the conflict set of a_j or a_j is in the conflict set of a_k. Note that the inequality $j < k$ in the set definition prevents the creation of two dummy slots d_{jk} and d_{kj} for the conflict between advertisers j and k. Instead, one of the two slots is created. To illustrate, in our running example we need to create two dummy slots to respresent the advertiser conflicts: d_{12} for the conflict of advertiser a_2 with a_1, and d_{13} for the conflict of advertiser a_3 with a_1. The set of slots R that the search engine auctions includes both the real slots S and the dummy slots D:

$$R = S \cup D. \quad (4)$$

Given the augmented set of slots, we can convert the advertiser bids and the real slots and their conflicts constraints to equivalent bids on bundles of R slots. In particular, we can replace the bids and the conflicts of advertiser a_j with the following m bundle bids:

$$B_{ji} = \{s_i\} \cup D_j \qquad i = 1, \ldots, m, \quad (5)$$

where D_j is the set of dummy slots that correspond to conflicts that involve a_j:

$$D_j = \{d_{jk} \mid a_k \in A \wedge d_{jk} \in D\} \cup \{d_{kj} \mid a_k \in A \wedge d_{kj} \in D\}.$$

The bundles that the four advertiser bid on in our running example are the following:

$$\begin{array}{ll} B_{11} = \{s_1, d_{12}, d_{13}\}, & B_{12} = \{s_2, d_{12}, d_{13}\}, \\ B_{21} = \{s_1, d_{12}\}, & B_{22} = \{s_2, d_{12}\}, \\ B_{31} = \{s_1, d_{13}\}, & B_{32} = \{s_2, d_{13}\}, \\ B_{41} = \{s_1\}, & B_{22} = \{s_2\}. \end{array}$$

Note that each bundle B_{ji} contains the real slot s_i and the dummy slots D_j. Thus, if there is a conflict between advertisers a_j and a_k and advertiser a_j wins some bundle, say B_{ji}, then advertiser a_j does not only win the slot s_i, but also he wins the the dummy slot d_{jk}. Slot d_{jk} belongs to both D_j and D_k. Since all of the bundles of advertiser a_k include the dummy slot d_{jk}, advertiser a_k cannot win any bundle, and consequently, he cannot win any real slot.

The advertiser valuation for the bundle B_{ji} is equal to the expected valuation from winning the real slot s_i:

$$v_j(B_{ji}) = v_j \rho_i \quad (6)$$

Recall that ρ_i is the click probability of slot s_i and v_j is the click valuation of advertiser a_j. Note that the advertiser does not value the dummy slots of the bundle. However, these slots are essential for the satisfaction of his and other advertisers' conflict constraints. The asvertiser bundle valuations in our running example are the following:

$$\begin{array}{ll} v_1(B_{11}) = \$20, & v_1(B_{12}) = \$10, \\ v_2(B_{21}) = \$16, & v_2(B_{22}) = \$8, \\ v_3(B_{31}) = \$10, & v_3(B_{32}) = \$5, \\ v_4(B_{41}) = \$2, & v_4(B_{42}) = \$1. \end{array}$$

Note that all of the advertisers have zero valuation for any other bundle that is not listed above.

If the binary variable $x_j(B) \in \{0, 1\}$ denotes whether advertiser j wins the bundle B, then the allocation rule that maximizes social welfare must solve the following ILP:

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_{j=1}^{n}\sum_{i=1}^{m} v_j(B_{ji}) x_j(B_{ji}) & \quad (7a) \\ \text{subject to} & \displaystyle\sum_{j=1}^{n}\sum_{B \in \mathcal{B}, r \in B} x_j(B) \leq 1, \qquad \forall r \in R & \quad (7b) \\ & \displaystyle\sum_{i=1}^{m} x_j(B_{ji}) \leq 1, \qquad \forall a_j \in A & \quad (7c) \\ & x_j(B) \in \{0, 1\} & \end{array}$$

where set

$$\mathcal{B} = \{B_{ji} \mid a_j \in A \wedge s_i \in S\}$$

contains all the slot bundles that the advertisers bid on. The objective of the problem is the social welfare defined in terms of the auctioned bundles. The sum has nm term, since each of the n advertisers bids on m bundles. The constraints of Equation 7b guarantee that each slot $r \in R$ (real or dummy) can belong to at most one of the bundles that are allocated to advertisers. In other words, the bundles that are allocated to advertisers cannot include shared items The number of such constraints is $m + |D|$, where $|D|$ is equal to the number of advertiser conflicts. Finally, the constraints of Equation 7c guarantee that each advertiser wins at most one bundle.

This ILP is known as the winner determination problem (WDP) in combinatorial auctions and it is well studied [**?**]. Although there are some tractable instances of the problem [**?**], the problem is NP-hard to solve in its general form [**?**]. The NP-hardness result holds also for our instance, since it is equivalent to the problem studied in Theorem 1. There are several approaches for the exact solution of the WDP [**?**] that use search algorithms to explore the space of possible solutions. Such algorithms can scale up to hundreds of thousands of items and tens of thousands of bids in seconds [**?**] (in certain cases). In our experiments we evaluate CABOB [15], the state-of-the-art search algorithm for the WDP, as the solver of the allocation problem in the sponsored search auctions with conflicts.

3.4 Payment rule

In traditional SSA the most popular payment rule requires from the advertiser who wins slot s_i to pay the bid of the advertiser who wins s_{i+1}. The rationale of this rule is that each advertiser pays the minimum bid that would be required to win his slot. We cannot apply this rule to SSA with conflicts, because an advertiser does not only have to outbid the advertiser with next highest bid, but also he has to outbid his conflicts.

To determine the prices for the advertisers that win the available slots we use the Vickrey-Clarke-Groves (VCG) mechanism. The VCG mechanism is a generalization of the Vickrey auction and similarly to the Vickrey auction it gives the bidders incentive to bid truthfully. The idea in VCG is that items are assigned to maximize the social welfare; then each advertiser pays the "opportunity cost" that his presence introduces to all the other advertisers. In the following paragraph we provide the explicit formula for the price charged to each advertiser.

Let $u_j^*(A, S)$ denote the utility of the advertiser a_j in the optimal allocation x^* (optimal with respect to social welfare maximization) of slots S to advertisers A:

$$u_j^*(A, S) = \sum_{i=1}^{m} v_j \rho_i x_{ij}^*. \quad (8)$$

Then, the price for the advertiser a_j equals the difference of the total utility that the other advertisers receive when a_j does not participate and when a_j participates in the auction:

$$p_j = \frac{\sum_{k \neq j} u_k^*(A - \{j\}, S) - \sum_{k \neq j} u_k^*(A, S)}{\rho_i} \quad (9)$$

The difference is divided by ρ_i, the click probability of the slot assigned to a_j in the optimal allocation, to obtain the price per click.

To illustrate, we calculate the advertiser payments in our running example. Recall that in the optimal solution advertiser a_1 wins s_1 and advertiser a_4 wins s_2. If advertiser a_1 does not participate in the auction, then the winners of the auction will be advertiser a_2 (s_1) and advertiser a_3 (s_2). Thus, the payment of advertiser a_1 is:

$$p_1 = \frac{v_2\rho_1 + v_3\rho_2 - (v_4\rho_2)}{\rho_1} = \frac{\$1.9}{0.2} = \$9.5$$

Note that the advertiser a_1 is mostly charged for depriving advertisers a_2 and a_3 of ad slots rather than for "pushing" advertiser a_4 to the second slot. With similar calculations the payment for a_4 is:

$$p_4 = \frac{v_2\rho_1 + v_3\rho_2 - (v_1\rho_1)}{\rho_2} = \frac{\$0.1}{0.1} = \$1.$$

The main criticism against the VCG mechanism is its computational inefficiency, since it requires solving the allocation problem n times to find the payment for each bidder. In our setting the number of bidders/advertisers that can win a slot is limited by the number m of slots and the advertisers that do not win any slot have zero payment. Thus, to determine the prices for the winners we need to the allocation problem at most m times no matter what the number of advertisers is.

3.5 Implications on utilities and bids

A natural question that arises with conflict constraints is whether the search engine is likely to make more or less money with this added feature. In particular, constraints rule out certain combinations of ads, so shouldn't search engine revenue drop as a consequence? The first step in answering this question is to consider whether advertisers will bid differently on a search engine that offers conflicts as opposed to one that does not.

To illustrate, consider an advertiser a_j that wishes to display ads on two search engines, T (traditional) and K (conflicts supported). The search engines and their advertisers are identical, except that K supports conflict constraints while T does not. Say that a_j submits an ad to K with a valuation $v_j = 100$ and a conflict with another advertiser a_k. What is the valuation of this same ad at T? Since K does not support conflict constraints, sometimes it will display the ad together with the undesired a_k ad, and sometimes is will be shown without a_k. Say that the former case (a_j and a_k appear) occurs with probability β.

We can estimate this probability as the fraction of auctions where both a_j and a_k win a slot versus all the auctions where a_j wins a slot. Say that $\beta = 0.5$ and for simplicity say that the click probability of the slot that a_j wins is 1. Then the expected utility of advertiser a_j at K is:

$$(1 - \beta)\rho_i v_j = 0.5 \times \$10 = \$5$$

Note that we have to discount his raw utility $\rho_i v_j$ by $1 - \beta$ because of the advertiser conflict with a_k. Since the advertiser utility is \$5, the advertiser has to bid \$5 at T instead of the \$10 he bid at K to avoid paying more than his utility. So if the advertiser bids truthfully and we know the value of β, we can estimate the value v_j at T as $v_j = b_j/(1 - \beta)$.

We can generalize this example to an advertiser a_j with $d = |C_j|$ conflicts. If advertiser a_j wins a slot along with one of his conflicts with probability β, and the probability for each conflict is independent from the others, then he will win a slot next to none of his conflicts with probability $(1-\beta)^d$. Assuming truthful bidding, we can estimate the advertiser's click valuation v_j at T from his K bid b_j as:

$$v_j = b_j/(1 - \beta)^d. \quad (10)$$

The formula shows that an advertiser will bid *more* at a site that supports conflict constraints, as opposed to one that does not. We take this fact into account in Section 4.4 when we compare the revenue at a search engine like K, as compared to one like T.

4. EXPERIMENTS

In this section we evaluate the performance of the Debbassac algorithm and we study the properties of sponsored search auctions with conflicts through a series of experiments. To goals of these experiments are the following:

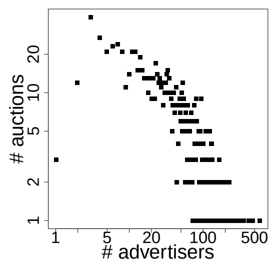
- to evaluate the run time efficiency of Debbassac and understand the factors that affect its performance, e.g., the number of advertisers that participate in the auction, the number of conflicts among them and the click probabilities distribution (Section 4.2);
- to compare the run time of Debbassac against the state-of-the-art algorithm that solves the allocation problem in the combinatorial auction formulation presented briefly in Section 3.3 (Section 4.3); and
- to assess the impact of the conflict constraints on the search engine revenue (Section 4.4).

4.1 Experimental Setup

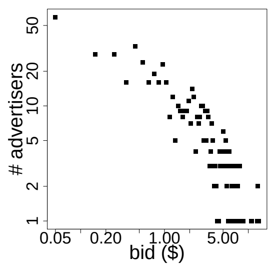
4.1.1 Dataset

We use the Yahoo! Search Marketing Advertiser Bidding Data from the Yahoo! Webscope [1] dataset collection. The dataset contains the advertiser bids for the top 1,000 keywords by volume during the time period from Jun 15, 2002 to Jun 14, 2003. Each dataset record includes an advertiser identifier, a keyword identifier, the bid amount and the time. These records contain 10,475 distinct advertiser identifiers.

[1]`http://webscope.sandbox.yahoo.com`



(a) Advertisers per auction



(b) Bid distribution (single auction)

Figure 2: Dataset statistics.

Since we focus on a single auction we ignore the time aspect of the bids and we construct 1,000 auctions, one for each keyword. The advertisers that participate in the auction for a certain keyword are all the advertisers with at least one bid on the keyword in the dataset. The bid of each advertiser is considered to be equal to the mean value of his bids on the auction keyword.

In Figure 2 we report some interesting statistics about the 1,000 auctions of our dataset. In Figure 2(a) we plot the distribution of the number of advertisers over the 1,000 keyword auctions in double logarithmic axes. Note that the distribution is skewed: there are few auctions with many advertisers and many auctions with few advertisers. As we discuss later in Section 4.2, such a distribution has implications on the reported mean run times of Debbassac over the auction instances of our dataset. In Figure 2(b) we plot the distribution of the advertiser bids in the auction for the keyword with identifier 122 (the distribution for other auctions look similar). Note that the distribution of bids is also skewed. Thus, although the number of advertisers that participate in an auction is large there are actually few who have realistic chances to get one of the m auctioned slots. As we discuss later such a bid distribution paired with the smart search space pruning of Debbassac makes the NP-hard allocation problem efficiently solvable in practical cases.

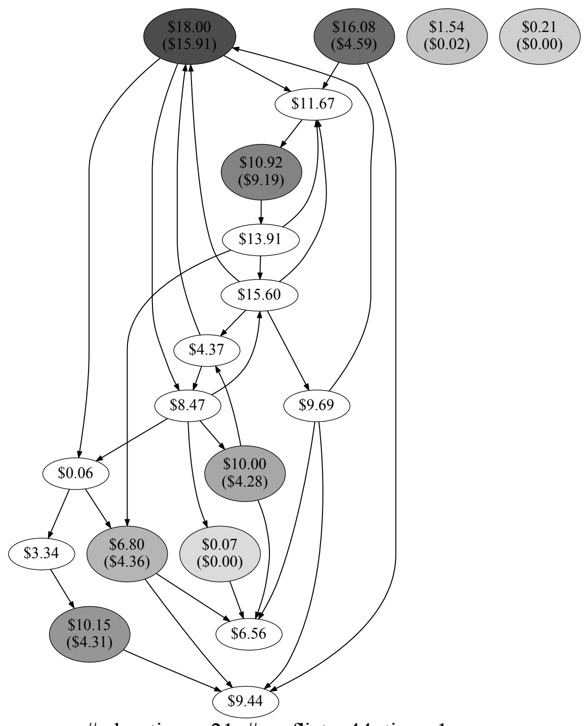
4.1.2 Conflicts

Our dataset auction instances come from real data and they do not contain conflicts. We introduce conflicts in these instances by simulating two scenarios that we discuss in the following paragraphs.

Competing advertisers.

The motivation for the competing advertisers or simply "competitors" scenario is that advertisers may introduce conflicts with their competitors to ensure exclusivity in the ad section of the results page. For example, Hertz may introduce conflicts against ads from Enterprise to get a greater market share. Recall that an advertiser can also place a bid without conflicts (as discussed in Section 2) to ensure that he is not deprived of a winning slot because of his own conflicts.

To implement this scenario we need to identify advertisers that can be considered as competitors. Although there are many ways to identify such advertisers, we picked competitors through the similarity of their bids. In particular, we define the similarity $sim(a_j, a_k)$ between advertisers a_j and a_k as the



#advertisers: 21, #conflicts: 44, time: 1 msec

Figure 3: The conflict graph (partial) of an auction with conflicts between competitive advertisers.

Jaccard index between their keyword bids:

$$sim(a_j, a_k) = \frac{\text{\# keywords with bids from both } a_j \text{ and } a_k}{\text{\# keywords with bids from either } a_j \text{ or } a_k} \quad (11)$$

Then, we consider the advertisers a_j and a_k as competitors if $sim(a_j, a_k) \geq 0.5$ and they have at least 2 keywords in common.

Given that advertisers a_j and a_k are competitors, we introduce a conflict between them with probability ψ. The direction of the conflict, i.e., $a_j \in C_k$ or $a_k \in a$, is determined by an unbiased coin. We generated conflicts for the following 10 values of the conflict probability ψ: $0.1, 0.2, \ldots, 1$ and for every value of ψ we created 20 instances of the conflict graph. Finally, we obtain $1,000 \times 10 \times 20 = 200,000$ instances of auctions with conflicts among competing advertisers.

In Figure 3 we present part of the conflict graph for one of the instances with $\psi = 0.3$. The gray nodes of the graph indicate advertisers that win a slot. The darker the gray color the higher the click probability of the slot. Each node in the graph shows the bid of the corresponding advertiser. The gray nodes show in the parenthesis the price charged to the advertiser for the slot he won. Note that the resulting conflict graph is quite dense even for $\psi = 0.3$. Thus, we believe that the auction instances that we study in this paper are more demanding than expected in practice and our run time results can be considered upper bounds of real-world performance.

Conflict-breeding advertisers.

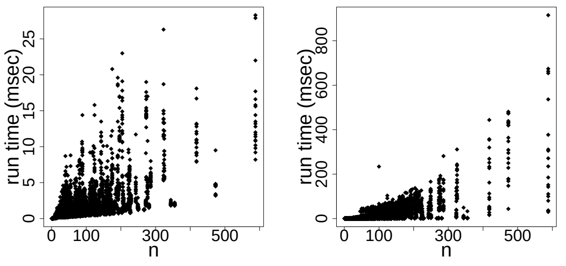
The motivation for the conflict-breeding advertisers or simply "conflict-breeders" scenario is a set of advertisers may attract conflicts from the rest of advertisers. For example, the ads of scammers may appear in the conflict sets of many non-scammer advertisers.

To implement this scenario we first select a set of κ conflict-breeding advertisers with weighted sampling. The weight of each advertiser is the sum of his bids. We use such a weight to select the conflict-breeders among the advertisers with either bids to many (and possibly unrelated) keywords or advertisers with high bids. The former advertisers include spammers who try to win slots independent of the submitted queries, while the latter advertisers include scammers, who have high valuations for clicks because of the potential high returns.

The values that we used for the parameters κ and ψ are $\kappa = 50, 100, \ldots, 300$ and $\psi = 0.1, 0.2, \ldots, 1$. We selected the conflict-breeding advertisers once for each value of κ and then we generated conflicts 20 times for each value of ψ. Finally we obtain $1,000 \times 6 \times 20 = 120,000$ auction instances with conflict-breeding advertisers.

In Figure 4 we present part of the conflict graph for one of the instances with $\kappa = 200$ and $psi = 0.3$. The node colors and the presented values have the same semantics as in Figure 3. Note that the resulting conflict graphs are bipartite with conflict-breeders in one part and the rest of the advertisers in the other. Note also that the resulting graphs are quite dense, since the expected in-degree of a conflict-breeding advertiser is $(n - \kappa)\psi$, while the expected out-degree of a regular advertiser is $\kappa\psi$. The conflict graph densities of our dataset auctions make the allocation problem more challenging than what is expected in practice.

4.1.3 Click probabilities



(a) Competitors (b) Conflict-Breeders

Figure 5: Debbassac run time vs number of advertisers.

Our dataset does not contain information about the click probabilities of ad slots. However, various other studies [7, 14] indicate that the click probabilities follow a Zipfian distribution. Thus, we assume that the click probability of slot s_i is $\rho_i \propto 1/i^\sigma$. Since the objective of the allocation problem (Equation 2) is a linear combination of the click probabilities, the solution to the problem is indifferent to the proportionality constant between ρ_i and $1/i^\sigma$. However, the parameter σ affects the solution and in Section 4.2.3 we show that it also affects the run time of Debbassac. Outside of Section 4.2.3 we use $\sigma = 0.5$.

4.1.4 Implementation

We implemented the Debbassac and the CABOB algorithm in Python and we have made our implementation available online[2]. We ran our code on a CentOS Linux server with two Xeon X5680 processors and 196GB of memory. Despite the large amount of available memory, our code uses only few tens of megabytes for each auction instance.

4.2 Debbassac run time analysis

In this section we study the run time of Debbassac. We are interested to see whether the Debbassac run time is acceptable in practice, where the allocation problem must be solved in few hundreds of milliseconds. This time is required by the search engine to compute the organic search results and the allocation problem should be solved concurrently in the same amount of time.

We perform the run time analysis with respect to the following parameters:

(i) the number of the auction advertisers n,

(ii) the density of the conflict graph that is controlled by the parameter ψ in the competitors scenario and by the parameters ψ and κ in the conflict-breeders scenario, and

(iii) the distribution of the click probabilities that is controlled by the parameter σ of the Zipfian distribution.

Note that the reported run times do not include the time to calculate the prices for the winners. We study the overhead of pricing in Section 4.2.4.

We run the Debbassac algorithm on all auction instances from both the competitors and the conflict-breeders scenario. We discuss the results in the following paragraphs.

4.2.1 Number of advertisers

In Figure 5 we plot the run time of Debbassac with respect to the number of advertisers in an auction n. Sub-figure 5(a) focuses on auction instances from the competitors scenario

[2]http://infolab.stanford.edu/~ppapadim

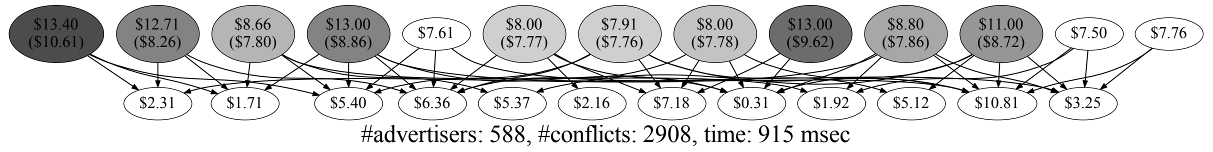
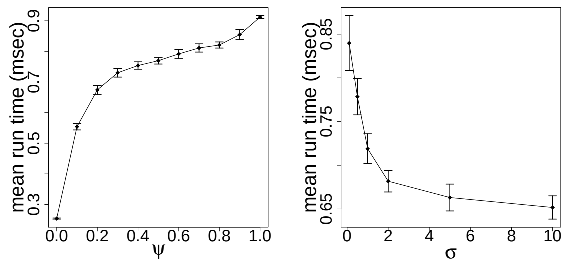


Figure 4: The conflict graph (partial) of an auction with conflict-breeding advertisers.



(a) $\sigma = 0.5$ (b) $\psi = 0.3$

Figure 6: Debbassac mean run time vs (a) the conflict probability ψ, and (b) the Zipfian distribution exponent σ in the competitors scenario auctions.

with $\psi = 0.3$. The vertical axis shows the run time and the horizontal axis shows the number of advertisers. Each point of the plot corresponds to an auction instance. The maximum run time is below 30 milliseconds which is considered acceptable. Note that the run time increases as n increases, but the increase is diminishing for large values of n. This trend is a consequence of the skewed bid distribution that we showed in Figure 2(b). If n is large, then there are enough advertisers with high bids to win all of the available slots and the tail advertisers are ignored by the Debbassac pruning. In such cases the run time of the algorithm is determined by the number of advertisers with high bids rather than the total number of advertisers n.

Sub-figure 5(b) focuses on auction instances from the conflict-breeders scenario with $\psi = 0.3$ and $\kappa = 200$. Note here that the run time increases with n and the increase seems to be quadratic on n. Thus, in the conflict-breeders scenarios there are run times for $n \approx 600$ that are not acceptable. The reason for the quadratic increase in the run time is not the number of advertisers but the number of conflicts that increase quadratically because of the scenario design. Since κ is constant, the probability that a conflict-breeder will participate in an auction grows linearly with the number of advertisers n. Every conflict-breeder attracts $n\psi$ conflicts and consequently the number of conflicts in the auction increases quadratically with n. We further discuss the impact of conflicts on the run time of Debbassac in the following subsection.

4.2.2 *Conflict graph density*

The conflict graph density is another factor that affects the run time of Debbassac, because conflicts increase the number of alternatives that the algorithm has to explore to allocate a slot. We show the effect of conflicts in Figure 6(a) for the competitors scenario and in Figures 7(a) and 7(b) for the conflict-breeders scenario (Figures 6(b) is discussed later on).

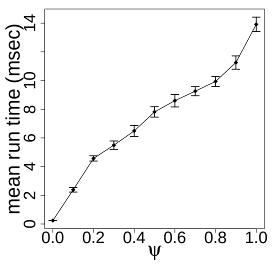
In Figure 6(a) we plot the mean running time of Debbassac as a function of ψ for the competitors scenario auctions. Recall that for every value of ψ we generated 20 different conflict graphs and for each conflict graph we solved the allocation problem for all $1,000$ auction instances. First we compute the mean run time over these $1,000$ instances and then we compute the mean of 20 means for each value of ψ. Each point in the plot shows the mean of the means for a particular value of ψ and the error bars indicate 95% confidence intervals. Note that the mean time is less than 1 millisecond for all values of ψ and this result shows that Debbassac can handle conflicts efficiently. Note also that the curve has a steep increase in the $0-0.3$ range of ψ and the increase slows down for larger value of ψ. The reason for the initial steep increase is that as ψ increases advertisers without conflicts obtain at least one conflict and Debbassac cannot take advantage of the pruning condition of lines 14-16 (Algorithm 1) in the top level of the search tree. To illustrate, say that an advertiser a_j has four competitors. The probability that there is at least one conflict between him and one of his competitors is $1 - (1 - \psi)^4$. This probability increases from 0 for $\psi = 0$ to 0.76 for $\psi = 0.3$ and it reaches 1 for $\psi = 1$. Thus, for $\psi = 0.3$ the probability that Debbassac does not explore the advertisers $\{a_k \mid v_k < v_j\}$ at the top level of the search tree drops to 0.24 for $\psi = 0.3$ from 1 for $\psi = 0$.

In Figure 7(a) we plot the mean run time of Debbassac as a function of ψ for the conflict-breeders scenario auctions and in Figure 7(b) we plot the mean run time as a function of κ. We restrict our study to auctions with $\kappa = 200$ for the former figure and to auctions with $\psi = 0.3$ for the latter figure. Recall that for each combination of κ and ψ we generated 20 conflict graphs and the reported run times are the means of the means similarly to Figure 6(a). The two curves have similar trends because increasing either ψ or κ has similar impact on the number of conflicts. The maximum run times are 14 milliseconds in Figure 7(a) is 7 milliseconds in Figure 7(b). Both these numbers are low and they confirm that Debbassac can handle conflicts efficiently.

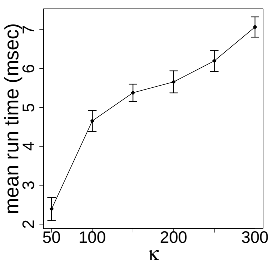
4.2.3 *Click probabilities*

The distribution of the click probabilities proved also to be an important factor for the run time of Debbassac. As mentioned above we assume that the click probabilities follow a Zipfian distribution and we study the impact of the exponent σ. We used the following values of σ: 0.1, 0.5, 1, 2, 5 and 10. For each value of σ we derived the slot click probabilities and we solved the allocation problem for all auction instances in our dataset. In Figure 6(b) we present the results for the competitors scenario auctions. The results for the conflict-breeders scenario are similar and we report them in [13].

In Figure 6(b) we plot the mean running time of Debbassac as a function of σ. The plot looks only at the $20 \times 1,000$ auction instances whose conflict graphs were generated with



(a) $\kappa = 200$



(b) $\psi = 0.3$

Figure 7: Debbassac mean run time vs (a) the conflict probability ψ, (b) the number of conflict-breeders κ in the conflict-breeders scenario auctions.

$\psi = 0.3$ (plots for other ψ values are similar). As discussed above, for each conflict graph we calculate the mean run time over the $1,000$ auctions and the plot shows the mean of the 20 means that we obtain. The error bars indicate 95% confidence intervals. Note that the mean run time drops from 6.5 milliseconds for $\sigma = 0.1$ to less than 5 milliseconds for $\sigma = 10$. The reason why the run time decreases is that high values of σ increase the relative difference among the slot click probabilities. For example, for $\sigma = 10$ the ratio of the click probabilities between the first and the second slot is $\frac{1/1^{10}}{1/2^{10}} = 2^{10}$. With such a big ratio, it is preferable to allocate the first slot to the advertiser with the highest bid while ignoring his many conflicts rather than allocating the first slot to another advertiser with lower bid and no conflicts. The allocation of the first slot to the highest bid is more beneficial for the the social welfare rather than the more efficient allocation of the rest of the slots. For smaller values of σ the algorithm has to estimate more carefully the trade-off of allocating a slot to an advertiser versus allocating the same slot or others to his conflicts. However, it is important that Debbassac can take advantage of certain click probability distributions and decrease its run time.

4.2.4 Pricing overhead

The reported run times so far do not include the time to assign prices to the winning advertisers. Although the search engine needs to find the slot winners by the time it displays the search results to users, it can determine the slot prices even beyond this time point. So the time constraint of few hundred milliseconds does not apply to the price determination time.

Even if the time contraint was applied to the price determination, note that the total run time to determine the prices is not equal to the minimum possible latency to learn these prices. Recall that the price determination requires solving up to m allocation problems. However, if the search engine needs to have the prices as early as possible, it can solve the m allocation problems in parallel, since they are independent. So even if the computational cost of pricing is m times the cost of solving the allocation problem, the minimum latency to calculate the prices is equal to the run time of a single allocation problem.

We measured the run time to determine the slot prices for all the auctions of our dataset. Our code solved the m allocation problems in a serial fashion. In Figure 8 we plot the empirical CDF of the ratio

$$\alpha = \frac{\text{pricing run time}}{\text{allocation run time}}$$

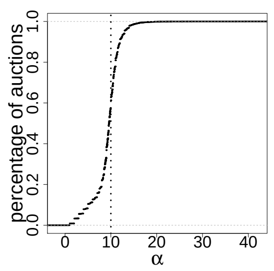


Figure 8: Empirical CDF of the α ratio ($\psi = 0.3$).

for the auctions of the competitors scenario with $\psi = 0.3$. The x-axis shows the value of α and the y-axis shows the percentage of auctions whose ratio of price to allocation run time is less than α. Since there are $m = 10$ slots, the expected value for α is 10. However, the value of α is not always exactly 10 because the m problems that need to be solved are not identical with the original allocation problem. Note that α is less or equal to 10 for approximately 60% of the auctions and more than 10 for the rest 40%. This was exected because an allocation problem is expected to be simpler to solve after we remove one advertiser. However, there are some rare cases where pruning cannot be efficient in the resulting problems and the pricing run time is 30 times greater than the allocation time. The results for the conflict-breeders sceario are similar and we do not report them here. In both cases the time overhead of price determination is on average slightly less than m times the run time of the winner determination problem.

4.3 Comparison with CABOB

In this section we compare the run time of Debbassac with CABOB [15], the state-of-the-art algorithm for the allocation problem in combinatorial auctions. Despite the different formulations, recall that both algorithms provide an exact solution to the same problem. So their solutions are the same and our comparison refers only to the run times of the two algorithms.

We implemented CABOB and a program to transform a sponsored search auction with conflicts to a combinatorial auction. Note that the run times we provide for CABOB do not include the run time of the transformation. Since the difference in the run times between Debbassac and CABOB exceed the three orders of magnitude, we provide here only one figure with numerical results and we discuss the differences of the two algorithms qualitatively. In Figure 9 we plot the mean run time of the two algorithms as a function of κ. The mean is computed over $1,000$ auctions with conflicts from one conflict graph with $\psi = 0.3$ and the κ indicated in the plot. We did not use all of the 20 conflict graphs that we have generated for every combination of κ and $\psi = 0.3$ because of the high cost of running CABOB. The dashed line in the plot looks at the CABOB running times and the solid line looks at the Debbassac run times. Note that the maximum time for Debbassac is below 10 milliseconds, while the minimum run time of CABOB exceeds the $10,000$ milliseconds.

Although both algorithms use DFS with branch-and-bound pruning, there is a huge performance gap between them. The main reasons for this gap are the following: (a) CABOB solves an LP problem in every node of the search tree. The time to solve one LP is usually more than the run time required by Debbassac to solve the allocation problem. (b) One of

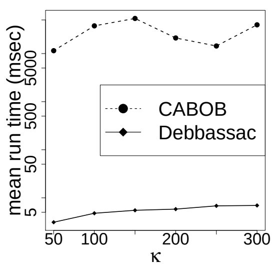


Figure 9: Mean run time comparison of Debbassac and CABOB in auctions with conflict-breeding advertisers.

the main optimizations of CABOB is that it tries to allocate early the items that have bids from many bidders. The hope is that after allocating such items, there will be only items with bids by a single advertiser and the allocation problem will become trivial. However, in the combinatorial auction formulation of the sponsored search auctions with conflicts all advertisers bids include the same items, i.e., the real slots, and the CABOB optimization cannot yield any benefit. (c) Debbassac is tailored to the allocation problem of sponsored search auctions with conflicts and it uses various problem properties to prune the search space. For example, Debbassac considers allocating slots with lower click probabilities only to advertisers with lower bids and it prunes the search space when a an advertiser without conflicts is encountered. Such pruning decisions reduce dramatically the number of nodes in the search tree and the running time of the algorithm.

Although CABOB is a general purpose algorithm, it is still the state-of-the-art approach for solving the allocation problem in combinatorial auctions. The fact that Debbassac yields great improvement in the run time with respect to CABOB shows that Debbassac does efficiently take advantage of the problem specific properties. However, the combinatorial auction formulation is still useful, since it can benefit from any progress in the allocation or pricing methods for combinatorial auctions.

4.4 Revenue Implications

In this section we study the implications of conflicts on search engine revenue. As in Section 3.5, we consider a traditional search engine T that does not support conflict constraints with an engine K that does support conflicts. Then we address the question: Under otherwise equal circumstances, will K make more or less money than T?

There are two competing factors in this comparison. On the positive side (for K), advertisers will bid more at K than at T (see Equation 10). On the negative side, due to conflicts K will have fewer display choices than T so may "lose out on some opportunities." Which factor is more important depends on the value of β (defined in Section 3.5).

While we could estimate the value of β in our two specific conflict scenarios, we believe it is more instructive to study search engine revenue as β varies. In other words, we will answer the question: How large does β have to be so that the search engine K does not make less money than T? To find the answer, we assume that the bids in our data sets are for engine T, and we calculate the K bids for $\beta = 0$ to $\beta = 0.02$ with 0.01 increments. In the calculation of the new bids, we take into account only the conflicts within each auction to avoid bid inflation. Then we solve the allocation problem using the new bids for all auctions in our dataset and we calculate the K search engine revenue. We plot the results for the auctions of the competitors scenario in Figure 10. The results for the conflict-breeders are almost identical and we do not plot them here. The x-axis shows the value of β and the y-axis shows the K revenue. There are three lines in the plot and each line looks at auctions with different value of ψ. The flat solid line shows the mean T revenue in the $1,000$ auctions without conflicts. The line is flat, since at K the advertisers report no conflicts and the change of β does not affect the advertiser bids. The dotted line looks at auctions with $\psi = 0.1$, the dashed line looks at auctions with $\psi = 0.2$ and the solid line looks at auctions with $\psi = 0.3$. Each point in any line shows the mean revenue among all auctions with the corresponding ψ value. Note that all lines are increasing because the K advertiser bids increase with β. Note also that the higher the ψ the higher the slope of the curves, because higher ψ results into more conflicts and higher bid increases. The most important conclusion from this graph is that all three blue lines cross the $\psi = 0$ line between the relatively low values of $\beta = 0.02$ and $\beta = 0.03$. Thus, if we expect that in traditional sponsored search auctions the advertisers win a slot next to a potential conflict, i.e., either a competitor or a spammer, with probability more than 0.02, then the search engine could increase its revenue by supporting conflicts.

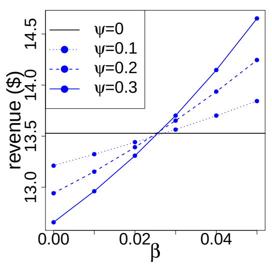


Figure 10: Search engine revenue versus probability β.

5. RELATED WORK

Sponsored search auctions are studied in the two seminal papers by Varian [16] and Edelman et al. [5]. A book chapter by Lahaie et al. [9] provides also a good overview about the work on the field.

The work of Ghosh et al. [6] and the work of Muthukrishnan [11] are the most relevant to our work. Ghosh et al. [6] study auctions where the advertiser bids are conditioned upon *exclusivity* on the sponsored search results, i.e., one advertiser wins the first slot and the rest of the slots remain empty. Our auction is more expressive, since an advertiser can achieve exclusivity by declaring conflicts against the rest of the advertisers. However, their limited setting allows them to present strong theoretical results about the payment rule and the search engine revenue. Muthukrishnan [11] studies auctions where the bid of advertiser a_j is conditioned on the ad appearing at the top-k_j positions. Such a constraint relies on the ad ranking and cannot be expressed through conflicts.

Our work is also related to various other works [1, 8] that study *externalities* in sponsored search or more expressive sponsored search auctions [10, 2, 4]. An externality refers to the dependence of a click valuation on the context where an ad is displayed. The works on externalities focus on the study of such dependences rather than on allowing advertisers to condition their bids upon them. The works on expressive sponsored search auctions study alternative utility functions that take into account clicks, impressions and may be discontinuous. We think that the ever-increasing volume of work on expressive sponsored search auctions reflects the advertisers' interest for greater control over the display of their ads. The support of conflict constraints is a big step towards this direction.

6. CONCLUSIONS

In this paper we introduced a variation of sponsored search auctions where advertisers are allowed to declare conflicts against their peers. The main focus of our study is the computational feasibility of the new auctions. Despite the NP-hardness result for the slot allocation problem, we have presented Debbassac, a DFS branch-and-bound algorithm, that can efficiently allocate the auctioned slots to advertisers in practical cases. Our experiments showed that Debbassac can solve the allocation problem of an auction with hundreds of advertisers and thousands of conflicts in tens to hundreds of milliseconds.

We have studied the implications of the new auction on search engine revenue using a VCG payment rule to determine prices. Our results show that a search engine can potentially increase its revenue by introducing conflicts. The increase of the revenue is not at the expense of the advertisers. The advertisers enjoy actually higher utility per auction, since they have greater control over the context where their ads get clicked.

In our future work we plan to explore alternative approaches for the payment rule of the new auction. Although the VCG mechanism provides truthful bidding, a simpler GSP-like payment rule might be more appealing to search engines. Another promising direction for future work is the introduction of additional interaction among advertisers. For example, advertisers may want to "follow" other advertisers and have their ads appear next to them. We plan to study such interactions in our future work.

7. REFERENCES

[1] G. Aggarwal, J. Feldman, S. Muthukrishnan, and M. Pal. Sponsored search auctions with markovian users. In *Internet and Network Economics*, *LNCS* 5385: 621–628. Springer Berlin / Heidelberg, 2008.

[2] G. Aggarwal, S. Muthukrishnan, D. Pál, and M. Pál. General auction mechanism for search advertising. In WWW, 2009.

[3] P. Cramton, Y. Shoham, and R. Steinberg. *Combinatorial Auctions*. The MIT Press, 2006.

[4] P. Dütting, M. Henzinger, and I. Weber. An expressive mechanism for auctions on the web. In WWW, 2011.

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

[6] A. Ghosh and A. Sayedi. Expressive auctions for externalities in online advertising. In WWW, 2010.

[7] D. Kempe and M. Mahdian. A cascade model for externalities in sponsored search. In WINE, 2008.

[8] D. Kempe and M. Mahdian. A cascade model for externalities in sponsored search. In *Internet and Network Economics*, *LNCS* 5385: 585–596. Springer Berlin / Heidelberg, 2008.

[9] S. Lahaie, D. M. Pennock, A. Saberi, and R. V. Vohra. Sponsored search auctions. In *Algorithmic Game Theory*. Cambridge University Press, 2007.

[10] D. J. Martin, J. Gehrke, and J. Y. Halpern. Toward expressive and scalable sponsored search auctions. In ICDE, 2008.

[11] S. Muthukrishnan. Bidding on configurations in internet ad auctions. In COCOON, 2009.

[12] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.

[13] P. Papadimitriou and H. Garcia-Molina. Sponsored search auctions with conflict constraints. Technical report 1010, Stanford InfoLab, 2011. URL: `http://ilpubs.stanford.edu:8090/1010/`

[14] D. H. Reiley, S.-M. Li, and R. A. Lewis. Northern exposure: a field experiment measuring externalities between search advertisements. In EC, 2010.

[15] T. Sandholm, S. Suri, A. Gilpin, and D. Levine. Cabob: a fast optimal algorithm for combinatorial auctions. In IJCAI, 2001.

[16] H. R. Varian. Position auctions. *International Journal of Industrial Organization*, 25(6):1163 – 1178, 2007.

APPENDIX

A. SSA WITHOUT CONFLICTS

In this section we present the allocation and the payment rule that are used in traditional sponsored search auctions. Recall that in such auctions the advertisers cannot specify conflicts.

Our presentation is informal and example-based. We point the reader to existing literature for more details. Throughout the section we use the following auction as an example: there are four advertisers $A = \{a_1, a_2, a_3, a_4\}$ with click valuations $v_1 = \$10$, $v_2 = \$8$, $v_3 = \$5$ and $v_4 = \$2$ that participate in an auction for two slots $S = \{s_1, s_2\}$. The click probabilities of the two ad slots are $\rho_1 = 0.2$ and $\rho_2 = 0.1$.

A.1 Allocation rule

The allocation rule in traditional sponsored search auctions solves an optimization problem that is identical to the allocation problem in Equation 2 without the constraints of Equation 2d that express advertiser conflicts.

The problem can be viewed as an instance of the *maximum weighted bipartite graph matching* problem that can be solved in polynomial time. Recall that a matching in a bipartite graph is a set of edges without common vertexes. The maximum weighted matching is the matching with the maximum total edge weight. We construct a bipartite graph $G = (V = (S, A), E)$ between the ad slots S and the advertisers A. The graph has an edge $(s_i, a_j) \in E$ between every slot s_i and every advertiser a_j with weight equal to $\rho_i v_j$. The maximum weighted matching in this graph yields an allocation that is feasible and maximizes the social welfare. In particular, each edge (s_i, a_j) of the matching indicates the allocation of slot s_i to the advertiser a_j. For example, in Figure A.1 we present the bipartite graph for our running example. The maximum weighted matching in this case consists of the edges (s_1, a_1) with weight 2 and $(2, 2)$ with weight 0.8. So the advertiser a_1 wins the first slot and the advertiser a_2 wins the second.

Note that in case of ad slots with advertiser independent click probabilities, the *assortative* allocation, e.g., the advertiser with the highest bid wins the first slot, the advertiser with the second highest bid wins the second slot, and so on, is optimal.

A.2 Payment rule

In this section we present two different payment rules: in Section A.2.1 we present a payment rule that is based on the Vickrey-Clarke-Groves (VCG) mechanism and motivates truthful bidding. The VCG mechanism can be applied in any auction where the allocation rule maximizes the social welfare and we will also use it for the payment rule of the sponsored search auctions with conflicts. In Section A.2.2 we present the payment rule that is used in practice by the search engines. Although this rule does not motivate truthful bidding, it is widely used because of its simplicity. Moreover, this rule has an expost equilibrium, i.e., there are advertiser bids such that the advertisers have no incentive to change their bids even if they know other advertisers' bids. The existence of such an equilibrium helps avoiding bid fluctuations that may result in inefficient allocations.

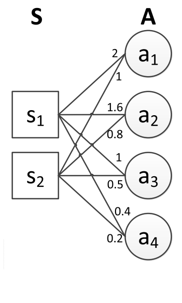


Figure 11: Maximum weighted bipartite graph matching.

A.2.1 VCG

The VCG mechanism is a generalization of the Vickrey auction[3] and similarly to the Vickrey auction it gives the bidders incentive to bid truthfully. The idea in VCG is that items are assigned to maximize the social welfare; then each advertiser pays the "opportunity cost" that his presence introduces to all the other advertisers. In the following paragraph we provide the explicit formula for the price charged to each advertiser.

Let $u_j^*(A, S)$ denote the utility of the advertiser a_j in the optimal allocation x^* (optimal with respect to social welfare maximization) of slots S to advertisers A:

$$u_j^*(A, S) = \sum_{i=1}^{m} v_j \rho_i x_{ij}^*. \tag{12}$$

Then, the price for the advertiser a_j equals the difference of the total utility that the other advertisers receive when a_j does not participate and when a_j participates in the auction:

$$p_j = \frac{\sum_{k \neq j} u_k^*(A - \{j\}, S) - \sum_{k \neq j} u_k^*(A, S)}{\rho_i} \tag{13}$$

The difference is divided by ρ_i, the click probability of the slot assigned to a_j in the optimal allocation, to obtain the price per click. The reader can find a formal proof for the truthful bidding property of the VCG mechanism in textbooks [12]. Here, we only provide some qualitative arguments. Note that the price charged on the advertiser a_j does not depend explicitly on his bid, but it is calculated as a function of the bids of the other advertisers. Consequently, if an advertiser wins a particular slot with different bids, his price will be the same no matter what his bid is. Thus, if a_j can win the i-th slot based on the advertisers' click valuations, he has no incentive to bid less than his valuation, because (a) he may lose the i-th slot (if he bids too low), or (b) if he wins the i-th slot he will pay the same price as if he had bid his true valuation. The advertiser does not also have incentive to bid more than his valuation, because he may end up winning the $i-1$-th slot at a price that he cannot afford.

To illustrate the calculations for the VCG prices, we compute the prices for the advertisers of our running example. Recall that in the optimal allocation advertiser a_1 wins the first slot and a_2 wins the second slot. To calculate the price for a_1 note that if he does not participate in the auction, then advertiser a_2 will win the first slot and a_3 will win the second slot.

[3]A Vickrey or second-price auction is a type of sealed-bid auction, where bidders submit written bids without knowing the bid of the other people in the auction, and in which the highest bidder wins, but the price paid is the second-highest bid. The Vickrey auctions provides the bidders with incentive for truthful bidding.

Thus, the VCG payment for advertiser a_1 is:

$$\begin{aligned} p_1^* &= \frac{v_2 \times \rho_1 + v_3 \times \rho_2 - v_2 \times \rho_2}{\rho_1} \\ &= \frac{\$8 \times 0.2 + \$5 \times 0.1 - (\$8 \times 0.1)}{0.2} \\ &= \$1.3/0.2 = \$6.5. \end{aligned}$$

Similarly, the price for advertiser a_2 is:

$$\begin{aligned} p_2^* &= \frac{v_1 \times \rho_1 + v_3 \times \rho_2 - v_1 \times \rho_1}{\rho_2} \\ &= \frac{v_3 \times \rho_2}{\rho_2} \\ &= v_3 = \$5 \end{aligned}$$

Note that advertiser a_1 has to pay for the displacement of both a_2 (from slot s_1 to slot s_2) and a_3 (from slot s_2 to non-winning positions), while advertiser a_2 has to pay only for the displacement of a_3. Thus, the lower the rank of the slot that an advertiser wins, the higher the price that he has to pay.

A.2.2 Generalized Second Price

As mentioned above, the most popular search engines assume advertiser independent click probabilities and, consequently, they sort advertisers by their bids to find the optimal allocation. Inspired by the Vickrey or second-price auction where the winner of the single auctioned item pays the second highest bid, these search engines have introduced the following payment rule:

The price of the winner of the i-th slot is equal to the bid of the winner of the $i+1$-th slot. The winner of the k-th slot pays the highest bid among the advertisers that do not win any slot.

The auction with an assortative allocation rule and the aforementioned payment rule is known as *Generalized Second Price* (GSP) auction. An interesting fact about this auction is that it was used in practice for some years, before it was theoretically studied [16], [5]. The lack of theoretical analysis and the similarity to the second price auction lead to some misconceptions in the first days of its use. For example, Google, which first introduced this type of auction in 2002, advertised that the new type of auction is a generalization of the Vickrey auction and it motivates truthful bidding. The first part of the statement is true but the second is false unless the auction has a single item where the second price and the generalized second price auctions are identical. In the following example we show that advertisers may obtain higher payoffs if they do not bid truthfully.

In our running example say that the advertisers bid truthfully and that the click probabilities of the ad slots are $\rho_1 = 0.2$ and $\rho_2 = 0.15$. The advertiser payments in a GSP auction are $p_1 = v_2 = \$8$ and $p_2 = v_3 = \$5$. Thus, the payoff of advertiser a_1 is $\rho_1(v_1 - p_1) = 0.2(\$10 - \$8) = \$0.4$ and the payoff of advertiser 2 is $\rho_2(v_2 - p_2) = 0.15(\$8 - \$5) = \$0.45$. Advertiser a_1 has incentive to change his bid to $b_1 = \$7$, because in that case he will get the second slot, his price will be $p_1 = \$5$ and his payoff will increase to $\rho_2(v_1 - p_1) = 0.15(\$10 - \$5) = \$0.75$.

Although the GSP does not motivate truthful bidding, there are advertiser bids that lead to a *locally envy-free equilibrium*. An allocation x is called locally envy-free if there exist prices $\{p_i\}$, one for each slot (this is the only case in the paper where prices refer to slots rather than advertisers), such that for all i, j with $x_{ij} = 1$

$$\begin{aligned} \rho_i(v_j - p_i) &\geq \rho_{i-1}(v_j - p_{i-1}) \\ \rho_i(v_j - p_i) &\geq \rho_{i+1}(v_j - p_{i+1}) \end{aligned}$$

In words, if advertiser a_j is assigned to slot s_i, the he prefers slot s_i to the slot just above him and the slot just below him. The reader can find more about the envy-free equilibrium in the works of Edelman et al. [5] and Varian [16]. Here we only provide an example that qualitatively justifies its existence and illustrates some of its properties. In the example of the previous paragraph we saw that advertiser a_1 can change his bid to increase his payoff. However, in that case the payoff of advertiser a_2 will decrease, since he will end up winning the first slot at a price that is equal to the untrue bid of advertiser a_1, i.e., $p_2 = b_1 = \$7$. Then, the payoff of a_2 is $\rho_1(v_2 - p_2) = 0.2(\$8 - \$7) = \$0.2$. Thus, advertiser a_2 will also change his bid to move to the second slot and increase his payoff. After some fluctuations, the two advertisers may end up with the following bids: $b_1 = \$10$ and $b_2 = \$7$. In that case the payoff of advertiser a_1 is \$0.6 and the payoff advertiser a_2 is \$0.45. With such payoffs none of the two advertisers has incentive to change his bid and we end up in a locally envy-free equilibrium.

Two attractive properties of the GSP equilibrium are the following: (a) the advertisers, despite their untrue bids, are sorted according to their true valuation in an envy-free equilibrium; and (b) the sum of prices is at least as high as the sum of VCG prices. These two properties along with the simplicity of the payment rule have made the GSP auction the most widely used in traditional sponsored search auctions.