The Ad Types Problem

Riccardo Colini-Baldeschi[1], Julián Mestre[2], Okke Schrijvers[1], and Christopher A. Wilkens[3]

[1] Core Data Science, Facebook Inc.
[2] School of Computer Science, University of Sydney.
[3] Tremor Technologies

Abstract. In this paper we introduce the *Ad Types Problem*, a generalization of the traditional positional auction model for ad allocation that better captures some of the challenges that arise when ads of different types need to be interspersed within a user feed of organic content.
The Ad Types problem (without gap rules) is a special case of the assignment problem in which there are k types of nodes on one side (the ads), and an ordered set of nodes on the other side (the slots). The edge weight of an ad i of type θ to slot j is $v_i \cdot \alpha_j^\theta$ where v_i is an advertiser-specific value and each ad type θ has a discount curve $\alpha_1^{(\theta)} \geq \alpha_2^{(\theta)} \geq ... \geq 0$ over the slots that is common for ads of type θ. We present two contributions for this problem: 1) we give an algorithm that finds the maximum weight matching that runs in $O(n^2(k + \log n))$ time for n slots and n ads of each type—cf. $O(kn^3)$ when using the Hungarian algorithm—, and 2) we show how to apply reserve prices in total time $O(n^3(k + \log n))$.
The Ad Types Problem (with gap rules) includes a matrix G such that after we show an ad of type θ_i, the next G_{ij} slots cannot show an ad of type θ_j. We show that the problem is hard to approximate within $k^{1-\epsilon}$ for any $\epsilon > 0$ (even without discount curves) by reduction from Maximum Independent Set. On the positive side, we show a Dynamic Program formulation that solves the problem (including discount curves) optimally and runs in $O(k \cdot n^{2k+1})$ time.

1 Introduction

Feeds aggregate a variety of content into a one-stop source of information. In order to present content in a way that maximizes engagement, state-of-the-art feeds like Facebook's News Feed, reddit, and Apple News must consider not only the users independent interest in each item but also the position in the feed and the relative order of items. Optimizing ad placement in these platforms presents similar challenges; to capture some of them, we introduce a generalization of the canonical position auction known as the *Ad Types Problem.*

Position auction [39, 11] is the default mechanism for simultaneously selecting multiple ads. A standard position auction is simple: rank ads according to their expected advertising value and rank slots according to their prominence (position in the feed); the highest-value ad then appears in the most prominent slot and so on until all slots are filled. Formally, the auction maximizes value using a separable value model that combines baseline value for each ad with a position discount for each slot $1 \geq \alpha_1 \geq .. \geq 0$ capturing the decay in value associated with lower-prominence.

Content feeds bring two important complexities that violate the simple separable model: ads are not homogeneous, and spacing matters. Firstly, in the same way that a feed aggregates many types of content, a feed may simultaneously include ads in many formats, including text, images, and video. Advertisers may also have different objectives – some advertisers only want users to see a static image, while other advertisers want users to finish a video or visit their site and make a purchase. Prominence impacts every type differently – for example, a user who has already scrolled deep into a content feed will still see an image ad but may be less likely to watch a video ad to completion. Secondly, spacing matters, since a user who (say) sees two video ads in a row may be less likely to view the second video ad (or simply be annoyed).

Given these complexities, naïvely implementing a position auction using a traditional separable model will be suboptimal. The following example illustrates the problem when the probability of a user watching a video ad decays differently than a link-click ad:

Example 1. Suppose we have a setting with 2 ad types, link-click ads and video ads, two ad slots, and we have discount curve $\alpha_1 = \frac{1}{2}, \alpha_2 = \frac{1}{4}$. These discounts are accurate for link-click ads (i.e. $\alpha_1^{(\text{link})} = \frac{1}{2}, \alpha_2^{(\text{link})} = \frac{1}{4}$), but for video ads, the user is more likely to watch the video in the second slot than they are to click a link in that slot: $\alpha_1^{(\text{video})} = \frac{1}{2}$, $\alpha_2^{(\text{video})} = \frac{1}{3}$.

Consider a video ad with bid \$12 and a link-click ad with bid \$10. The optimal allocation assuming that discount curve α is accurate for both ads would assign the video ad to slot 1 and the link-click ad to slot 2 for total value $\frac{1}{2} \cdot \$12 + \frac{1}{4} \cdot \$10 = \$8.50$. However, switching the ads yields total value $\frac{1}{2} \cdot \$10 + \frac{1}{3} \cdot \$12 = \$9 > \$8.50$.[4]

In this paper we propose a new theoretical model for online advertising that addresses these issues. It captures the position auction as a special case, but can handle discount curves for multiple types and intersperse advertising with organic content in a dynamic manner.[5] An *Ad Types Problem* instance has k ad types[6], that each have their own discount curve over n slots, i.e. all ads of type θ

[4] Note that this example also implies that VCG prices w.r.t. α would not be incentive compatible.

[5] While our motivation for studying this problem comes from online advertising in content streams, it captures many other interesting settings that are unrelated to online advertising. For example, the setting without gap rules can model a worker with different time slots and jobs of different types that need to be done; jobs are most valuable when completed early and delays for jobs of the same type are discounted similarly. Adding gap rules can model the cost of moving between locations (in the physical world) or context-switching (in the digital world).

[6] In economics literature, "type" sometimes refers to private information. That's not the case here: type represents the content type, e.g. video or link-click, and is publicly known.

have discount curve $\alpha_1^{(\theta)} \geq \alpha_2^{(\theta)} \geq \ldots \geq \alpha_n^{(\theta)} \geq 0$ that represents the slot-specific action-rates. All ad types agree on the order of the slots. Gap rules are modeled by a $k \times k$ matrix G, which indicates for each pair of ad types (θ_i, θ_j), that after showing an ad of type θ_i, the next G_{ij} stories cannot be of type θ_j.

We first focus on the special case where $G = \mathbf{0}$, i.e. different ad types have different discount curves but there are no constraints on the gaps between ads. In this setting, the Ad Types Problem is a special case of the maximum-weight bipartite matching problem (also known as the assignment problem), so we could find an optimal allocation using the Hungarian algorithm in $O(kn^3)$ time [36] (where k is the number of types, n the number of slots, and we have n ads per type). Our first result is an algorithm that finds the optimal allocation in $O(n^2(k + \log n))$ time, saving a linear factor. We also show that we can compute incentive-compatible prices with advertiser-specific reserve prices for all ads in $O(n^3(k + \log n))$ time.

Next we consider the more general Ad Types problem with both discount curves and gap rules (where $G \neq \mathbf{0}$). We show that the problem is hard to approximate within $k^{1-\epsilon}$ for any $\epsilon > 0$ (even without discount curves) by reduction from Maximum Independent Set. On the positive side, we show a Dynamic Program formulation that solves the problem (including discount curves) optimally and runs in $O(k \cdot n^{2k+1})$ time, which is a significant improvement over the brute-force running time of $O(k^n)$ since typically $k \ll n$.

1.1 Related Work

Assignment Problem. The maximum-weight bipartite matching problem, also known as the assignment problem, is a classical problem in operations research. Let (A, B, E) be a complete bipartite graph with edge weights $v : E \to \mathbb{R}^+$, and $V = A \cup B$ the set of nodes; the goal is to find a matching M of maximal total weight $\sum_{e \in M} v(e)$. Kuhn [28] proposed an algorithm for this problem—which he called the Hungarian algorithm—based on ideas by Kőnig and Egerváry, though he only proved that the algorithm would terminate, not what the time complexity is. Munkres [31] showed that the time complexity of the Hungarian algorithm is $O(|V|^4)$. Edmonds and Karp [12] gave an $O(|V|^3)$ time algorithm for balanced graphs, and Ramshaw and Tarjan [36] more recently gave an algorithm for unbalanced graphs (without loss of generality, assume $|A| < |B|$) that runs in $O(|E||A| + |A|^2 \log |A|)$. Since the seminal work on the assignment problem, there has been active research into relevant special cases. In particular there is a line of work on *convex bipartite graphs*, where the right side of the graph is ordered, and nodes on the left can only be connected to a single contiguous block of nodes on the right. For the unweighted case, a line of work starting with Glover [21, 29, 16] shows that the problem can be solved in time linear in the number of nodes $O(|V|)$. General weights are not considered, though early work on *vertex-weighted* bipartite graphs (where each node i has an associated weight w_i and the weight of an edge from i to j is $w_{ij} = w_i + w_j$) yield an $O(|E| + |B| \log |A|)$ time algorithm [25]. More recently, Plaxton [34, 35] showed that Two-Directional Orthogonal

Ray Graphs (a generalization of convex graphs) admit an $O(|V|\log|V|)$ time algorithm.

Sharathkumar and Agarwal [38] consider a more general set of edge weights, where nodes are embedded in d-dimensional space, and the weights of the complete bipartite graph are all either the L_1 or L_∞ metric. They give an algorithm to solve maximum weight bipartite matching in $O(|V|^{3/2}\log^{d+O(1)}(|V|)\log\Delta)$, where Δ is the diameter of the space that contains the points.

None of the results on specializations cover The Ad Types Problem setting (even without gap rules).

Ad Auctions. The simple separable model for position auctions appears in Varian [39] and Edelman et al. [11]. One body of related work relaxes the assumption that action rates are separable. One common theme is to model externalities between ads (also related to our gap rules) [26, 18, 20, 3, 2, 22, 19, 14, 17]. Of note, [26, 18, 13, 2] study algorithms for computing allocations in models where the user's attention cascades and prove hardness results. A different generalization is to allow arbitrary action rates that are still independent between ads [1, 6, 5], which corresponds to the Ad Type Problem (without gap rules) where each ad has a unique type.

Another generalization of the basic position auction allows ads to be placed in complex ways. A few papers study mechanisms that permit presentation constraints and/or ads with variable presentation [24, 4, 23, 32, 8]. Mahdian et al. study auctions for ads displayed on maps along with organic results [30] (since places of interest are connected to a physical location, this imposes constraints on where ads can be placed).

Finally, the connections between ad auctions and max-weight matching (and the Hungarian algorithm) have been studied before as well [9, 5, 27, 10].

1.2 Contributions

This paper presents three main contributions:

– **Optimal Allocation.** Firstly, we give an algorithm to optimally solve the Ad Types problem *without* gap rules. This setting is a special case of the assignment problem with applications beyond ad auctions. Our algorithm is a specialization of the Hungarian algorithm to find the maximum-weight matching in the bipartite graph that uses the structure of the Ad Types Problem to run in $O(n^2(k+\log n))$ time (compared to $O(kn^3)$ for running the Hungarian algorithm on the instance; Theorem 1).
– **Pricing.** Secondly, we show that we can do incentive-compatible pricing in this setting with minimal overhead. First, we show that we can apply reserve prices (and in fact in all single-parameter environments) without a change-point algorithm [33, 37]. For our case, this yields an $O(n^3(k+\log n))$ time algorithm. We also confirm that—similar to the general bipartite matching case—the dual variables in our algorithm for the Ad Types setting (without

reserves) can be used to recover VCG prices without increasing the asymptotic running time. This yields VCG prices in $O(n^2(k + \log n))$. Due to lack of space these results are deferred to the full verison of this paper [7].

– **Gap Rules.** Finally, we consider the more general Ad Types problem with both discount curves and gap rules (where $G \neq \mathbf{0}$). We show that the problem is hard to approximate within $k^{1-\epsilon}$ for any $\epsilon > 0$ (even without discount curves) by reduction from Maximum Independent Set (Theorem 2). On the positive side, we give a Dynamic Program formulation that solves the problem (including discount curves) optimally and runs in $O(k \cdot n^{2k+1})$ time (Theorem 3) which is a significant improvement over the brute-force running time of $O(k^n)$ since typically $k \ll n$.

2 Preliminaries

In this section we give a formal definition of the Ad Types problems and with it, the notation that we will be using throughout the paper. Our results build on the known results from Auction Theory and the Hungarian Algorithm for solving the assignment problem.

The Ad Types Problem involves computing an allocation of a set of N ads to $n \leq N$ slots. Ads come in one of k different ad types θ_l, for $l \in \{1, ..., k\}$. We let the ads of type θ_l be $a_i^{(\theta_l)}$ for $i \in 1, \ldots, n_l$, where n_l represents the number of ads of type θ_l. There are three main components to the definition of the problem:

– **Valuations.** Ad i of type θ has a value-per-conversion (a.k.a. value-per-action) $v_i^{(\theta)}$. Ads of different types have different conversion events, e.g. for a display ad the conversion event is a view, for a link ad the conversion event is a link click, and for a video ad the conversion event is the user watching video ad. For each ad type θ, we index the ads in non-increasing order of valuation, i.e. $v_1^{(\theta)} \geq v_2^{(\theta)} \geq \ldots \geq v_{n_l}^{(\theta)} \geq 0$.

– **Discount curves.** We assume a separable model for discount curves where we can write

$$\Pr[\text{conversion on ad } i \text{ (of type } \theta) \text{ in slot } j] = \alpha_j^\theta \cdot \beta_i$$

where α_j^θ is the slot effect for a particular ad type θ (e.g., the probability that a user will watch a video ad if it is shown in the jth slot) and β_i is the advertiser quality (this separable model is also standard in position auctions [39, 11]). In the remainder of the paper we assume without loss of generality that the advertiser effect has already been included in the advertiser's value, i.e., if the value-per-conversion of the advertiser is v_i', then $v_i = \beta_i \cdot v_i'$. We further abuse notation to let $v_{ij} = \alpha_j^\theta \cdot v_i$ for ad i of type θ in slot j.
Discounts are monotonically non-increasing, and all ad types agree on the order of slots, i.e. for each ad type θ, we have $\alpha_1^{(\theta)} \geq \alpha_2^{(\theta)} \geq \ldots \geq \alpha_n^{(\theta)} \geq 0$.

– **Gap rules.** When ads are interspersed with organic content, there must be some way to control how many ads are shown. In the simplest case, where there's only one type of ad, this can be implemented by a gap rule g, which states that two ads must be at least g slots apart from each other. When there are multiple ad types, there is a $k \times k$ matrix G, which indicates for each pair of ad types (θ_i, θ_j), that after showing an ad of type θ_i, the next G_{ij} stories cannot be of type θ_j.

The Ad Types Problem is to find a social welfare maximizing allocation that obeys the gap rules.

2.1 The Hungarian Algorithm

The Hungarian Algorithm [28, 31] is a classical algorithm for computing a maximum weight matching in a bipartite graph. Starting from a trivial primal solution (empty matching) and a trivial dual solution, the algorithm iteratively increases the cardinality of the matching while improving the value of the dual solution until the value of the primal solution equals that of the dual.

Let (U, V, E) be a complete bipartite graph with edge weights $v : E \to \mathbb{R}^+$. The primal/dual pair of linear programs capturing the problem are as follows.

$$\begin{aligned} &\text{maximize } \textstyle\sum_{(i,j)\in E} v_{ij} x_{ij} \\ &\text{subject to } \quad \begin{aligned} \textstyle\sum_j x_{ij} &\le 1 && \forall i \in U \\ \textstyle\sum_i x_{ij} &\le 1 && \forall j \in V \\ x_{ij} &\ge 0 && \forall (i,j) \in E \end{aligned} \end{aligned}$$

$$\begin{aligned} &\text{minimize } \textstyle\sum_{i\in U} u_i + \sum_{j\in V} p_j \\ &\text{subject to } \quad \begin{aligned} u_i + p_j &\ge v_{ij} && \forall (i,j) \in E \\ u_i &\ge 0 && \forall i \in U \\ p_j &\ge 0 && \forall j \in V \end{aligned} \end{aligned}$$

The algorithm starts from an empty primal solution $M = \emptyset$, and a trivial feasible dual solution $u_i = 0$ for all $i \in U$ and $p_j = \max_{(i,j)\in E} v_{ij}$ for all $j \in V$. In each iteration, the algorithm identifies the set of tight edges $T = \{(i, j) \in E : u_i + p_j = v_{ij}\}$ and builds an alternating BFS tree B (also known as Hungarian tree) in (U, V, T) out of the free vertices in V. If the alternating tree contains an augmenting path A, we augment M with A thus increasing its cardinality; if no such path is available, we can update the dual solution by reducing the dual value of $V \cap B$ and increasing the dual value of $U \cap B$ by the same amount until a new edge becomes tight. This update maintains feasibility while reducing the value of the dual solution and makes at least one new edge tight, which in turn allows us to grow the alternating tree further.

Throughout the execution of the algorithm we maintain the invariants that the dual solution is feasible and that the edges in the matching M are tight. As a result, at the end of the algorithm we have a matching whose weight equals the value of the dual feasible solution, which acts as a certificate of its optimality.

Using the right data structures, it is possible to implement the algorithm so that the amount of work done between each update to M is $O(|E| + |U| \log |U|)$. Therefore, if we let M^* be a maximum weight matching, then the Hungarian algorithm can be implemented to run in $O(|M^*|(|E| + |U| \log |U|))$ time [15].

Algorithm 1 provides the full pseudo-code of the Hungarian Algorithm applied to the Ad Types problems.

Algorithm 1 Hungarian algorithm for the Ad Types problem.

```
Input: Values $v_1^{(\theta)} > v_2^{(\theta)} > ... > 0$, and
   discounts $\alpha_1^{(\theta)} > \alpha_2^{(\theta)} > ... > 0$ for each ad type $\theta$.
Output: Matching $M$ that maximizes $\sum_{(i,j) \in M} v_{ij}$.
 1: Initialize the dual solution so that
       ◦ $u_i \leftarrow 0$ for all ads $i$,
       ◦ $p_j \leftarrow \max v_{i',j'}$ for all slots $j$.
 2: Let $M \leftarrow \emptyset$ be the matching.
 3: for slot $j$ in descending order do
 4:    Let $B \leftarrow \{j\}$ be an alternating BFS tree
 5:    Let $P$ be an empty priority queue
 6:    $P \leftarrow$ UpdatePossibleNewEdges$(P, v, \alpha, M, j)$
 7:    while $B$ does not contain an augmenting path do
 8:       $(i', j') \leftarrow$ remove from $P$ next tight edge
 9:       $\Delta \leftarrow v_{i'j'} - u_{i'} - p_{j'}$        // note that $\Delta$ could be 0
10:       Implicitly update the dual solution so that
             ◦ $u_{i''} \leftarrow u_{i''} + \Delta$ for all ads $i'' \in B$,
             ◦ $p_{j''} \leftarrow p_{j''} - \Delta$ for all slots $j'' \in B$.
11:       if $i'$ is matched in $M$ then
12:          $B \leftarrow B \cup \{(i', j'), (i', M(i'))\}$
13:          UpdatePossibleNewEdges$(P, v, \alpha, M(i'))$
14:       else
15:          $B \leftarrow B \cup \{(i', j')\}$        // now an augmenting path exits
16:       end if
17:    end while
18:    $A \leftarrow$ augmenting path in $B$
19:    $M \leftarrow$ AugmentMatching$(M, A)$.
20:    explicitly update the dual solution $(u, p)$
21: end for
```

3 Ad Types Problem without Gap Rules

In this section we consider the ad types problem with discount curves but no gap rules. In this model we have k ad types, and each ad type has its own monotonically decreasing discount curve $\alpha_j^{(\theta_l)}$ for $l \in 1, 2, \ldots, k$. Without gap rules, the problem becomes a simple maximum weight bipartite assignment on a complete graph with N vertices (ads) on one side of the bipartition and n

vertices (slots) on the other side of the bipartition, with $n < N$. Therefore, the Hungarian algorithm can solve this problem in $O(Nn^2)$ time. We will assume throughout there are exactly n ads of each type[7], hence the Hungarian algorithm runs in $O(kn^3)$ time.

In this section we start by giving an algorithm that finds the maximum-weight bipartite matching in $O(n^2(k + \log n))$ time (Section 3.1). We show that in some sense the dependency on k unavoidable: namely if $k = n$, we show that the Ad Types problem reduces to the assignment problem (i.e. monotonicity and a common order of the slots does not improve the running time; see Section 3.2).

3.1 Finding the Optimal Allocation

We present an adaptation of the Hungarian algorithm [28, 31] that exploits the special structure of the Ad Types problem. In the following we use the language of markets to describe the Hungarian algorithm: the dual variable of a slot j corresponds to a price p_j, while a dual variable of an advertiser i corresponds to the utility u_i of the advertiser if they get an item out of their demand set (given the prices) [10]. Moreover, the instance is a complete bipartite graph with ads on one side and slots on the other side where the weight of the edge (i, j) is v_{ij}. The maximum-weight matching in the bipartite graph corresponds to the social-welfare maximizing allocation of ads to slots. For ease of exposition, we assume that values and discounts are monotonically *strictly* decreasing, this restriction can be lifted by consistent tie-breaking.

Algorithm 1 in the preliminaries shows how to compute the optimal allocation in an Ad Types instance using the Hungarian Algorithm. Our approach is to implement more efficiently how we maintain the set of possible new edges in Lines 6 and 13. The algorithm initializes the dual solution (u, p) to be feasible, and starts with an empty matching M. Algorithm considers slots in descending order in each iteration of the for loop in Line 3; we call each such iteration a *phase.*

During each phase we iteratively update the dual variables until we find an augmenting path to increase the size of the matching M by one. In each of these iterations within a phase we explore a tight edge leading to a matched edge and both edges are added to our alternating tree. Every time we add a new matched slot j' to the alternating tree we explore the edges incident on j' using the routine UpdatePossibleNewEdges, which scans the edges incident on j' and works out which edges are tight and when the remaining edges will become tight. All these new edges are stored in a priority queue for later retrieval.

High-level analysis approach. Even though the algorithm is not fully defined yet (the implement of UpdatePossibleNewEdges is given in the next subsection), still we can say something about the running time of the algorithm.

[7] If an ad type has fewer than n ads, we can append ads with value 0, if there are more than n ads of a type, with loss of generality we can restrict attention to the n highest-value ads.

Each phase is implemented using a priority queue P over some of the ads not in B. For each ad i' in P we keep track of the next edge (i', j') that would become tight given the current structure of B. The priority of i' captures *when* this next edge becomes tight, the smaller the priority the sooner it becomes tight; similarly, if i' already has a tight edge incident on itself then it should have the smallest priority in the queue.

In the normal implementation of the Hungarian Algorithm, the procedure UPDATEPOSSIBLENEWEDGES(P, v, α, j') iterates over all edges (i', j') incident on j'. If $i' \in B$ we can ignore the edge as i' has already been discovered and its slack $v_{i',j'} - p_{j'} - u_{i'}$ will not change with future updates (since now both i' and j' belong to B). If $i' \notin B$ then we compute its current slack $v_{i',j'} - p_{j'} - u_{i'}$ to work out when it will become tight and compare this against the time of the current next tight edge incident on i', which we may need to update.

Without making any assumptions on the structure of the valuations, in the worst case in each iteration of the while loop in Line 7 we perform $O(nk + \log nk) = O(nk)$ work (assuming a Fibonacci heap implementation for P) since there are kn ads in total and kn edges incident on j' (one per ad). In each iteration be grow B by adding one new matched edge, so we have at most j iterations of the while loop. Therefore, the overall running time is $O(\sum_{j=1}^{n} jnk) = O(n^3k)$.

However, as we shall see shortly, we can come up with a more efficient implementation of UPDATEPOSSIBLENEWEDGES(P, v, α, j') that exploits the special structure of our valuation function so that P holds at most $n + k$ ads and only $O(k)$ edges incident on j' need to be scanned without sacrificing the overall correctness of the algorithm. With this improvement in performance, each iteration of the while loop in Line 7 takes at most $O(\log n + k)$ work. Again, since in each iteration be grow B by adding one new matched edge, we have at most j iterations of the while loop. Therefore, the overall running time is $O(\sum_{j=1}^{n} j(k + \log n)) = O(n^2(k + \log n))$.

Theorem 1. *Given an input with k ad types and n slots, Algorithm 1 can be implemented to run in time $O(n^2(k + \log n))$.*

Our goal for the the rest of this section is to provide an efficient implementation of UPDATEPOSSIBLENEWEDGES(P, v, α, j') where the size of P is always at most $n + k$ and only $O(k)$ edges are considered in each invocation of the routine. Key to our analysis is the observation that tight edges cannot cross is the following sense: Given two ads $i < i'$ of the same type θ, and two slots $j < j'$, then we cannot have the edge from ad i to slot j' be tight, and simultaneously have the edge from ad i' to j be tight.

Lemma 1 (Non-crossing lemma). *Given two ads $i < i'$ of the same type θ, and two slots $j < j'$, if $v_i > v_{i'}$ and $\alpha_j^{(\theta)} > \alpha_{j'}^{(\theta)}$ then in any feasible dual solution we cannot have the edge from ad i to slot j' be tight, and simultaneously have the edge from ad i' to j be tight.*

Proof. We prove by contradiction. If the edges between i and j' and i' and j are both tight, then we must have dual variables $u_i, u_{i'}, p_j, p_{j'}$ such that

$$\alpha_{j'}^{(\theta)} \cdot v_i = u_i + p_{j'} \qquad \text{and} \qquad \alpha_{j}^{(\theta)} \cdot v_{i'} = u_{i'} + p_j.$$

At the same time, due to the slackness constraints, we must have that

$$\alpha_{j}^{(\theta)} \cdot v_i \leq u_i + p_j \qquad \text{and} \qquad \alpha_{j'}^{(\theta)} \cdot v_{i'} \leq u_{i'} + p_{j'}.$$

We can combine these and obtain

$$\alpha_{j'}^{(\theta)} \cdot v_i + \alpha_{j}^{(\theta)} \cdot v_{i'} = u_i + p_{j'} + u_{i'} + p_j \geq \alpha_{j}^{(\theta)} \cdot v_i + \alpha_{j'}^{(\theta)} \cdot v_{i'}.$$

Which is false due to the standard exchange argument. We give the argument for completeness: Rearranging we have $(\alpha_{j}^{(\theta)} - \alpha_{j'}^{(\theta)}) \cdot (v_i - v_{i'}) \leq 0$; however, due to strict monotonicity $\alpha_{j}^{(\theta)} > \alpha_{j'}^{(\theta)}$ and $v_i > v_{i'}$, so have reached a contradiction. □

UpdatePossibleNewEdges The goal of UPDATEPOSSIBLENEWEDGES(P, v, α, j') is to iterate over the edges incident on j' that are tight or that can potentially become tight later in the execution of the current phase. For each such edge (i', j') we compare its slack with the priority associated with i' and update the entry for i' in P accordingly if needed.

The exact definition of the edges inspected is given by Algorithm 2. Before we describe how this works, let us make some observations about the set of edges that can potentially become tight, and then we shall see that the Algorithm indeed considers all these edges.

For each ad type θ we first consider the edges of the form $(a_i^{(\theta)}, j')$ where $a_i^{(\theta)}$ is matched and $M(a_i^{(\theta)}) < j'$. We claim that we only need to consider the largest such i. Recall that all the edges in M are tight and remain tight throughout the execution of the phase; in particular, $(a_i^{(\theta)}, M(a_i^{(\theta)}))$ is tight and remains tight. Thus, any edge $(a_{i'}^{(\theta)}, j')$ with $i' < i$) is not tight and will never become tight due the Non-crossing Lemma 1 and the fact that $i' < i$ and $M(a_i^{(\theta)}) < j'$.

Now consider the edges of the form $(a_i^{(\theta)}, j')$ where $a_i^{(\theta)}$ is matched and $M(a_i^{(\theta)}) > j'$. We claim that we only need to consider the smallest such i. Recall that all the edges in M are tight and remain tight throughout the execution of the phase; in particular, $(a_i^{(\theta)}, M(a_i^{(\theta)}))$ is tight and remains tight. Thus, any edge $(a_{i'}^{(\theta)}, j')$ with $i' > i$ is not tight and will never become tight due the Non-crossing Lemma 1 and the fact that $i' > i$ and $M(a_i^{(\theta)}) > j'$.

Finally, we need to consider edges of the form $(a_i^{(\theta)}, j')$ where $a_i^{(\theta)}$ is unmatched. We claim that we only need to consider the smallest such i available[8].

[8] It is worth noting that even this case can be ignored if there exists a matched $a_i^{(\theta)}$ such that $M(a_i^{(\theta)}) > j'$; however, for ease of presentation we add the slot to X even if such $a_i^{(\theta)}$ exists.

Indeed, for any other $i' > i$ note that $v_i^{(\theta)} > v_{i'}^{(\theta)}$ and since the u variable of both ads is 0 (only slots that are part of an alternating tree get their dual variables increased and those are always matched) the slack of $(a_i^{(\theta)}, j')$ will always be smaller than the slack of $(a_{i'}^{(\theta)}, j')$ since $v_i^{(\theta)} \alpha_{j'}^{(\theta)} - p_{j'} < v_{i'}^{(\theta)} \alpha_{j'}^{(\theta)} - p_{j'}$. Furthermore, notice that if the edge $(a_i^{(\theta)}, j')$ become tight, then we immediately have an augmenting path in B, which concludes the phase.

These three cases are precisely those covered by Algorithm 2.

Algorithm 2 UPDATEPOSSIBLENEWEDGES

```
Input: $P, v, \alpha, M, j'$
 1: $X \leftarrow \emptyset$
 2: for ad type $\theta$ do
 3:     let $a_i^{(\theta)}$ be the unmatched ad of type $\theta$ with smallest $i$
 4:     add $a_i^{(\theta)}$ to $X$
 5:     if exists matched ad $a_i^{(\theta)}$ such that $M(a_i^{(\theta)}) < j'$ then
 6:         let $a_i^{(\theta)}$ be such an ad with largest $i$
 7:         add $a_i^{(\theta)}$ to $X$
 8:     end if
 9:     if exists matched ad $a_i^{(\theta)}$ such that $M(a_i^{(\theta)}) > j'$ then
10:         let $a_i^{(\theta)}$ be such an ad with smallest $i$
11:         add $a_i^{(\theta)}$ to $X$
12:     end if
13: end for
14: for $i' \in X$ do
15:     if $i' \notin B$ and either $i' \notin P$ or $i'$' current slack in $P$ is $> (v_{i',j'} - u_{i'} - p_{j'})$ then
16:         update the priority of $i'$ using $(i', j')$ or set if $i' \notin P$
17:     end if
18: end for
```

Lemma 2. *There can be at most $n + k$ ads in P at any given point in time.*

Proof. Notice that the the only edges (i', j') that we consider in Line 16 are either to a matched node in M or to the highest unmatched ad of each type. There are exactly $j < n$ matched ads in phase j and there are k ad types, so the lemma follows. □

Lemma 3. UPDATEPOSSIBLENEWEDGES *considers only $O(k)$ edges when updating P and these are the only edges we need to look at. Furthermore, these edges can be identified in $O(k)$ time provided we carry out $O(nk)$ pre-processing every time the matching M is augmented in Line 19 of Algorithm 1.*

Proof. For each ad type we need to consider at most three edges incident on j' (namely, those consider the for loop in Line 14) so the algorithm inspects at most $3k$ edges. The reason why we can focus just on these edges has already been explained in the description of the algorithm UpdatePossibleNewEdges.

In order to identify these edges efficiently, we maintain an array of length k where for each ad type θ we store the smallest index i such that $a_i^{(\theta)}$ is unmatched. In addition to this, we maintain a $k \times n$ array where for each ad type θ and position j' we store the largest index i such that $M(a_i^{(\theta)}) < j'$ and the smallest index i such that $M(a_i^{(\theta)}) > j'$.

It is easy to see that that constructing these data structures can be done in $O(kn)$ time given M and that given these data structures we can execute Algorithm 2 in $O(k)$ time. □

Notice that the pre-processing needed for executing Algorithm 2 efficiently, does not add to the time complexity of the algorithm since the matching is updated at most n times, so the overall time spent on the pre-processing step alluded in Lemma 3 is $O(kn^2)$.

Similar to the general bipartite matching case, the dual variables in our algorithm for the Ad Types setting can be used to recover VCG prices without increasing the asymptotic running time. This yields VCG prices in $O(n^2(k + \log n))$ time.

3.2 Large Number of Ad Types

When each ad has its own type (so $k = n$) the running time from Theorem 1 becomes $O(n^3)$, meaning that it is no faster than running the standard Hungarian algorithm. The following lemma shows that this is to be expected as any instance of the assignment problem can be reduced to an instance where all ads agree on the order of the slots.

Lemma 4. *With $k = n$ ad types, the Ad Types problem is no easier to solve than the assignment problem, even with monotone discount curves.*

Due to lack of space this proof is deferred to the full verison of the paper.

4 The Ad Types Problem with Gap Rules

In this section we switch our attention to the full version of the Ad Types problem where we do have gap rules. This problem is much harder if we do not place any restriction on the instances

Theorem 2. *The Ad Types problem with Gap Rules is hard to approximate better than $k^{1-\epsilon}$ for any $\epsilon > 0$, unless $P = NP$, even when the discount curves of all the ad types are identically equal to* 1.

Due to lack of space, the proofs in the section are given in the full version. On the positive side, we show that the problem is tractable if the number of ad types is very small. Note that this running time still represents a significant improvement over the brute-force approach yielding a $O(k^n)$ running time.

Theorem 3. *The Ad Types Problem with Gap Rules can solved optimally in $O(k \cdot n^{2k+1})$ time.*

References

1. Zoe Abrams, Arpita Ghosh, and Erik Vee. Cost of conciseness in sponsored search auctions. In *Proc. of 3rd WINE*, pages 326–334, 2007.
2. Gagan Aggarwal, Jon Feldman, S. Muthukrishnan, and Martin Pal. Sponsored search auctions with markovian users. In *Proc. of 4th WINE*, pages 621–628, 2008.
3. Susan Athey and Glenn Ellison. Position auctions with consumer search. *The Quarterly Journal of Economics*, 126(3):1213–1270, 2011.
4. Ruggiero Cavallo, Prabhakar Krishnamurthy, Maxim Sviridenko, and Christopher A Wilkens. Sponsored search auctions with rich ads. In *Proc. of 26th WWW*, pages 43–51, 2017.
5. Ruggiero Cavallo, Maxim Sviridenko, and Christopher A Wilkens. Matching auctions for search and native ads. In *Proc. of 19th EC*, pages 663–680, 2018.
6. Ruggiero Cavallo and Christopher A. Wilkens. Gsp with general independent click-through-rates. In Tie-Yan Liu, Qi Qi, and Yinyu Ye, editors, *Proc. of 10th WINE*, pages 400–416, 2014.
7. Riccardo Colini-Baldeschi, Julián Mestre, Okke Schrijvers, and Christopher A. Wilkens. The ad types problem. *CoRR*, abs/1907.04400, 2019.
8. Xiaotie Deng, Yang Sun, Ming Yin, and Yunhong Zhou. Mechanism design for multi-slot ads auction in sponsored search markets. In *Proc. of the 4th FAW*, pages 11–22, 08 2010.
9. Paul Dütting, Monika Henzinger, and Ingmar Weber. Sponsored search, market equilibria, and the hungarian method. *Information Processing Letters*, 113(3):67–73, 2013.
10. David Easley, Jon Kleinberg, et al. *Networks, crowds, and markets*, volume 8. Cambridge university press Cambridge, 2010.
11. Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American economic review*, 97(1):242–259, 2007.
12. Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. of ACM*, 19(2):248–264, 1972.
13. Gabriele Farina and Nicola Gatti. Ad auctions and cascade model: Gsp inefficiency and algorithms. In *Proc. OF 13th AAAI*, pages 489–495, 2016.
14. Dimitris Fotakis, Piotr Krysta, and Orestis Telelis. Externalities among advertisers in sponsored search. In *Proc. of the 4th SACT*, volume 6982, pages 105–116, 2011.
15. Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. of ACM*, 34(3):596–615, 1987.
16. Harold N Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. of computer and system sciences*, 30(2):209–221, 1985.
17. Nicola Gatti, Marco Rocco, Paolo Serafino, and Carmine Ventre. Towards better models of externalities in sponsored search auctions. *Theoretical Computer Science*, 745:150–162, 2018.
18. Arpita Ghosh and Mohammad Mahdian. Externalities in online advertising. In *Proc. of 17th WINE*, pages 161–168, 2008.
19. Arpita Ghosh and Amin Sayedi. Expressive auctions for externalities in online advertising. In *Proc. of 19th WINE*, pages 371–380, 2010.
20. Ioannis Giotis and Anna R. Karlin. On the equilibria and efficiency of the gsp mechanism in keyword auctions with externalities. In *Proc. of 4th WINE*, pages 629–638, 2008.

21. Fred Glover. Maximum matching in a convex bipartite graph. *Naval research logistics quarterly*, 14(3):313–316, 1967.
22. Renato Gomes, Nicole Immorlica, and Evangelos Markakis. Externalities in keyword auctions: An empirical and theoretical assessment. In *Proc. of 5th WINE*, pages 172–183, 2009.
23. Jason Hartline, Nicole Immorlica, Mohammad Reza Khani, Brendan Lucier, and Rad Niazadeh. Fast core pricing for rich advertising auctions. In *Proc of 19th EC*, pages 111–112, 2018.
24. Patrick Hummel. Position auctions with dynamic resizing. *International Journal of Industrial Organization*, 45:38–46, 2016.
25. Irit Katriel. Matchings in node-weighted convex bipartite graphs. *INFORMS Journal on Computing*, 20(2):205–211, 2008.
26. David Kempe and Mohammad Mahdian. A cascade model for externalities in sponsored search. In *Proc. of 4th WINE*, pages 585–596, 2008.
27. Walter Kern, Bodo Manthey, and Marc Uetz. Note on VCG vs. price raising for matching markets. *CoRR*, abs/1604.04157, 2016.
28. Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
29. Witold Lipski and Franco P Preparata. Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. *Acta Informatica*, 15(4):329–346, 1981.
30. Mohammad Mahdian, Okke Schrijvers, and Sergei Vassilvitskii. Algorithmic cartography: Placing points of interest and ads on maps. In *Proc. of 21st KDD*, pages 755–764, 2015.
31. James Munkres. Algorithms for the assignment and transportation problems. *J. of SIAM*, 5(1):32–38, 1957.
32. S. Muthukrishnan. Bidding on configurations in internet ad auctions. In *Proc. of 15th COCOON*, pages 1–6, 2009.
33. Roger B Myerson. Optimal auction design. *Mathematics of operations research*, 6(1):58–73, 1981.
34. C Greg Plaxton. Fast scheduling of weighted unit jobs with release times and deadlines. In *Proc. of 35th ICALP*, pages 222–233, 2008.
35. C Gregory Plaxton. Vertex-weighted matching in two-directional orthogonal ray graphs. In *Proc. of 24th ISAAC*, pages 524–534, 2013.
36. Lyle Ramshaw and Robert E Tarjan. On minimum-cost assignments in unbalanced bipartite graphs. *HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2012-40R1*, 2012.
37. Tim Roughgarden. Revenue-maximizing auctions. *Twenty lectures on algorithmic game theory*, 2016.
38. R Sharathkumar and Pankaj K Agarwal. Algorithms for the transportation problem in geometric settings. In *Proc. of 23rd SODA*, pages 306–317, 2012.
39. Hal R Varian. Position auctions. *International Journal of Industrial Organization*, 25(6):1163–1178, 2007.