Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Ad Auctions and Cascade Model: GSP Inefficiency and Algorithms

Gabriele Farina and **Nicola Gatti**
Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano
Piazza Leonardo da Vinci, 32
I-20133, Milan, Italy
gabriele2.farina@mail.polimi.it, nicola.gatti@polimi.it

Abstract

The design of the best *economic mechanism* for Sponsored Search Auctions (SSAs) is a central task in computational mechanism design/game theory. Two open questions concern (i) the adoption of user models more accurate than the currently used one and (ii) the choice between Generalized Second Price auction (GSP) and Vickrey–Clark–Groves mechanism (VCG). In this paper, we provide some contributions to answer these questions. We study Price of Anarchy (PoA) and Price of Stability (PoS) over social welfare and auctioneer's revenue of GSP w.r.t. the VCG when the users follow the famous *cascade model*. Furthermore, we provide exact, randomized, and approximate algorithms, showing that in real–world settings (Yahoo! Webscope A3 dataset, 10 available slots) optimal allocations can be found in less than 1s with up to $1{,}000$ ads, and can be approximated in less than 20ms even with more than $1{,}000$ ads with an average accuracy greater than 99%.

Introduction

SSAs, in which a number of *advertisers* bid to have their *ads* displayed in some slot alongside the search results of a keyword, constitute one of the most successful applications of *microeconomic mechanisms*, with a revenue of about \$50 billion dollars in the US alone in 2014 (IAB 2015). Similar models are used in many other advertisement applications, e.g. contextual advertising (Varian and Harris 2014). A number of questions remain currently open in the study of effective SSAs. Among these, two are crucial and concern (i) the adoption of the best *user model* in real–world settings and (ii) the design of the best *economic mechanism*. In this paper, we provide some contributions to answer these questions.

The commonly adopted user model is the *position dependent cascade* (PDC) in which the user is assumed to observe the slots from the top to the bottom with probabilities to observe the next slot that depend only on the slots themselves (Narahari et al. 2009). The optimal allocation can be efficiently found in a greedy fashion. The natural extension is the *ad/position–dependent cascade* (APDC) model, in which the probability to observe the next slot depends also on the ads actually allocated. This model is extensively experimentally validated (Craswell et al. 2008; Joachims et al. 2007). However, it is not known whether finding the best allocation of APDC is $\mathcal{FNP}$–hard, but it is commonly conjectured to be. (Gatti and Rocco 2013) provides an exact algorithm that can be used in real time only with 40 ads or less, while in real–world applications the ads can be even more than $1{,}000$. The problem admits a constant–ratio $(1-\epsilon)/4$, with $\epsilon \in [0,1]$, approximation algorithm (Gatti and Rocco 2013; Kempe and Mahdian 2008), but no known algorithm is proved to lead to a *truthful* economic mechanism. In addition, the known approximation algorithms can be applied in real time only with 100 ads or less (Gatti and Rocco 2013). More expressive models include a contextual graph, see (Gatti et al. 2015; Fotakis, Krysta, and Telelis 2011), but they are Poly–$\mathcal{APX}$–hard. Hence, the APDC is considered as the candidate providing the best trade–off between accuracy and tractability.

While the GSP is still popular in many SSAs, the increasing evidence of its limits is strongly pushing towards the adoption of the more appealing VCG mechanism, which is already successfully employed in the related scenario of contextual advertising, by Google (Varian and Harris 2014) and Facebook (Hegeman 2010). The main drawback of the GSP is the *inefficiency* of its equilibria (in terms of social welfare) w.r.t. the VCG outcome (the only known results concern PDC): considering the whole set of Nash equilibria in full information, the PoA of the GSP is upper bounded by about 1.6, while considering the set of Bayes–Nash equilibria the PoA is upper bounded by about 3.1 (Leme and E. Tardos 2010). Furthermore, automated bidding strategies, used in practice by the advertisers to find their best bids, may not even converge to any Nash equilibrium and, under mild assumptions, the states they converge to are shown to be arbitrarily inefficient (Evangelos and Telelis 2010). Some works study how inefficiency is affected by some form of externalities, showing that no Nash equilibrium of the GSP provides a larger revenue than the VCG outcome (Kuminov and Tennenholtz 2009; Gomes, Immorlica, and Markakis 2009).

Original contributions. Under the assumption that the users behave as prescribed by APDC, we study the PoA and PoS both over the social welfare and over the auctioneer's revenue of the GSP w.r.t. the VCG based on APDC (we study also the restricted case in which the advertisers do not overbid). We extend our analysis done for the GSP to the VCG based on PDC. Both analyses show a wide dispersion of the equilibria with PoAs in most cases unbounded, pointing out the limits of the traditional models. Furthermore, we provide a polynomial–time algorithm that dramatically shrinks auctions instances, safely discarding ads (even more than 90% in our experiments). We also provide a randomized algorithm (that can be derandomized with polynomial cost) finding optimal allocations with probability 0.5 in real–world instances (based on the *Yahoo!* Webscope A3 dataset) with up to 1,000 ads and 10 slots in less than 1s and a *maximal–in–its–range* (therefore leading to truthful mechanisms if paired with VCG–like payments) approximation algorithm returning allocations with an average approximation ratio larger than about 0.99 in less than 20ms even with 1,000 ads. This shows that the APDC can be effectively used in practice also in large instances.

Due to reasons of space, the proofs of some theorems are omitted. We point an interested reader to (Farina and Gatti 2015) where they can find an extended version of the paper, including missing proofs and more detailed experimental results.

Problem formulation

We adopt the same SSA model used in (Gatti and Rocco 2013) with minor variations. The model without externalities is composed of the following elements:

- $\mathcal{A} = \{1, \dots, N\}$ is the set of ads. W.l.o.g. we assume each advertiser (also called agent) to have a single ad, so each agent a can be identified with ad $a \in \mathcal{A}$;
- $\mathcal{K} = \{1, \dots, K\}$ is the set of slots s ordered from the top to the bottom. We assume w.l.o.g. $K \leq N$;
- $q_a \in [0, 1]$ is the *quality* of ad a (i.e., the probability a user will click ad a once observed);
- $v_a \in V_a \subseteq \mathbb{R}^+$ is the value for agent a when ad a is clicked by a user, while $\mathbf{v} = (v_1, \dots, v_N)$ is the value profile;
- $\bar{v}_a$ is the product of q_a and v_a, and can be interpreted as the expected fraction of the ad value that is actually absorbed by the users observing ad a;
- $\hat{v}_a \in \hat{V}_a \subseteq \mathbb{R}^+$ is the value reported by agent a, while $\hat{\mathbf{v}} = (\hat{v}_1, \dots, \hat{v}_N)$ is the reported value profile;
- Θ is the set of (ordered) allocations of ads to slots, where each ad cannot be allocated in more than one slot.

Given an allocation θ of ads to slots, we let:

- $\text{ads}(\theta) \subseteq \mathcal{A}$ to denote the subset of ads allocated in θ;
- $\text{slot}_\theta(a)$ to denote the slot in which ad a is allocated, if any;
- $\text{ad}_\theta(s)$ to denote the ad allocated in slot s.

The externalities introduced by the cascade model assume the user to have a Markovian behavior, starting to observe the slots from the first (i.e., slot 1) to the last (i.e., slot K) where the transition probability from slot s to slot $s + 1$ is given by the product of two parameters:

- (*ad–dependent externalities*) $c_a \in [0, 1]$ is the *continuation probability* of ad a;
- (*position–dependent externalities*) $\lambda_s \in [0, 1]$ is the *factorized prominence* of slot s (it is assumed $\lambda_K = 0$).

The click through rate $\text{CTR}_\theta(a) \in [0, 1]$ of ad a in allocation θ is the probability a user will click ad a and it is formally defined as

$$\text{CTR}_\theta(a) = q_a \cdot \Lambda_{\text{slot}_\theta(a)} \cdot C_\theta(a),$$

where

$$C_\theta(a) = \prod_{s\,<\,\text{slot}_\theta(a)} c_{\text{ad}_\theta(s)},$$

$$\Lambda_s = \prod_{k\,<\,s} \lambda_k.$$

Parameter Λ_s is commonly called *prominence* (Kempe and Mahdian 2008).

Given an allocation θ, its social welfare is defined as:

$$\text{SW}(\theta) = \sum_{\text{ads}(\theta)} v_a \cdot \text{CTR}_\theta(a) = \sum_{\text{ads}(\theta)} \bar{v}_a \cdot \Lambda_{\text{slot}_\theta(a)} \cdot C_\theta(a).$$

The problem we study is the design of an economic mechanism $\mathcal{M}$, composed of

- an *allocation function* $f : \times_{a \in \mathcal{A}} V_i \to \Theta$ and
- a *payment function* for each agent a, $p_a : \times_{a \in A} \hat{V}_a \to \mathbb{R}$.

Each agent a has a linear utility $u_a(v_a, \hat{\mathbf{v}}) = v_a \cdot \text{CTR}_{f(\mathbf{v})}(a) - p_a(\hat{\mathbf{v}})$, in expectation over the clicks and pays $p_a(\hat{\mathbf{v}})/\text{CTR}_{f(\hat{\mathbf{v}})}(a)$ only once its ad is clicked. We are interested in mechanisms satisfying the following properties:

Definition 1. *Mechanism $\mathcal{M}$ is dominant strategy incentive compatible (DSIC) if reporting true values is a dominant strategy for every agent (i.e., $\hat{v}_a = v_a$).*

Definition 2. *Mechanism $\mathcal{M}$ is individually rational (IR) if no agent acting truthfully prefers to abstain from participating to the mechanism rather than participating.*

Definition 3. *Mechanism $\mathcal{M}$ is weakly budget balance (WBB) if the mechanism is never in deficit.*

Definition 4. *Mechanism $\mathcal{M}$ is computationally tractable when both f and p_a are computable in polynomial time.*

Definition 5. *Allocation function f is maximal in its range if f returns an allocation maximizing the social welfare among a given subset of allocations that is independent of the agents' reports.*

When f is maximal in its range, VCG–like payments can be used, obtaining a DSIC, IR and WBB mechanism (Nisan and Ronen 2000).

Inefficiency of GSP and VCG with PDC

GSP analysis

In the GSP, ads are allocated according to decreasing reported values: supposing $\hat{v}_1 \geq \cdots \geq \hat{v}_N$, the allocation function maps ad a to slot $s = a$, for each $a = 1, \dots, K$. Every agent $a = 1, \dots, K - 1$ is charged a price $p_a = q_{a+1}\hat{v}_{a+1}$. Agent K is charged a price $p_K = 0$ if $K = N$, or $p_K = q_{K+1}\hat{v}_{K+1}$ otherwise. The following lemma holds:

Lemma 1. *The GSP is not IR, when users follow the APDC.*

The GSP is well known not to be DSIC. For this reason, we study the inefficiency of its Nash equilibria.

Lemma 2. *The PoA of the social welfare in the GSP when users follow APDC is unbounded.*

Lemma 3. *The PoA of the social welfare in the GSP when users follow APDC is $\geq K$, in the restricted case the agents do not overbid.*

Lemma 4. *The PoA of the revenue in the GSP when users follow APDC is unbounded (even without overbidding).*

Lemma 5. *The PoS of the social welfare in the GSP when users follow APDC is 1 (even without overbidding).*

Lemma 6. *The PoS of the revenue in the GSP when users follow APDC is 0 (even without overbidding).*

Lemma 2 and 3 together show a huge dispersion of the social welfare of the allocations associated with GSP equilibria: it may be arbitrarily larger than the social welfare of the APDC. Lemma 4 and 6 show that this situation gets even worse when we turn our attention onto the auctioneer's revenue, with the revenue of GSP equilibria being arbitrarily smaller or larger than that of the APDC.

VCG with PDC analysis

The PDC model is very similar to the APDC: the difference lies in the fact that $c_a = 1$ for each ad $a \in \mathcal{A}$. Payments are calculated according to the rules attaining the VCG mechanism, thus providing a IR, WBB, DSIC mechanism under the assumption that users experience ad fatigue due only to the positions of the slots and not to the allocated ads. When the user's behavior is affected by ad continuation probabilities, the above mentioned properties do not necessarily apply anymore. Indeed, the following lemma holds:

Lemma 7. *The VCG with PDC is neither IR nor DSIC, when users follow APDC.*

We study the inefficiency of Nash equilibria, VCG with PDC not being truthful.

Lemma 8. *The PoA of the social welfare in the VCG with PDC when users follow APDC is unbounded.*

Lemma 9. *The PoA of the social welfare in the VCG with PDC when users follow APDC is $\geq K$, in the restricted case the agents do not overbid.*

Lemma 10. *The PoA of the revenue in the VCG with PDC when users follow APDC is unbounded (even without overbidding).*

Lemma 11. *The PoS of the social welfare in the VCG with PDC when users follow APDC is 1.*

Lemma 12. *The PoS of the revenue in the VCG with PDC when users follow APDC is 0.*

Lemma 13. *The PoS of the revenue in the VCG with PDC when users follow APDC is ≤ 1, in the restricted case the agents do not overbid.*

Again, we see a huge dispersion, both in terms of social welfare and of revenue, among the different equilibria of VCG with PDC.

Algorithms

DOMINATED–ADS algorithm

We present an algorithm meant to reduce the size of the problem (i.e., the number of ads), without any loss in terms of social welfare or revenue of the optimal allocation. The central observation for our algorithm is that, under certain circumstances, given two ads a, b with parameters $(\bar{v}_a, c_a)$ and $(\bar{v}_b, c_b)$ respectively, it is possible to establish *a priori* that, if in an optimal allocation b is allocated to a slot, then ad a is allocated in a slot preceding that of b; whenever this is the case we say that ad a "dominates" ad b. As an example, consider two ads a and b, satisfying the condition $(\bar{v}_a > \bar{v}_b) \wedge (c_a > c_b)$: in accordance with intuition, a simple exchange argument shows that a dominates b. A weaker sufficient condition for deciding whether a dominates b is given in the following lemma and proved in (Farina and Gatti 2015).

Lemma 14. *Let $\lambda_{\max} = \max\{\lambda_s : s \in \mathcal{K}\}$, and B an upper bound*[1] *of the value of the optimal allocation; also, let $\mathcal{D} = [0, \lambda_{\max}] \times [0, B]$. Given two ads a, b with parameters $(\bar{v}_a, c_a)$ and $(\bar{v}_b, c_b)$, consider the affine function $w_{a,b} : \mathcal{D} \to \mathbb{R}$, defined as*

$$w_{a,b}(x, y) = \det \begin{pmatrix} x & -y & 1 \\ 1 & \bar{v}_b & c_b \\ 1 & \bar{v}_a & c_a \end{pmatrix}$$

If the minimum of $w_{a,b}$ over $\mathcal{D}$ is greater than 0, then a dominates b.

We will use the notation $a \prec b$ to denote that ad a dominates ad b, in the sense of Lemma 14. Note that $\prec$ defines a partial order over the set of ads. Since $w_{a,b}$ is an affine function defined on a convex set, it must attain a minimum on one of the corner points of $\mathcal{D}$, hence the following holds:

Lemma 15. *If the four numbers $w_{a,b}(0, 0)$, $w_{a,b}(0, B)$, $w_{a,b}(\lambda_{\max}, 0)$, $w_{a,b}(\lambda_{\max}, B)$ are all positive, then $a \prec b$.*

We define the *dominators* of an ad a as the set $\text{dom}(a) = \{b \in \mathcal{A} : b \prec a\}$. The following lemma is central to our algorithm, as it expresses a sufficient condition for discarding an ad from the problem:

Lemma 16. *If $|\text{dom}(a)| \geq K$, then ad a can be discarded, as it will never be chosen for an allocation.*

This suggests a straightforward algorithm to determine the set of safely deletable ads, as streamlined in Algorithm 1.

A naïve implementation of Line 3 tests every ad a against all the other ads, keeping track of the number of dominators of a. Therefore, neglecting the cost of computing $\mathcal{D}$ on Line 1, this results in a $O(N^2)$ time algorithm.

[1]More precisely, it is enough that B is a (weak) upper bound of the quantity

$$\tilde{B} = \max_i \{\lambda_i \cdot \text{ALLOC}(i + 1, K)\},$$

where $\text{ALLOC}(i, K)$ is the value of the optimal allocation of the problem having as slots the set $\mathcal{K}_i = \{i, \ldots, K\} \subseteq \mathcal{K}$.

Algorithm 1 DOMINATED–ADS

```
1: procedure DOMINATED–ADS(ads, slots)
2:     Determine $\mathcal{D}$, as defined in Lemma 14
3:     For each ad $a$, compute $|\text{dom}(a)|$
4:     Discard all ads $a$ having $|\text{dom}(a)| \geq K$
```

COLORED–ADS algorithm

We present an algorithm that can determine an optimal allocation in time polynomial in N and exponential in K. As the number of slots is generally small (typically ≤ 10), this algorithm is suitable for real–world applications. The key insight for the algorithm is that the problem of finding the allocation maximizing the social welfare can be cast to an instance of the well–known problem of finding a maximal K–vertex weighted simple path in an un undirected graph. Efficient algorithms for the latter problem can therefore be used to solve our problem.

We introduce the definition of a *right–aligned allocation*.

Definition 6. *Given an ordered sequence $S : a_1, \dots, a_n$ of ads, we say that the* right–aligned allocation *of S is the allocation mapping the ads in S to the last n slots, in order, leaving the first $K - n + 1$ slots vacant.*

Consider the complete undirected graph $\mathcal{G}$ having the available ads for vertices; every simple path $\pi : a_1, \dots, a_n$ of length $n \leq K$ in $\mathcal{G}$ can be uniquely mapped to the right–aligned allocation of π, and *vice versa*. Following this correspondence, we define the *value* of a simple path in $\mathcal{G}$ as the value of the corresponding right–aligned allocation. In order to prove that the problem of finding a maximal K–vertex weighted simple path in $\mathcal{G}$ is equivalent to that of finding an optimal allocation, it is sufficient to prove that there exists (at least) one optimal allocation leaving no slot vacant, i.e. using all of the K slots. To this end we introduce the following lemma:

Lemma 17. *In at least one optimal allocation all the available slots are allocated.*

The problem of finding a maximal K–vertex weighted simple path in $\mathcal{G}$ can be solved by means of the *color coding* technique (Alon, Yuster, and Zwick 1994). The basic idea is as follows: in a single iteration, vertices are assigned one of K random colors; then, the best path visiting every color exactly once is found using a dynamic programming approach; finally, this loop is repeated a certain amount of times, depending on the type of algorithm (randomized or derandomized). In the randomized version, the probability of having missed the best K–path decreases as e^{-R/e^K}, where R is the number of iterations. Therefore, using randomization, we can implement a $O((2e)^K N)$ compute time and $O(2^K + N)$ memory space algorithm, as streamlined in Algorithm 2. In the derandomized version, the algorithm is exact and requires $O((2e)^K K^{O(\log K)} N \log N)$ time when paired with the derandomization technique presented in (Naor, Schulman, and Srinivasan 1995).

Algorithm 2 COLORED–ADS

```
1: procedure COLORED-ADS(ads, slots)
2:     repeat $e^K \log 2$ times                    ▷ 50% success probability
3:         Assign random colors $\in \{1, \dots, K\}$ to $\mathcal{G}$'s vertices
           note: each color must be used at least once

         ▷ We now construct a memoization table MEMO$[\mathcal{C}]$, reporting,
         for each color subset $\mathcal{C}$ of $\{1, \dots, K\}$ the maximum value of
         any simple path visiting colors in $\mathcal{C}$ exactly once.

4:         MEMO$[\emptyset] \leftarrow 0$
5:         $\mathcal{P} \leftarrow$ powerset of $\{1, \dots, K\}$, sorted by set size
6:         for $\mathcal{C} \in \mathcal{P} - \{\emptyset\}$, in order do
7:             $\tilde{\lambda} \leftarrow \lambda_{K-|\mathcal{C}|+1}$
8:             for each color $c \in \mathcal{C}$ do
9:                 for each vertex (ad) $a$ in $\mathcal{G}$ of color $c$ do
10:                    VALUE $\leftarrow \bar{v}_a + c_a \tilde{\lambda} \cdot$ MEMO$[\mathcal{C} - \{c\}]$
11:                    MEMO$[\mathcal{C}] \leftarrow \max\{$MEMO$[\mathcal{C}]$, VALUE$\}$
12:    return MEMO$[\{1, \dots, K\}]$
```

Note that Algorithm 2 is just a simplified, randomized and non–parallelized sketch of the algorithm we test in the experimental section of this paper; also, for the sake of presentation, in Algorithm 2 we only return the value of the optimal allocation and not the allocation itself. We also considered the idea of using $1.3K$ different colors, as suggested by the work of (Hüffner, Wernicke, and Zichner 2008), but we found out that for this particular problem the improvement is negligible.

We conclude this subsection with some remarks about the above algorithm. First, we remark that, given the nature of the operations involved, the algorithm proves to be efficient in practice, with only a small constant hidden in the big–oh notation. Furthermore, it is worth to note that the iterations of the main loop (Lines 2 to 15) are independent; as such, the algorithm scales well horizontally in a parallel computing environment. Second, we point out an important economic property of the algorithm, that makes COLORED–ADS appealing: it allows the design of truthful mechanisms when paired with VCG–like payments. While this is obviously true when the algorithm is used in its derandomized exact form, it can be proven that the truthfulness property holds true even when only a partial number of iterations of the main loop is carried out. This easily follows from the fact that the algorithm is maximal–in–its–range, searching for the best allocation in a range that does not depend on the reports of the agent. This implies that it is possible to interrupt the algorithm after a time limit has been hit, without compromising the truthfulness of the mechanism. This leads to approximate algorithms that offer convenient time–approximation trade–offs, as long as K is small.

SORTED–ADS approximate algorithm

While the general problem of finding an optimal allocation is difficult, polynomial time algorithms are easy to derive—as we show below—when we restrict the set of feasible allocations to those respecting a given total order $\prec_{\text{ads}}$ defined

Algorithm 3 SORTED–ADS

```
1: procedure SORTED-ADS(ads, slots, $\prec_{\text{ads}}$)
2:     Sort the ads according to $\prec_{\text{ads}}$, so that ad 1 is the minimum ad w.r.t. the given order
       ▷ We now construct a memoization table $\mathcal{T}[n,k]$, reporting, for each $1 \leq n \leq N$ and $1 \leq k \leq K$, the value of the best allocation that uses only ads $n,\dots,N$ and slots $k,\dots,K$ (i.e. no ads get allocated to slots $1,\dots,k-1$ and $\lambda_i = 1, \forall i < k$).
3:     $\mathcal{T}[N,k] \leftarrow \bar{v}_N, \quad k = 1,\dots,K$        ▷ Base case
4:     for $n = N-1$ downto $1$ do
5:         for $k = 1,\dots,K$ do
6:             if $k < K$ then
7:                 VALUE $\leftarrow \bar{v}_n + \lambda_k\, c_n\, \mathcal{T}[n+1,k+1]$
8:                 $\mathcal{T}[n,k] \leftarrow \max\{\text{VALUE}, \mathcal{T}[n+1,k]\}$
9:             else
10:                $\mathcal{T}[n,k] \leftarrow \max\{\bar{v}_n, \mathcal{T}[n+1,K]\}$
11:    return $\mathcal{T}[1,1]$
```

on the ads set. This suggests this simple approximation algorithm: first, T total orders $\prec_{\text{ads},1},\dots,\prec_{\text{ads},T}$ over the ads set are chosen; then, the optimal allocation satisfying $\prec_{\text{ads},i}$ is computed, for each i; finally, the value of the optimal allocation for the original unrestricted problem is approximated with the best allocation found over all the T orders. The number of total orders T is arbitrary, with more orders obviously producing higher approximation ratios.

In order to find the optimal allocation respecting the total order $\prec_{\text{ads}}$, we propose a simple $O(NK)$ time dynamic programming algorithm, which we name SORTED–ADS, described in Algorithm 3. The idea behind the algorithm is to find, for each $n = 1,\dots,N$ and $k = 1,\dots,K$, the value $\mathcal{T}[n,k]$ of the best allocation for the problem having $\mathcal{A}_n = \{1,\dots,n\} \subseteq \mathcal{A}$ as ads set and $\mathcal{K}_k = \{k,\dots,K\} \subseteq \mathcal{K}$ as slots set. The values of $\mathcal{T}[n,k]$ can be computed inductively, noticing that the associated subproblems share the same optimal substructure. As before, in Algorithm 3 we only show how to find the value of the optimal allocation, and not the allocation itself.

We end this subsection with the analysis of some economical and theoretical properties of the algorithm. We note that, if the total orders used do not depend on the reported types of the agents, SORTED–ADS is maximal–in–its–range, leading thus to a truthful mechanism when paired with VCG–like payments. Furthermore, the resulting mechanism requires polynomial time both for the allocation and the payments.

Experimental evaluation

For a better comparison of the results, we adopted the same experimental setting used in (Gatti and Rocco 2013) and given to us by the authors. We briefly describe it, details can be found in the original paper. The experimental setting is based on *Yahoo! Webscope A3* dataset. Each bid is drawn from a truncated Gaussian distribution, where the mean and standard deviation are taken from the dataset, while quality is drawn from a beta distribution. The values of λ_s of the first 10 slots are $\{1.0, 0.71, 0.56, 0.53, 0.49, 0.47, 0.44, 0.44, 0.43, 0.43\}$. We considered two scenarios, one having $K = 5$ and one having $K = 10$. In both cases we let $N \in \{50, 60, \dots, 100\} \cup \{200, 300, \dots, 1000\}$. For each pair (K, N), 20 instances were generated. We implemented our algorithms in the C++11 language and executed them on the OSX 10.10.3 operating system. The main memory was a 16GB 1600MHz DDR3 RAM, while the processor was an Intel Core i7–4850HQ CPU. We compiled the source with GNU g++ version 4.9.1. Parallelization was achieved using OpenMP version 4.0.

DOMINATED–ADS algorithm

We study the average number of ads that survive DOMINATED–ADS in Figure 1. The upper bounding strategy needed in Lemma 14 was implemented using a $O(NK)$ time algorithm.

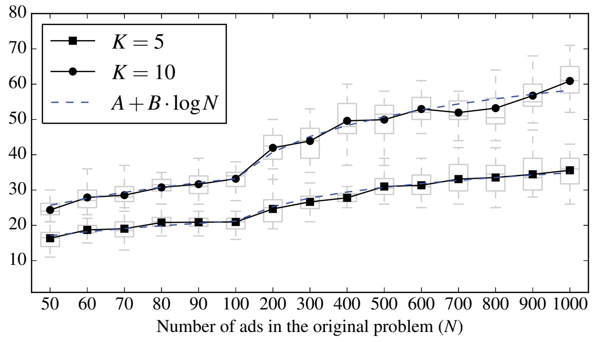

Figure 1: Average number of ads after running the DOMINATED–ADS algorithm.

Notice that the number of removed ads already considerable when N is small (e.g., $N \approx 50$), and that the prune ratio increases dramatically as N grows (for instance, the prune ratio is approximately 96% on average when $N = 1{,}000$ and $K = 5$). Experiments show that the number of surviving ads is of the form $\tilde{N} = A + B \cdot \log N$ for suitable values[2,3] of A and B.

In Figure 2 we report the average running time for DOMINATED–ADS. The graph shows that the running time depends quadratically on the original problem size N, but it remains negligible (around 20ms) even when $N = 1{,}000$.

[2]For $K = 5$, we have $A \approx -6.1, B \approx 5.9$ and $R^2 > 0.98$, where R^2 is the coefficient of determination of the fit.

[3]For $K = 10$, we have $A \approx -16.9, B \approx 10.9$ and $R^2 > 0.98$.

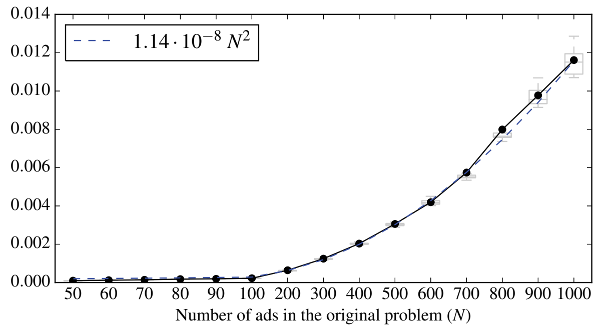

Figure 2: Avg. DOMINATED–ADS run. time [s]. This is independent of K and grows as $\approx 1.14 \cdot 10^{-8} N^2$ ($R^2 > 0.99$).

COLORED–ADS algorithm

Figure 3 shows the average running time for COLORED–ADS, implemented in its randomized version. The number of iterations was set to $e^K \log 2$, i.e. 50% probability of finding the optimal allocation. The algorithm is run on the reduced instances produced by DOMINATED–ADS. As a result of the shrinking process, we point out that the running times for $N = 50$ and $N = 1{,}000$ are comparable. We also remark that, in order for COLORED–ADS to be applicable in real–time contexts when $K = 10$, some sort of hardware scaling is necessary, as a running time of ≈ 0.5 seconds per instance is likely to be too expensive for many practical applications. When $K = 5$, though, no scaling is necessary, and the algorithm proves to be extremely efficient.

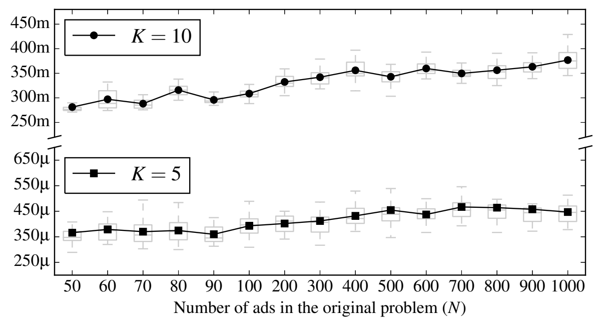

Figure 3: Avg. COLORED–ADS run. time [s] ($e^K \log 2$ runs).

SORTED–ADS algorithm

In figure 4 we study the approximation ratio of SORTED-ADS. Optimal allocation values were computed using an exponential branch–and–bound algorithm, similar to that of (Gatti and Rocco 2013). For each shrunk problem instance, we generated $2K^3$ random orders, and used SORTED–ADS to approximate the optimal allocation. Surprisingly, even though the number of iterations is relatively low, approximation ratios prove to be very high, with values $> 97\%$ in the worst case, with both medians and means always $> 99\%$. Finally, in Figure 5 we report the corresponding computation times. These prove to be in the order of a handful of milliseconds, thus making SORTED–ADS suitable in real-time contexts even for a large number of ads.

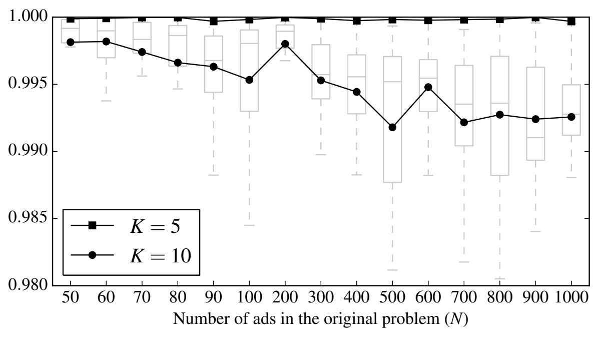

Figure 4: Avg. SORTED–ADS approx. ratio ($2K^3$ orders).

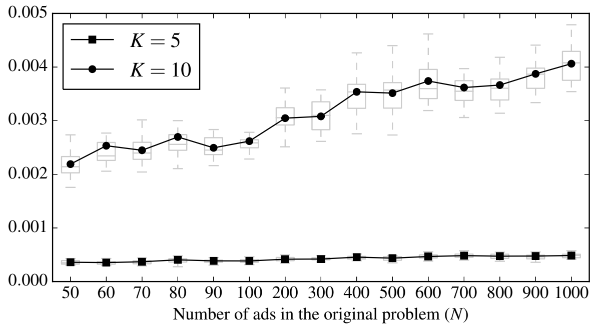

Figure 5: Avg. SORTED–ADS running time [s] ($2K^3$ orders).

Conclusions and future work

In this paper, we provide several contributions about the ad/position dependent cascade model for sponsored search auctions. This model is commonly considered the one providing the best tradeoff between accuracy and computational complexity, but no algorithm suitable for real–world applications is known. Initially, we study the inefficiency of GSP and VCG with the only position dependent cascade model w.r.t. the VCG with the ad/position dependent cascade model, analyzing PoA and PoS both of the social welfare and the auctioneer's revenue. Our results show that the inefficiency is in most cases unbounded and that the dispersion over the equilibria is extremely large, suggesting that the GSP, as well as the VCG with position dependent cascade model, presents severe limits. Subsequently, we provide three algorithms and we experimentally evaluate them with a real–world dataset. Our first algorithm reduces the size of the instances, discarding ads. Empirically, the number of non–discarded ads is logarithmic. Our second algorithm finds the optimal allocation with high probability, requiring a short time (less than 1ms when $K = 5$ even with $1{,}000$ ads), but too long for real–time applications when $K = 10$. Our third algorithm approximates the optimal allocation in very short time (less than 5ms even with $1{,}000$ ads) providing very good approximations (>0.98). This shows that the ad/dependent cascade model can be effectively used in practice.

In future, we will extend our analysis and our algorithms to models in which there is a contextual graph among the ad, and in which there are multiple slates.

References

Alon, N.; Yuster, R.; and Zwick, U. 1994. Color-coding: A new method for finding simple paths, cycles and other small subgraphs within large graphs. In *ACM STOC*, 326–335.

Craswell, N.; Zoeter, O.; Taylor, M.; and Ramsey, B. 2008. An experimental comparison of click position–bias models. In *WSDM*, 87–94.

Evangelos, E., and Telelis, O. 2010. Discrete strategies in keyword auctions and their inefficiency for locally aware bidders. In *WINE*, 523–530.

Farina, G., and Gatti, N. 2015. Ad auctions and cascade model: GSP inefficiency and algorithms. *ArXiv e-prints, 1511.07397*.

Fotakis, D.; Krysta, P.; and Telelis, O. 2011. Externalities among advertisers in sponsored search. In *SAGT*, 105–116.

Gatti, N., and Rocco, M. 2013. Which mechanism in sponsored search auctions with externalities? In *AAMAS*, 635–642.

Gatti, N.; Rocco, M.; Serafino, P.; and Ventre, C. 2015. Cascade model with contextual externalities and bounded user memory for sponsored search auctions. In *AAMAS*, 1831–1832.

Gomes, R.; Immorlica, N.; and Markakis, E. 2009. Externalities in keyword auctions: An empirical and theoretical assessment. In *WINE*, 172–183.

Hegeman, J. 2010. Facebook's ad auction. *Talk at Ad Auctions Workshop*.

Hüffner, F.; Wernicke, S.; and Zichner, T. 2008. Algorithm engineering for color-coding with applications to signaling pathway detection. *Algorithmica* 52(2):114–132.

IAB. 2015. IAB internet advertising revenue report. 2014 full year results.

Joachims, T.; Granka, L.; Pan, B.; Hembrooke, H.; Radlinski, F.; and Gay, G. 2007. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems* 25(2):7.

Kempe, D., and Mahdian, M. 2008. A cascade model for externalities in sponsored search. In *WINE*, 585–596.

Kuminov, D., and Tennenholtz, M. 2009. User modeling in position auctions: re-considering the GSP and VCG mechanisms. In *AAMAS*, 273–280.

Leme, R. P., and E. Tardos, E. 2010. Pure and Bayes–Nash price of anarchy for generalized second price auction. In *FOCS*, 735–744.

Naor, M.; Schulman, L. J.; and Srinivasan, A. 1995. Splitters and near-optimal derandomization. In *FOCS*, 182–191.

Narahari, Y.; Garg, D.; Narayanam, R.; and Prakash, H. 2009. *Game Theoretic Problems in Network Economics and Mechanism Design Solutions*. Springer.

Nisan, N., and Ronen, A. 2000. Computationally feasible VCG mechanisms. In *ACM EC*, 242–252.

Varian, H. R., and Harris, C. 2014. The VCG auction in theory and practice. *American Economic Review* 104(5):442–445.