Output URL Bidding*

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

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

Ali Dasdan
Ebay Inc
San Jose, CA, USA
adasdan@ebay.com

Santanu Kolay
Ebay Inc
San Jose, CA, USA
skolay@ebay.com

ABSTRACT

Output URL bidding is a new bidding mechanism for sponsored search, where advertisers bid on search result URLs, as opposed to keywords in the input query. For example, an advertiser may want his ad to appear whenever the search result includes the sites `www.imdb.com` *and* `en.wikipedia.org`, instead of bidding on keywords that lead to these sites, e.g., movie titles or actor names.

In this paper we study the tradeoff between the simplicity and the specification power of output bids and we explore their utility for advertisers. We first present a model to derive output bids from existing keyword bids. Then, we use the derived bids to experimentally study output bids and contrast them to input query bids. Our main results are the following: (1) Compact output bids that mix both URLs and hosts have the same specification power as more lengthy input bids; (2) Output bidding can increase the recall of relevant queries; and (3) Output and input biding can be combined into a hybrid mechanism that combines the benefits of both.

1. INTRODUCTION

The dominant form of search engine advertising is keyword advertising, in which search engines run a continuous auction to sell advertisement (ad) space on the search engine results page (SERP). In this form of auction, advertisers bid on keywords in user queries and search engines determine the winning bids using a combination of the bid amounts and the clickability of the associated ads. In the end, search engines earn money when users click on the advertisements (ads) and advertisers benefit when these clicks result in a purchase.

A new alternative to such keyword bidding is output bidding [8, 10]. With output bidding, advertisers bid on terms appearing in the output SERP, as opposed to bidding on terms in the input query. Output URL bidding is a variation of output bidding where the terms to bid on are the URLs or host names of search results. For example, Netflix, a popular DVD rental web site, may want its ad to appear whenever the search results include the URL `http://www.imdb.com`, the Internet movie database web site. Of course, Netflix could bid on the keywords that lead to results containing this site such as thousands of movie titles. However, bidding on one URL is a more compact or intuitive way of referring to all those keywords. Another advantage of using the URL `http://www.imdb.com` over the query keywords that lead to this site is that the advertiser may not be interested in some of those keywords. For instance, Netflix may not want to bid on actor names using keyword bidding, but yet, users who land at `imdb.com` because they were searching for actors, may end up looking at the actors' movies (which Netflix does rent) once they arrive at the site. For simplicity, in this paper we use "output bidding" to refer to "output URL bidding".

Incidentally, note that the keyword generation tools offered by popular ad networks [1] accept not only "related queries" but also "related URLs" as input and generate keywords that lead to those URLs. This existing indirect use of URLs indicates that URLs are a natural way to describe advertiser's interests and that output bidding can be used in conjunction with keyword bidding, as the techniques are complementary.

Even though output bidding intuitively seems to have huge potential, there are still many unresolved issues regarding its use and "effectiveness" in describing the interests of actual advertisers. In particular, these are some of the questions we address in this paper:

What are the options for output bidding? There are many ways in which an advertiser can describe result pages of interest. In this paper we focus on the use of result URLs. For instance, one simple way is to provide a list of URLs, and to check if any of these URLs appear in the output. We can extend this simple model by allowing the advertiser to specify logical expressions of some type, e.g., the output matches if 2 of these URLs are found, or if this URL is not found. We can also extend to matching of URL substrings, e.g., the host part of the result URLs. Each option represents a tradeoff between specification power, compactness and simplicity. In this paper we partially explore this space of options.

How "compactly" can interests be specified with output bidding, relative to keyword bidding? For example, say an advertiser has a list of 100 keywords related to ad a. This list leads to a set of queries R where the ad a would be displayed. If there are say 10 URLs such that at least one of these 10 URLs appears in the results of all R queries, then we could compactly *cover* the

*This work was supported by a grant from Yahoo! Inc and an Onassis Foundation Scholarship.

Proceedings of the VLDB Endowment, Vol. 4, No. 3

[1]For example, `http://adwords.google.com` or `http://adcenter.microsoft.com`.

set R with those 10 URLs with output bidding. We validate the compactness advantage of output over keyword bidding using real keyword ads.

Can output bidding help advertisers reach more relevant users? In the previous example, note also that these 10 URLs may also appear in results of queries that are not in R and these *spill* queries may or may not be related to a. To evaluate the utility of output bidding we study the relevance of spill queries to a given ad.

How can output bidding be combined with keyword bidding? As mentioned above output bidding can be used as a complementary mechanism to keyword bidding. In that case output bidding may be more suitable for reaching part of the target R queries, while keyword bidding is more suitable for the rest. We study the relation of the two mechanisms in this context to evaluate the utility of output bidding.

The answers to these questions provide a thorough picture about the strengths of different output bidding variations and their added value with respect to keyword bidding. This kind of analysis is an essential first step for the introduction of a new bidding mechanism that affects almost every aspect of existing sponsored systems.

Other important sets of questions revolve around the run-time performance and the auction design of output bidding. Regarding performance, note that in principle, to find ads that match a given results page, we first need to compute the results page. In a naive implementation, this matching would follow the construction of the results page, adding to the time end users need to wait. With keyword matching, on the other hand, matches can be computed in parallel to the preparation of the results page. Although we do not deal with these questions in this paper, we present some preliminary ideas in the Appendix (Section A) that suggest that there are ways around this delay problem, e.g., relying on caching or adding an index to the output bid matching process. Regarding auction design, we plan to investigate relevant questions in our future work.

The rest of the paper is organized as follows: In Section 2 we introduce a model for output URL bidding that allows bid expressions with varying complexity and specification power. In Section 3 we introduce the problem of "covering" a query set with an output bidding expression and we provide an algorithm to solve it efficiently. We use this algorithm as part of a model for deriving realistic output bidding expressions that we use to study output bidding. In Section 4 we present a model to evaluate whether advertisers can reach more users of interest by using output URL bidding. In Section 6 we experimentally compare the different output bidding models and we study a hybrid mechanism that combines keyword and output bidding in Section 6. We present related work in Section 7 and we conclude in Section 8.

2. MODEL AND NOTATION

A search engine takes each query q and computes $U(q)$, a ranked list of the top n URLs (organic search results). In this paper we treat $U(q)$ as a set, but a generalization to ranked lists is straightforward. The value $n = |U(q)|$ varies among search engines, but 10 is a typical value.

In output bidding, advertisers submit ads that specify URLs of interest. For each ad a, the advertiser provides an *output expression*: a logical expression that states when a should be a candidate for display. We overload a to denote the ad as well as its output expression. To illustrate, consider the output expression $a = (u_1 \in U(q)) \vee (u_2 \in U(q))$, stating that a is a display candidate if either URL u_1 or u_2 appear in the organic results. Since URLs are always matched against $U(q)$, we will write the above expression as $a = u_1 \vee u_2$. In this paper we work with all output expressions in Disjunctive Normal Form (DNF) because we believe this form is more natural for our task. With a DNF expression, each disjunct describes one condition that can trigger the placement of the ad, independent of the other disjuncts.

Note that one could also define output expressions based on hosts, as opposed to full URLs. For instance, we discussed in the introduction that an advertiser may want an ad to appear when the host `www.imdb.com` appears in the results page. We can analogously define the set of hosts in the results page, $H(q)$ and logical expression based on hosts, e.g., $a = (h_1 \in H(q)) \vee (h_2 \in H(q))$. In the first sections of this paper we focus on URL based expressions. However in our experimental results we also study host expressions, and even expressions that mix URLs and hosts.

Given a query q, the sponsored search system finds matching ads from its database. We refer to the set of available ads as A, and the match function that evaluates output expression a against q as $m(a, q)$. (That is, $m(a, q)$ checks if the URLs in $U(q)$ satisfy the logical expression a.)

3. QUERY SET OUTPUT COVER

As discussed in the introduction, one way to evaluate output bidding is to see if output bids can represent existing input (query) bids more compactly. That is, say we are given an input bid R containing queries an advertiser is interested in. Can we represent the advertisers intent with an output expression a that, say, contains many fewer URLs than queries in R?

First, if expression a is to be a replacement for input bid R, then the ad in question must be displayed every time that ad would be displayed with R. To capture this notion, we say that output expression a *covers* query set R if

$$\forall q \in R, m(a, q) \text{ is true.} \tag{1}$$

In addition to wanting an expression a that covers R:

- we want a to be *compact*, i.e., able to describe the pages of interest with few URLs or URL combinations; and
- we want a to match few queries other than the queries in R.

In the following two subsections we define these two properties. Then we present an algorithm that constructs a "desirable" output expression given an input bid R.

3.1 Compactness

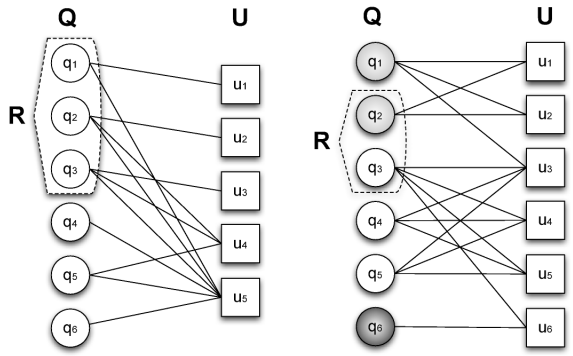
Intuitively, a compact output expression is more practical than a lengthy expression. For example, it seems better for an advertiser that rents DVDs to bid only on `www.imdb.com` rather than on 100 smaller web sites that appear in the same results pages as `www.imdb.com`. Even if it was intuitive to name these 100 web sites, it would require more resources to monitor that all of them appear consistently in the results pages of interest.

There are several ways to define compactness for output expressions. We chose one based on the number of disjuncts. In particular, we define the "cost" of an output expression a to be equal to its disjuncts count and we call this count the *size* $|a|$ of the expression:

$$|a| = \#\text{ disjuncts in } a. \tag{2}$$

For example, the size of the expression $u_1 \vee u_2$ is 2 and the size of the expression $(u_1 \wedge u_2) \vee u_3 \vee (u_4 \wedge u_5)$ is 3.

We believe that the number of disjuncts is an adequate first order approximation to the human cost of understanding and creating an output expression. Furthermore, this simple cost metric simplifies the synthesis of output expressions (Section 3.4). Nevertheless, in the experimental section we study how output expressions are also affected by the number of conjuncts within disjuncts.



(a) Query Set Output Cover (b) Spill Evaluation

Figure 1: Queries - Result URLs Bipartite Graphs

3.2 Spill

An output expression that covers a set R may also match queries that do not belong to R. We refer to the set of such queries as the *spill* of the expression a with respect to R:

$$spill(a, R) = \{q : q \in (Q - R) \wedge m(a, q)\} \tag{3}$$

The query set Q is a reference set that contains all of the queries that we consider for the spill calculation. Such a set should contain a representative sample of real queries submitted to a search engine. We can obtain this set from the search engine logs.

To illustrate how to calculate the spill of an output expression, it is convenient to represent queries Q and their search results with a bipartite graph as in Figure 1(a). The U nodes in the graph represent URLs. An edge such as (q_1, u_1) shows that $u_1 \in U(q_1)$, i.e., URL u_1 appears in the top search results of query q_1. If $R = \{q_1, q_2, q_3\}$, then some of the expressions that cover R and their corresponding spill sets are the following:

$$\begin{aligned} a_1 &:= u_1 \vee u_2 \vee u_3 & spill(a_1, R) &= \{\} \\ a_2 &:= u_1 \vee u_4 & spill(a_2, R) &= \{q_5\} \\ a_3 &:= u_5 & spill(a_3, R) &= \{q_4, q_5, q_6\} \end{aligned} \tag{4}$$

For instance, the spill for expression a_2 is $\{q_5\}$ because a_2 matches queries q_1, q_2, q_3, q_5, and the first three queries are in R.

Although spill queries are not in R, it is not necessarily bad to display our ad when the user types in a spill query. For instance, say the R queries are "CNN," "BBC" and "ABC News," while q_5 is "New York Times." If the advertiser wanted his ad shown to users interested in "news," then q_5 is not a bad choice, as the New York Times is a major news site. Perhaps the advertiser did not include a complete list of news organizations in his R set. On the other hand, if the advertiser only wanted users interested in "television news," then q_5 could be considered undesirable.

In the rest of this section we will consider all spill as undesirable. Then in Section 4 we divide spill into *positive* (acceptable) and *negative* based on (a rough approximation) to) advertiser intent.

3.3 Optimization Problem

To summarize the previous discussion, creating an output expression from an input query set comes down to solving the following problem denoted as the *Query Set Output Cover*: Given a set of queries R find a compact output expression a that matches all queries in R, while it matches as few as possible queries that are not in R.

By using Equations 1, 2 and 3 we can express the Query Set Output Cover problem as an optimization problem:

$$\begin{aligned} \underset{\text{(over } a\text{)}}{\text{minimize}} \quad & \gamma|a| + (1 - \gamma)|spill(a, R)| \\ \text{subject to} \quad & m(a, q), \quad \forall q \in R \end{aligned} \tag{5}$$

where $\gamma \in [0, 1]$ is a regularization parameter between the compactness and the spill minimization objective. These two objectives are inherently conflicting. An expression such as a_2 of Equation 4 that covers a set R with very few disjuncts usually contains few very common URLs that yield many spill queries. On the contrary, an expression such as a_1 that covers R with many disjuncts can be very specific and yield no spill.

The value of parameter γ determines the desired tradeoff between the two objectives. If $\gamma = 1$, then the optimal expression is the most compact one, independent of the spill that it yields; if $\gamma = 0$, then the optimal expression is any expression that yields no spill independent of its size. For example, if we had to choose among the expressions in Equation 4 to cover set R, then expression a_3 would be optimal if $\gamma = 1$, the expression a_1 would be optimal if $\gamma = 0$ and the expression a_2 would be optimal for $\gamma = 0.5$.

To determine the value of parameter γ, first we need to solve the optimization for different values in the range $[0, 1]$. Then, we examine all the solutions and we keep the value of γ that corresponds to the most practical one.

The Query Set Output Cover problem is very hard to solve. Even if we ignore either of the objectives the problem is still NP-hard:

THEOREM 1. *It is NP-hard to find the most compact output expression expression that covers all queries in a set R.*

THEOREM 2. *It is NP-hard to find an output expression that covers all queries in a set R, but yields the minimum possible spill.*

In the extended version of the paper [23], we prove Theorem 1 with a reduction to the Set Cover problem and Theorem 2 with a reduction to the Red-Blue Set Cover problem[5]. Although the greedy algorithm for the Set Cover problem is efficient and achieves an approximation ratio of $\log(n)/2$, there is no good approximation algorithm for the Red-Blue Set Cover problem ([5] has an LP-based $2\sqrt{n}$ approximation algorithm for a very restricted case of the problem).

3.4 Algorithm

In this section we present a heuristic solution to the Query Set Output Cover problem. Our algorithm uses an efficient greedy approach inspired by Set Cover greedy algorithm. Since output expression a should be in DNF, we select the disjuncts it should contain in an incremental fashion. We start with an "empty" disjunction and at each step we add the single URL that yields the minimum increase to the problem objective as defined in Equation 5.

Algorithm 1 describes the greedy approach to find an expression a that is a disjunction of URLs. At the end of this section we also discuss the appropriate algorithm modifications so that the algorithm returns a disjunction of URL conjunctions that contain no more than k URLs each.

The algorithm input consists of the query set R, the value of γ and the reference set Q. The algorithm returns the output expression a and the corresponding *spill* with respect to R.

In lines 1-7 the algorithm calculates for each URL u that appears in the results of some query $q \in R$ the following sets:

$$\begin{aligned} C[u] &= \{q | q \in R \wedge u \in U(q)\} && (R \text{ queries covered by } u) \\ S[u] &= \{q | q \in (Q - R) \wedge u \in U(q)\} && (\text{spill of } u \text{ w.r.t. } R) \end{aligned}$$

Algorithm 1 Greedy Query Set Output Cover

Input: R, γ, Q
Output: a, $spill$

```
1: Allocate empty hash tables C and S. C[u] (or S[u]) returns ∅
   if key u is not in C (or S).
2: for all q ∈ Q do
3:     for all u ∈ U(q) do
4:         if q ∈ R then
5:             C[u] ← C[u] ∪ {q}
6:         else
7:             S[u] ← S[u] ∪ {q}
8: R^rem ← R                          ▷ R^rem: Remaining queries to be covered
9: U_a ← ∅                            ▷ U_a: URLs in logical expression a
10: spill ← ∅
11: while R^rem ≠ ∅ do
12:     u ← arg max_{u'∉U_a} |C[u'] ∩ R^rem| − w(γ)|S[u'] − spill|
13:     U_a ← U_a ∪ {u}
14:     R^rem ← R^rem − C[u]
15:     spill ← spill ∪ S[u]
16: a ← DISJUNCTION(U_a)
17: return a, spill
```

In the main loop, in lines 11-15, the algorithm calculates incrementally the disjuncts set U_a of the output expression a and the $spill$. In each iteration the URL u to be added to U_a is selected in a greedy manner taking into account:

- $|C[u] \cap R^{rem}|$: the number of "uncovered" queries where u occurs; and
- $|S[u] - spill|$: the number of additional spill queries that u yields.

URL u should maximize the former number and minimize the latter. Thus, the algorithm selects u to maximize the difference between the two numbers accounting the additional spill with weight $w(\gamma)$. The value of the weight $w(\gamma)$ should reflect in this local decision the desired tradeoff between compactness and spill as defined by the value of γ. A valid weight function must have the following three properties: (i) $w(\gamma)$ is a decreasing function of γ, (ii) $w(0) = \infty$ and (iii) $w(1) = 0$. In this paper we use:

$$w(\gamma) = (1 - \gamma)/\gamma \tag{6}$$

that satisfies all three properties and in addition it yields $w(0.5) = 1$, i.e., it considers both objectives as equally important if $\gamma = 0.5$. At the end of each iteration the algorithm updates the $spill$ and the queries R^{rem}, i.e., the remaining queries to be covered in the following iterations.

Finally, in lines 16-17, the algorithm calculates a as the disjunction of all URLs in U_a and it returns a and $spill$.

As an implementation hint, note that Algorithm 1 uses only set counts $|C[u'] \cap R^{rem}|$ and $|S[u'] - spill|$ to select a URL to add to U_a. Although the use of sets makes the algorithm presentation more clear, the actual implementation can use just the counts, which can be efficiently updated in every iteration. By storing the counts in a heap, the selection step (line 12) will take $\log |U|$ time in the worst case. If the degree of the selected URL u is $d(u)$, i.e., u appears in the top-n results of $d(u)$ queries, then the algorithm must update the counts of all the URLs in the results of these $d(u)$ queries. Updating the counts and the heap will take $nd(u) \log |U|$ in the worst case. Note that with such an implementation, the algorithm needs to fetch the queries that are connected to a popular URL such as `www.imdb.com` only if the URL is selected in line 12. This cannot be avoided, because the algorithm must output the spill of the returned output expression a.

We can modify the first steps of the algorithm to consider also conjunctions with up to k URLs. To do this we should calculate sets $C[u]$ and $S[u]$ not only for singleton URLs u, but also for conjunctions of up to k URLs that appear in the same results list $U(q)$, e.g., $C[u \wedge u']$ for $k = 2$. Although this approach is not scalable in general, it is adequate for values of k up to 2 or 3.

We can also use Algorithm 1 with hosts in place of URLs to obtain output expressions based on hosts. Similarly, we obtain output expressions than contain both URL and host conjunctions by calculating the sets $C[u]$ and $S[u]$ for the union of URLs and hosts.

4. SPILL EVALUATION

As discussed in Section 3.2 one needs to know the "intention" of the advertiser in evaluating the spill of output expression a with respect to input bid R. Since it is very hard to get access to advertisers, we infer their intentions instead by analyzing a set of queries, and clustering them into groups that reflect a similar intent.

To illustrate, in Figure 1(b) we present a bipartite graph that is similar to the graph in Figure 1(a). Suppose that we have been able to cluster or group queries by their intent, as follows:

$$G_1 = \{q_1, q_2\} \quad \text{(light gray queries)}$$
$$G_2 = \{q_3, q_4, q_5\} \quad \text{(white queries)}$$
$$G_3 = \{q_6\} \quad \text{(dark gray query)}$$

That is, queries q_1 and q_2 are similar, and so on. There are many ways to form these clusters and we discuss the method we use in our experiments in Section B of the Appendix.

To illustrate how we use these clusters to determine if spill is positive or negative, consider an input bid $R = \{q_3, q_4\}$. An output expression that covers R is $a = u_2 \vee u_6$. Note that R contains queries from 2 clusters, thus we say it has two possible intents, G_1 and G_2. We say that q_1 is positive spill, because it is in one of the possible intents. We say that q_6 is negative, because it is not in one of the R intents.

Since our clustering only yields an approximation to the advertiser's intent, our positive and negative spill evaluation should be taken with caution. If negative spill turns out to be very low, it *suggests* that an expression matches the intent of the advertiser pretty well. If it does not, it suggests that the ad will be displayed on pages that are not of interest to the advertiser.

In Section C of the Appendix we discuss the integration of our spill evaluation approach in the Query Set Output Cover problem and the required changes to the greedy algorithm.

5. COMBINED EXPRESSIONS

In this section we explore the properties of expressions that cover an input query set partially by using an output expression and partially by a keyword expression, i.e., by explicitly listing the queries to be matched. We call such expression *combined expressions*. In the following two paragraphs we discuss the motivation for studying such expressions. We evaluate combined expressions experimentally in Section 6.4.

The output expressions we have studied so far were generated to cover all the queries in their input query sets. Such output expressions often include disjuncts that are used to cover only a certain query of the input query set. For example, say that input query set R has 10 queries and we used Algorithm 1 to generate the 1-URL output expression $a = u_1 \vee u_2 \vee u_3 \vee u_4$. In this expression every disjunct is a single URL and the index in every URL shows the order of selection by Algorithm 1. Since the URLs are selected in a greedy manner URL u_1 may cover 5 input queries, the disjunction $u_1 \vee u_2$ may cover 8 input queries, the disjunction $u_1 \vee u_2 \vee u_3$ may

cover 9 queries and the disjunction $u_1 \vee u_2 \vee u_3 \vee u_4$ covers all 10 queries. Note that URLs u_3 and u_4 were added to the disjunction only to cover one query each.

In such cases it is hard to argue that output bidding is more intuitive than keyword bidding. So instead of an output expression that covers all the queries in R, we can only generate an output expression with disjuncts that cover more than one query each. We can obtain such an expression from Algorithm 1 by terminating the main algorithm loop when the new disjunct to be added to the output expression covers only one additional query. In our example the output expression would be $u_1 \vee u_2$. To cover the set of the remaining queries R^{rem} we can use keyword bidding and treat each query as a keyword.

6. EXPERIMENTS

We present an experimental evaluation of output URL bidding based on output expressions that we generate from real keyword bidding ads using Algorithm 1. The goals of our experiments are the following:

- to study the tradeoff between the compactness of output expressions and the spill that they yield (Section 6.2);
- to evaluate the relevance of spill queries to the queries of the base input query set (Section 6.3); and
- to explore the use of output URL bidding as a complementary mechanism to keyword bidding (Section 6.4).

Prior to presenting the main experimental results, we describe the experiments setup in Section 6.1. For the dataset that we use and implementation details see Sections E.1 and E.2 of the Appendix. For additional results see the extended version of this paper [23].

6.1 Experimental Setup and Variations

We generate output expressions that are equivalent to the dataset keyword bidding ads and we use these expressions to evaluate output bidding. The generation procedure of an output expression has the following two steps: (1) we pick a dataset ad and we let R be the set of log queries where the ad was shown; then (2) we provide the set R as input to Algorithm 1 to obtain an output expression that covers all R queries. In our experiments we used 2,251 ads that we picked at random from our dataset. The only restrictions in our selection were that an ad should appear in more than 10 different queries, i.e., $|R| \geq 10$, so that set R is representative of the advertiser intent; and $|R| < 10,000$, so that we can afford to test many different variations of output bidding (see Section E.2 for running times). The average $|R|$ in the 2,251 sets is 128.

We explore different output bidding variations that differ in (a) the type of literals that they use, e.g., URLs, hosts or both; and (b) the maximum number k of conjuncts in a disjunct. We name every variation using the number k and the type of literal it uses. For example, a 2-URL expression is based on URLs and may use up to 2 conjuncts per disjunct. We studied the following 8 variations: 1-URL, 2-URL, 3-URL; 1-host, 2-host, 3-host; and 1-mixed and 2-mixed. A k-mixed expression mixes URL with host conjunctions and each URL or host conjunction may have up to k conjuncts.

6.2 Compactness and Spill Tradeoff

In this set of experiments we study the tradeoff between the compactness of output expressions and the spill that they yield for all 8 variations of output URL bidding. (We do not classify spill into positive or negative until Section 6.3.) Recall that an output expression that covers a certain set R can either have few disjuncts, i.e., be compact, and yield much spill or it can have many disjuncts and yield less spill depending on the value of parameter γ that was used for its generation. To study the tradeoff for each variation we generated output expressions using the following values[2] of γ: $0.1, 0.5, 0.9, 0.99, 0.999, 0.9999$. For each γ we generated output expressions for all 2,251 query sets of our dataset. In the following paragraphs we provide a brief statement of the experiments main results and then we proceed with analysis in finer detail.

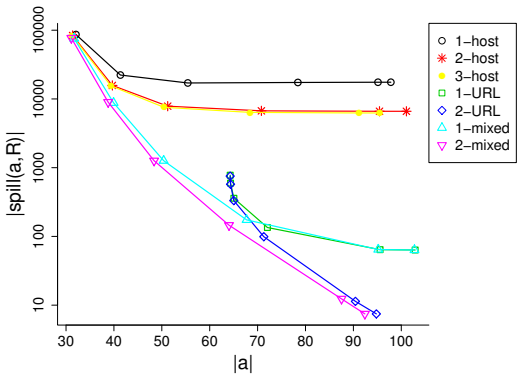


Figure 2: Compactness and Spill Tradeoff

Result 1: Output expressions based both on URLs and hosts are more compact and yield smaller spill compared to expressions based only on URLs or only on hosts.

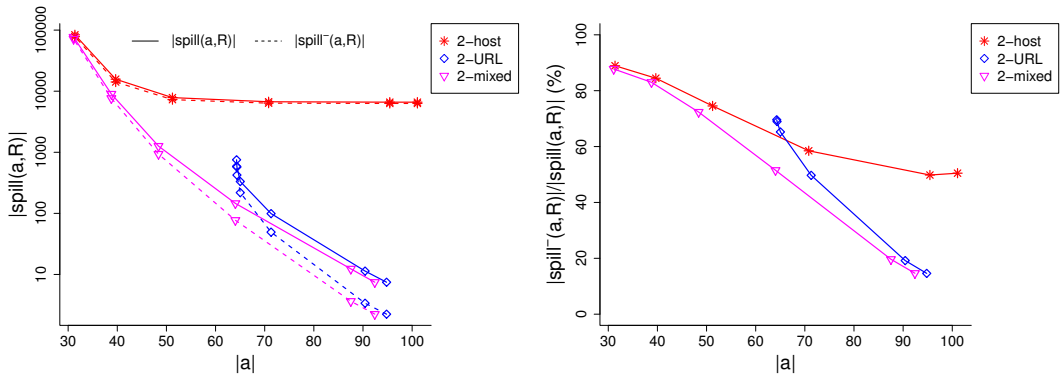
In Figure 2 we plot the average spill size versus the average expression size in double logarithmic scale. Each line in the plot looks at a different variation and each point of a line corresponds to a different γ value. The value of γ decreases toward the right. For example, the right-most triangle in the 2-mixed line shows that for $\gamma = 0.1$ a 2-mixed output expression has on average 92 disjuncts and the expression matches about 9 spill queries. The averages are computed over all 2,251 input query sets. For presentation's sake we omit the line that corresponds to 3-URL expressions, since it overlaps almost exactly with the 2-URL line.

Note that a point in Figure 2 is strictly better than all other points that are both above it and to the right, i.e, to put it simply, bottom and left is better. We will first discuss the impact of the literals used, i.e., URLs, hosts or both, and then we will discuss the role of k as part of the next result discussion.

Host-based output expressions can be more compact than URL-based expressions, because a single host matches all of the result pages with URLs from this host. However, matching many queries has a negative impact on spill. For example, the left-most points in the 1-host, 2-host and 3-host lines show that for $\gamma = 0.9999$ the host-based output expressions have approximately 35 disjuncts. However, these expressions include popular hosts such `en.wikipedia.org` that match millions of queries and as a result the average spill size is more than 1M. For lower values of γ the spill size is still high, because many queries contain only popular hosts in their results and the spill penalization cannot prevent their inclusion in the logical expression.

URL-based expressions contain on average more than 60 disjuncts. Many URLs appear only once in all of the result pages of set R queries and up to 50% of the R queries contain only such URLs. So the URL expressions must contain disjuncts that cover only a single query and this constrains their compactness. The URL sparsity is also responsible for the small spill size the expressions.

[2]We selected such values since the maximum size of an output expression is $|R|$ and $|R| < 10K$, while the spill in the worst case has 12M queries. The worst case occurs when the spill contains all of the dataset queries that do not belong to R.



(a) Spill vs Output Expression Size (b) Negative Spill Fraction

Figure 3: Spill Evaluation.

Mixed output expressions combine the advantages of both host- and URL-based expressions. They can have small size by using popular hosts, but they can also minimize the spill by using rare URLs. So, the tradeoff curves of k-mixed expressions are strictly better than the ones of k-URL and k-host expressions.

Result 2: Output expressions that use more than two conjuncts per conjunction yield negligible gains in compactness and spill decrease.

Output expressions that use up to two conjuncts per conjunction, i.e., $k = 2$, yield fewer spill queries than expressions that have no conjunctions, i.e., $k = 1$. The reason for the spill decrease is the increased sparsity of URL or host pairs compared to single URLs or hosts. However, increasing k to 3 results in a negligible spill decrease for both URL-based and host-based queries compared to $k = 2$. In case of URL-based expressions, URL pairs are already very sparse and the consideration of URL triples does not add any value. In case of host-based expressions, the spill size does not decrease because search results with popular hosts are often repeated. For example, the top hosts in the results of queries about song lyrics are almost the same, e.g. `www.metrolyrics.com`, `www.last.fm`, etc.. Consequently, no matter how many and which conjuncts we select to cover one lyrics query the conjunction that they will form will also match any other lyrics query. Hence, the number of lyrics spill queries cannot be decreased by using host-based expressions.

Result 3: Output expressions that cover big query sets are relatively more compact than expressions that cover small query sets.

Due to the lack of space, we present the detailed discussion of this result in the extended version of this paper [23]. The result comes from the analysis of the variation of output expression sizes over different sizes of the input query sets.

6.3 Spill Evaluation

We clustered the queries of our dataset using the algorithm that we present in Section B of the Appendix. The algorithm basically clusters together queries with more than $j = 4$ overlapping results. As discussed in Section B.2 the clustering results show that the choice of $j = 4$ is rather strict. So the positive spill sizes that we present in this section are only conservative estimates of the values we would obtain with a looser clustering, e.g., $j = 3$.

We use the obtained query clusters to evaluate the spill of all output expressions that we generated in in Section 6.2 using the spill evaluation approach that we presented in Section 4. In the following paragraphs we state and discuss our main results.

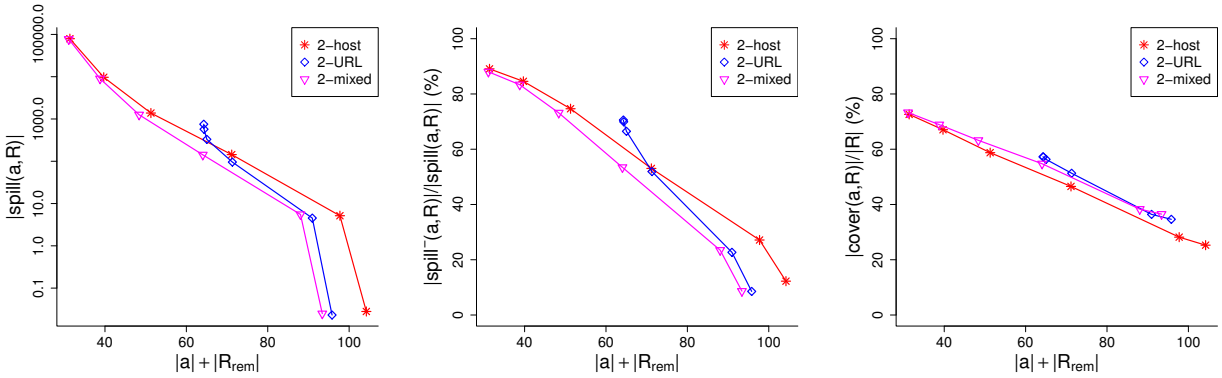
Result 4: The use of URL conjuncts results in spill queries that are mostly relevant to input queries, while the use of host conjuncts results in mostly irrelevant spill queries.

In Figure 3(a) we plot the total spill and negative spill for the 2-host, 2-URL and 2-mixed output expressions. We did not plot the rest of the variations to simplify the figure. The solid lines in Figure 3(a) show the total spill for each variation and they are the same as the corresponding lines in Figure 2. The dashed lines show the average negative spill. For every variation, the points at the same $|a|$ value correspond to the same value of γ. For example, the fourth triangles from the left in the 2-mixed solid and dashed line show the average total and the average negative spill for $\gamma = 0.9$. These two triangles show that on average a 2-URL expression of size 65 yields 135 spill queries but only 73 of them are negative spill (recall that the y-axis is in logarithmic scale).

Note that in case of 2-host expressions the negative spill line is very close to the total spill line. As we discussed before, host-based expressions usually contain some very popular hosts that appear in many queries and most of them are not relevant to the input query sets. On the contrary, the negative spill line of 2-URL expressions is far below the total spill line. The content of a single web page is more specific than all the contents of one host. So queries that have the one or two URLs in common in their results are more likely to be relevant than queries that share one or two hosts. Mixed output expression yield more negative spill if they contain mostly host conjuncts, while they yield less negative spill if they mostly contain URL conjuncts. Hence, the distance between the 2-mixed solid and the 2-mixed dashed line increases as $|a|$ increases, because less compact 2-mixed ads contain mostly URL conjuncts.

Result 5: The positive spill of output expressions can outweigh the negative spill.

In Figure 3(b) we plot the average percentage of negative spill. Figure 3(b) presents the results for the same variations as Figure 3(a) and both figures have the same x-axis. The y-coordinate of a point in Figure 3(b) is calculated in two steps: (1) we calculate percentage of negative spill to the total spill for all 2,251 output expressions that we obtained for a certain variation and a value of γ; then, (2) the y-coordinate is the average of all 2,251 percentage values. For example, the fourth triangle from the left shows that for $\gamma = 0.9$ 2-mixed expressions have on average size 65 and 50% of their spill is negative. Observe that that the percentage of negative spill is 50% or higher for 2-host expressions, while it can be as low as 15% for mixed and URL-based expressions. Such an observation reinforces Result 4.



(a) Spill (b) Negative Spill Fraction (c) Covered Queries Fraction

Figure 4: Combining Keyword and Output Bidding.

Figure 3(b) also shows implicitly the percentage of positive spill, since both positive and negative spill percentages add up to 100%. Note that for mixed and URL-based expressions the positive spill is higher than 50% of the total spill for expression sizes higher than 65 and 70 respectively. Such percentage translates into approximately 50 relevant queries on average that are not included in the input query set. Taking into account that input query sets contain on average 128 queries, we see that output expressions can significantly increase the coverage of relevant queries.

In general, cases where positive spill is higher than 50% are really good. The advertiser gets his ad displayed on more relevant pages (positive spill) than what set R achieved and the number of ad placements on non-relevant pages is outweighed by the gains (positive spill). Recall also that the calculation of the positive spill is based on query clustering with the fairly conservative $j = 4$. So, in reality the positive spill is expected to be even larger.

6.4 Combined Expressions

We repeated the experiments of Sections 6.2 and 6.3 using the modified version of Algorithm 1 that we presented in Section 5. In the following paragraphs we present our main results.

Result 6: Combined expressions are as compact as pure output expressions, but the former yield significantly fewer spill queries than the latter.

In Figure 4(a) we plot the average spill size versus the average combined expression size in double logarithmic scale. This figure is similar to Figure 2, but it only includes three variations for presentation purposes. The x-axis shows the size of the combined expression that is equal to the sum of the output expression size $|a|$ and the keyword expression size $|R^{rem}|$. All three tradeoff curves in Figure 4(a) are lower than the corresponding curves in Figure 2. The difference is more notable for 2-host combined expressions that yield orders of magnitude less spill than the corresponding 2-host output expressions. The reason for the spill decrease is that a combined expression does not yield any spill for covering the queries in the set R^{rem}, since these queries are covered explicitly by the keyword part of the expression. The decrease is dramatic in case of host-based expressions, since the queries of R^{rem} often include in their results only hosts that yield high spill, and these hosts had to be added to the output expression.

Result 7: Combined expressions with host disjuncts can yield mostly relevant spill queries as opposed to host-based pure output expressions.

In Figure 4(b) we plot the average percentage of negative spill for combined expressions. The figure is similar to Figure 3(b) for pure output expressions. Note that in both figures the lines for 2-URL and 2-mixed expressions are have almost no changes, while the line for 2-host expressions shows a dramatic negative spill decrease in combined expressions compared to output expressions.

Result 8: Output expressions are suitable for matching 30%-70% of queries targeted by a keyword bidding ad.

In Figure 4(c) we show the percentage of the input query sets that are covered by the output expression part. The semantics of the x-axis is similar to the previous figures. The fourth triangle from the left shows that for $\gamma = 0.9$ the output part of a combined expression is used to cover on average the 50% of the input query set. Note that the percentage of queries covered by the output part is between 30% and 70% for all three variations. So, the keyword and the output part of combined expressions appear to be equally important for expressing the advertiser's intent.

7. RELATED WORK

For search advertising, Fain and Pedersen [12] provide a short review, Jansen and Mullen [17] provide a longer one, while Feldman and Muthukrishnan [13] review the area with a more algorithmic focus. Wikipedia is also a good source, especially for some of the early papers (e.g., Davis et al. [11]).

Output bidding in its general form is originally proposed in a patent by Dasdan [8] and is further discussed in a position paper by Dasdan et al. [10]. Dasdan and Gonen [9] develop output bidding further and propose an auction model for bidding for bundles of search results from output. Since the problem of handling arbitrary bundles has high complexity [7], the authors reduce the problem complexity by decomposing bundles from advertisers into smaller bundles that can contain at most one organic result and at most one sponsored result. Ghosh et al. [16] propose algorithms for computing optimal bundles for input bidding. Their proposal can also be useful for output bidding. More general auction models for bundles may use techniques from combinatorial auctions [7].

A variation of output bidding for associating ads with site names in a SERP is independently proposed by Manavoglu et al. [19]. The authors state that the site owner is not required to participate in the auction and that the ad matching to a site in SERP may also be independent of the input query that generated the SERP.

Apart from these references, we are unaware of any other work on output bidding. However, there is related work on using parts of output for input bidding, which is presented next.

Allowing bids on more parameters for input bidding is discussed by Aggrawal et al. [1] and Muthukrishnan et al. [21]. The former work suggests allowing advertisers to specify bids and a preference for positions in the list of ads, while the latter allows advertisers to affect the number of ads shown. In both references, the authors show how to generalize the current auction models to handle these new constraints. We consider these references as related to output bidding because the new parameters for bids refer to parameters of sponsored search section, hence, those of output.

Broder et al. [3] and Choi et al. [6] study query expansion using web search results and ad expansion using landing pages respectively to improve the sponsored search ad relevance. These approaches use the search results for the ad selection and they are relevant to output bidding. However, the authors view the use of the search results as a means to eliminate the need for advertiser bidding rather than a opportunity to augment the bidding language and give more control to the advertiser over the ad targeting.

In the appendix we discuss that an implementation of output bidding that hides the extra latency from the user can rely on results caching. Search results caching is also studied by Gan et Suel [15] and Cambazoglu et al. [4]. Both works show that a high hit rate is possible and this conclusion supports the implementation feasibility of output bidding.

8. CONCLUSIONS AND FUTURE WORK

Modern search engines are becoming more and more successful in correctly identifying users' search intents and satisfying users' needs. However, advertisers who use keyword bidding cannot benefit from the advancements of search engines. To come up with appropriate keywords they need to "reverse engineer" the way users translate their intents into queries.

Our results suggest that advertisers can use output bidding to express their interests compactly and rely on the search engine to identify the relevant users. In this way, the advertisers not only spend fewer resources to create their ads, but they also can reach more relevant users. In total, we believe that output bidding represents an exciting new way to match advertisers to potential customers, giving advertisers a powerful novel way to specify their interests.

In our future work we plan to investigate other variations of output bidding beyond URL bidding. For example, with output bidding an advertiser could bid on anything on the output page, e.g., an image or some text string. They could also bid on types of URLs, e.g., from a given country, from a particular content category (say sports), or the ranking of URLs. They could also specify negative conditions on the output, for instance, an advertiser may not want their ad to appear on any results page that includes pornography sites. For all these variations we need to evaluate their benefits with respect to keyword and URL bidding.

Another direction for future work is the design and evaluation of a system that will serve output bidding ads. The discussion in Section A of the Appendix is the first step towards this goal. Among the unresolved issues is the auction design for output bidding.

Finally, a real-world implementation of output bidding will allow a user-level comparative study between keyword and output bidding. In this paper we used the number of keywords as the cost metric for keyword bidding and the number of disjuncts as the cost metric for output bidding. An experiment with real advertisers could help to accurately interpret the relation between the two cost units. In such an experiment advertisers would be asked to design campaigns in both mechanisms and then we could compare the spent effort, i.e., human hours and used resources, as well as the success of each campaign, i.e., covered relevant queries versus spill queries. Note that the requirement for a real-world experiment setting is essential to avoid introducing any bias, e.g., difference in the effort spent in a test setting versus effort spent in real setting with money at risk.

9. REFERENCES

[1] G. Aggarwal, J. Feldman, and S. Muthukrishnan. Bidding to the top: VCG and equilibria of position-based auctions. In *Proc. WAOA*, Zrich, Switzerland, 2006.

[2] A. Anurag, M. Cutts, J. Dean, P. Haahr, M. Henzinger, U. Hoelzle, S. Lawrence, K. Pfleger, O. Sercinoglu, and S. Tong. Information retrieval based on historical data. US Patent Pub. App. No. 20050071741, filed in 2005.

[3] A. Z. Broder, P. Ciccolo, M. Fontoura, E. Gabrilovich, V. Josifovski, and L. Riedel. Search advertising using web relevance feedback. In *Proc. CIKM*, pages 1013–1022, New York, NY, USA, 2008.

[4] B. B. Cambazoglu, F. P. Junqueira, V. Plachouras, S. Banachowski, B. Cui, S. Lim, and B. Bridge. A refreshing perspective of search engine caching. In *Proc. WWW*, pages 181–190, New York, NY, USA, 2010.

[5] R. D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *Proc. SODA*, pages 345–353, Philadelphia, PA, USA, 2000.

[6] Y. Choi, M. Fontoura, E. Gabrilovich, V. Josifovski, M. Mediano, and B. Pang. Using landing pages for sponsored search ad selection. In *Proc. WWW*, pages 251–260, New York, NY, USA, 2010.

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

[8] A. Dasdan. System for displaying advertisements associated with search results. US Patent Pub. App. No. 20080270228, filed in 2007.

[9] A. Dasdan and R. Gonen. System and method for offering an auction bundle in an online advertising auction. US Patent Appl., 2008.

[10] A. Dasdan, K. Santanu, P. Papadimitriou, and H. Garcia-Molina. Output bidding: A new search advertising model complementary to keyword bidding. In *Ad Auctions*, pages 1–6, Stanford, CA, USA, 2009.

[11] D. J. Davis, M. Derer, J. Garcia, L. Greco, T. E. Kurt, T. Kwong, J. C. Lee, K. L. Lee, P. Pfarner, and S. Skovran. System and method for influencing a position on a search result list generated by a computer network search engine. US Patent 6269361, 2001.

[12] D. C. Fain and J. O. Pedersen. Sponsored search: a brief history. *Bulletin of ASIS&T*, 32:12–13, 2006.

[13] J. Feldman and S. Muthukrishnan. Algorithmic methods for sponsored search advertising. In *Proc. SIGMETRICS*, pages 91–124, Annapolis, MD, USA, 2008.

[14] A. Fuxman, P. Tsaparas, K. Achan, and R. Agrawal. Using the wisdom of the crowds for keyword generation. In *Proc. WWW*, pages 61–70, New York, NY, USA, 2008.

[15] Q. Gan and T. Suel. Improved techniques for result caching in web search engines. In *Proc. WWW*, pages 431–440, New York, NY, USA, 2009.

[16] A. Ghosh, H. Nazerzadeh, and M. Sundararajan. Computing optimal bundles for sponsored search. In *Proc. WINE*, pages 576–83, 2006.

[17] B. J. Jansen and T. Mullen. Sponsored search: An overview of the concept, history, and technology. *Int. J. Electronic Business*, 6(2):114–31, 2008.

[18] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the web for emerging cyber-communities. *Comput. Netw.*, 31(11-16):1481–1493, 1999.

[19] E. Manavoglu, A. Popescul, B. Dom, and C. Brunk. System for targeting data to sites referenced on a page. US Patent Appl., 2008.

[20] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[21] S. Muthukrishnan. Bidding on configurations in internet ad auctions. In *Proc. COCOON*, pages 1–6, Niagara Falls, NY, USA, 2009.

[22] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1:19–30, 2010.

[23] P. Papadimitriou, H. Garcia-Molina, A. Dasdan, and S. Kolay. Output url bidding. Technical Report 969, Stanford InfoLab, 2010.

APPENDIX

A. IMPLEMENTATION

In this section we study different architecture alternatives for the efficient retrieval of output URL bidding ads. We focus on 1-URL expressions to simplify the presentation of the different alternatives. However, in our discussion we briefly discuss the changes required for expressions with different literal type or greater k.

A.1 Ad Indexing

The search engine needs to fetch ads by URLs. For example, if the URL u appears in the result URL list U_q, the search engine needs to fetch every ad $a \in A$ that contains u in some literal. To achieve this the search engine uses an inverted hash index. Say, for example, that set A contains three ads as in Figure 5(a). The search engine will use the hash index of Figure 5(b) that we denote as $I(U \rightarrow A)$ to retrieve all ads that correspond to the URLs in a list U_q. The index keys are the URLs and a posting list for the URL/key u contains all ads with u in their boolean formulas.

In cases of ads with conjunctions, i.e., k-ads with $k \geq 2$, the search engine still uses the same index to find for each conjunction the intersection of the conjunction URLs posting lists. For example, for the conjunct $u_1 \wedge u_2$ the search engine has to intersect the lists $[a_1]$ and $[a_1, a_3]$ that correspond to u_1 and u_2 respectively. In this paper we do not discuss the intersection calculation, but the reader can find efficient algorithms for this in any introductory information retrieval book such as [20]. In cases of host and mixed ads the search engine needs to maintain an index $I(H \rightarrow A)$ and $I(H \cup U \rightarrow A)$ respectively.

The search engine maintains also other similar indexes to facilitate fast look ups. The size of an index $I(X \rightarrow Y)$ depends on the size of X and the average number of elements in Y that an element in X points to. In the rest of the paper, we assume that a look up operation in such an index takes constant time independent of the index size.

A.2 Architectures

In existing advertising mechanisms sponsored search results are independent from organic search results. So in a typical search engine architecture a user query q is submitted to two different components whose results are merged to form the SERP. We illustrate this architecture in Figure 6. The latency l to return a SERP for a user query is:

$$l = \max(t_o, t_s) \tag{7}$$

where t_o is the running time of the organic search component and t_s is the running time off the sponsored search component. In real systems t_o is 3 to 4 times longer than t_s. Therefore in this architecture $t = t_o$.

Such an architecture is not directly applicable to cases where the sponsored search component relies on the organic search component e.g., in our case the input to sponsored search component is not the the query q but the URLs that correspond to the results in the SERP. We discuss the alternative options to handle this dependency in the following subsections.

$$\begin{aligned} A &= \{a_1, a_2, a_3\} \\ a_1 &= u_1 \vee u_2 \vee u_3 \\ a_2 &= u_3 \vee u_4 \\ a_3 &= u_2 \vee u_4 \end{aligned}$$

(a) Ads

URL	Ads
u_1	a_1
u_2	a_1, a_3
u_3	a_1, a_2
u_4	a_2, a_3

(b) Index

Figure 5: Ad Indexing.

A.2.1 Serialization

The naive and most straight forward approach to is the serialization of the organic and the sponsored search procedures. Such an approach requires the following changes to the search engine architecture: Every query is submitted initially only to the organic search component. This component outputs the organic results that serve as input to the sponsored search component. Figure 7(a) illustrates the modified architecture.

The latency in this case is equal to the sum of the running times of the organic and the sponsored search component:

$$l = t_o + t_s \tag{8}$$

The advantage of this approach is simplicity and its disadvantage is the increased latency.

A.2.2 Pipelining

The need for serialization comes from the fact that the sponsored search component requires the URLs that appear in the top SERP results. However, the output of the organic component is not restricted to these URLs, but it also includes some information that is not used by the sponsored search component. The search component retrieves all the pages that are relevant to a user query, it ranks them and then it does some post-processing that can be ignored by the sponsored search component. The post-processing includes tasks such as the selection of query-dependent summarizing snippets for each result page and results clustering. So we can split the search component into two subcomponents: (a) the *retrieval-ranking* subcomponent that includes the index of terms to document and the ranking function; and (b) the *post-processing* subcomponent that handles the post-processing tasks.

To reduce latency we can forward the top ranked results from retrieval-ranking subcomponent to the sponsored search component and run the post-processing tasks and the ad selection in parallel. We illustrate the modified architecture in Figure 7(b), where O_r stands for the ranking-retrieval subcomponent and O_p stands for the post-processing subcomponent. Thus, the the running time of the search component is $t_o = t_r + t_p$ where t_r is the running time of the O_r sub-component and t_p is the running time of O_p. Then the latency to obtain a SERP is:

$$l = t_r + \max(t_p, t_s) \tag{9}$$

Note that if the post-processing step takes more time than the the ad retrieval, i.e., $\max(t_p, t_s) = t_p$, then the latency of this approach depends only on the running time of the organic search component, which is the case in existing advertising mechanisms. However,

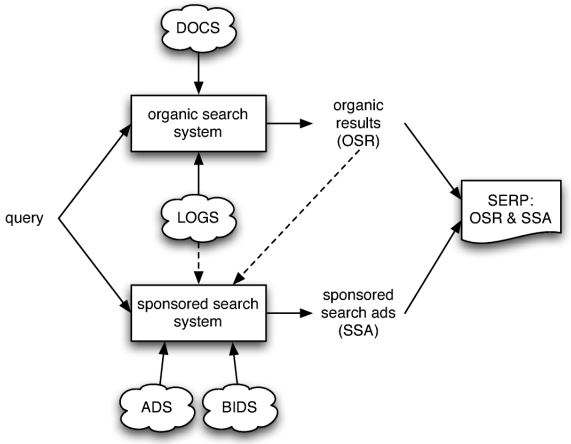
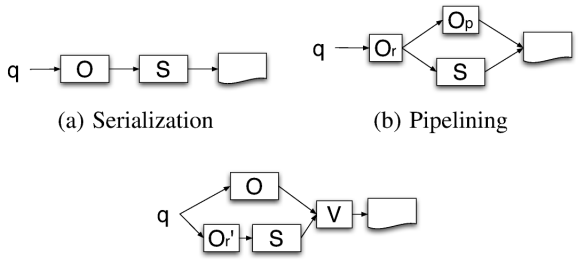


Figure 6: General Sponsored Search System Architecture.



(a) Serialization (b) Pipelining

(c) Parallelization

Figure 7: Simplified search engine architecture in different implementation alternatives. O-box stands for the Organic Search component and S-box stands for the sponsored search component.

the disadvantage of this approach is the requirement to introduce some communication and the two components that are conventionally considered as independent. In many cases it is not trivial to drop this convention since it may even require changes in the search engine resource allocation, e.g., store ad and page data and corresponding indexes to the same data center instead of dedicated data centers.

A.2.3 Parallelization with Redundant Computation

We can eliminate the need for communication between the two components at the cost of some redundant computation in the sponsored search component. The brute-force approach is to create a replica O'_r of the O_r component and make it part of the sponsored search component. In such an approach the flow chart that describes the interaction between the two components (Figure 7(c)) during the query processing is similar to Figure 6. The latency will be the same as in the pipelining approach.

However, we can reduce the required storage space and the document retrieval time in the sponsored search subcomponent by reducing the size of the document index it maintains. Note that O'_r needs only to index documents for which there is at least one associated ad. Given query q, component O'_r will assign a relevance score only to such documents and then S will retrieve the ads that match the top-ranked ones. The documents in O'_r output will have the same relative ranking order as in O_r output, because O and O_r use the same scoring function. However, since there are fewer documents in O'_r compared to O, a validation component V is required to eliminate any ads fetched by S that correspond to documents that do not appear in the top-ranked documents of O.

Maintaining a smaller index affects not only the amount of data that should be replicated, but also the running time t'_r of the document retrieval and the scoring/ranking procedures. Even if the look up time in the index is the same, joining the results of word look ups and ranking the results takes less time in the presence of a smaller index. So we expect that $t'_r < t_r$. If V takes t_v time to validate the ads, the latency in this case will be:

$$l = max(t_o, t'_r + t_s) + t_v \tag{10}$$

Note that t_v is expected to be negligible, because the V input contains only the top-ranked documents of O_r and O'_r and the ads that correspond to top-ranked O'_r documents.

A.3 Caching

For every submitted query q the search engine has to retrieve the result URLs U_q and then the set of ads A_q that are associated with U_q. To avoid the computational effort to retrieve these sets for every submission of the same query q the search engine can calculate these sets once and cache the results. So the next time that query q is submitted the sponsored search component may find the ads to display to the query q with a simple look up operation in a hash index. Caching may be the only option if the on-demand calculation of U_q or A_q is so time-consuming that it increases dramatically the response time to query q.

Although caching has a positive impact on the response time, it comes with the cost of extra storage space and the effort to keep the cache up to date. In the following subsections we explore different caching variations and we discuss their implications in latency, storage space, cache maintenance effort and search engine revenue.

A.3.1 Results vs Ads

For every query q the search engine may cache (1) the corresponding URLs U_q or (2) the corresponding ads A_q.

In the first case the sponsored search component caches an index of queries to URLs $I(Q \rightarrow U)$ and uses the index $I(U \rightarrow A)$. When a new query q is received, first, the search engine retrieves all the URLs U_q that will appear in the organic results with one look up in index $I(Q \rightarrow U)$. Then, it uses index $I(U \rightarrow A)$ to look up the ads A_u that correspond to each URL $u \in U_q$. The query ads A_q is the union of these ads, i.e., $A_q = \cup_{u \in U_q} A_u$. The total number of look up operations is $1 + |U_q|$.

In the second case the sponsored search component caches the index $I(Q \rightarrow A)$. Using this index the search engine retrieves ads A_q with one look up operation. The disadvantage of this approach is that the cache updates in case of a new ad a are more expensive, since we do not know which of the cached queries contain any a URL in their query results. Hence, to update the cache we need to submit all cache queries that and check their results against the new ad a. To avoid this expensive operation, we may allow the cache to be inconsistent for some time period and do batch updates periodically.

A.3.2 Static vs Dynamic Cache

The decision of what to cache is either off-line (static) or online (dynamic). A static cache is based on historical information and is periodically updated. For example, we can choose to fill the cache once a day with most frequent queries of the previous day and leave it unchanged for 24 hours.

A dynamic cache replaces entries according to the sequence of queries. When a new query arrives, the cache system decides whether to evict some query from the cache in the case of a cache miss. Such online decisions are based on a cache policy, and several different policies have been studied in the past. In this paper we only consider the *Least Recently Used* or LRU policy that discards the least recently used query from the cache.

A.3.3 Cache Miss Handling

A cache miss refers to a failed attempt to retrieve ads (or URLs) for a query q using the cache. Cache misses are inevitable, since the search engine cannot store all possible queries. Even if the search engine had unlimited space to cache ads for all queries found in the query log, there is always the possibility of a new query.

In case of a cache miss in a query q, the search engine has two alternative options to handle it:

- The search engine can ignore the cache miss and display no ads to user who submitted q. The advantage of this approach is that the user experiences no latency in the display of the search results that consist solely of the organic search results. However, the search engine misses an opportunity of possible revenue, since by displaying no ads there is no possibility of a user ad click. Note in case of dynamic cache the URLs or

Cluster Queries	Common Hosts
compare ipod and zune, zune vs ipod, zune vs i pod, microsoft zune vs ipod comparison	electronics.howstuffworks.com www.dvdtozune.com www.roughlydrafted.com www.zunescene.com
hard dirves, 60gb ipod dimensions, seagate sata momentus 5400.2 60gb, lcd computer monitor, dvd burner, usb flash, vizio p42hdtv10a plasma hdtv, fellowes 15' heavy duty indoor extension cord, computer video card, lcd computer monitors, computerhardware, dvd +- burner, 8.5" philips dcp851 case, hard drives...	www.newegg.com www.amazon.com www.overstock.com www.geeks.com
appl, goog, czyb, nsrgy, wyn, ctxs, lfxg, crox, "nke", tgt, goog, celg, gnw, orcl, ilmn, lyg, cwtre, tscdy, cdns, mrk nyse, csco, qtww, lihr, cwtr, bmy, emkr, peix, .fcel, vlgea, hpq, lehmq, msft, nke, eqix, fxi, vlkay, lly nyse, sbkc, cwtry, nabzy, ntdoy, clne, fcel, mdas, xlf, sbux, blk, swdby, spls, yge, chkp, trmb, ms morgan, mblx	finapps.forbes.com www.marketwatch.com www.google.com finance.yahoo.com

Table 1: Cluster examples for $i = 2$ and $j = 4$.

ads that match with q are still computed after the results page (with no ads) is displayed. The results are then added to the cache so that NEXT time q is seen the cache will have the necessary data.

- The search engine can retrieve the ads A_q that correspond to q by applying one of the methods we discussed in Section A.2. In this case the search engine does not miss the opportunity to get revenue from user interested in the sponsored content. However, depending on the method that is used to find relevant adds, the user may experience an increased response time to his query.

B. QUERY CLUSTERING

In this section describe the clustering method that we employ in the spill evaluation approach we present in Section 4. First, in Section B.1, we present the algorithm that we use, and then, in Section B.2, we present the clusters that we obtain by applying this algorithm to the dataset that we describe in Section E.1. We use these clusters for the spill evaluation in Section 6.3 of the experiments.

B.1 Algorithm

After experimenting with several query clustering schemes, we selected the algorithm proposed by Kumar et al. [18]. This algorithm not only yields good results (see Section B.2) but is one of the few that is scalable to the very large query sets we have. The algorithm clusters queries based on their results: (a) if a URL appears in the search results of two queries, then these two queries have similar search intents; and (b) the more URLs two queries have in common the more similar their intents are.

According to [18], all of the similar intent queries and their result URLs form a dense bipartite subgraph. It is also shown that such a dense bipartite subgraph contains at least one *i, j-core*, that is a complete bipartite graph with at least i query nodes and at least j URL nodes. For example, in Figure 1(b) the subgraph of q_1, q_2 and u_1, u_2 is a 2, 2-core.

The values of parameters i and j control the granularity of the clustering. For example, if $i = 2$ and $j = 2$ then the query clusters in Figure 1(b) are the following: $\{q_1, q_2\}$ as part of 2, 2-core, $\{q_3, q_4, q_5\}$ as part of the 3, 3-core and $\{q_7\}$ that is not part of any core. However, if $i = 2$ and $j = 3$ then there are 4 clusters: $\{q_1\}$, $\{q_2\}$, $\{q_3, q_4, q_5\}$ and $\{q_7\}$.

Note that in the aforementioned examples we treated every unclustered query as a singleton cluster. Although there are works [18, 14] that suggest how to assign such queries to clusters, we followed this conservative approach. As a result, the estimated positive spill is only a conservative estimate of the actual positive spill and more aggressive clustering approaches can yield better results.

B.2 Results

We clustered the queries of our dataset by applying the trawling algorithm to the bipartite graph $G = (Q, H, E)$ of queries Q and the hosts H. Recall that each of the output clusters contains at least i queries, and that all queries in a cluster have at least j hosts in common in their search results. In the following paragraphs we discuss the selection of parameters i and j.

The value of i controls the minimum size of each cluster. For the spill evaluation task even a cluster that contains only two queries is important, since if either of them is included in some set R, the other can be part of the positive spill of the corresponding output expression. So, we selected $i = 2$.

The value of j implicitly controls the cohesiveness of each cluster, i.e., the strength of the intra-cluster query similarity. For example, for $j = 1$ clusters may contain very diverse queries, since the only common host in a cluster may be a host with diverse web pages such as `en.wikipedia.org`. On the other hand, for $j = 10$ all the queries in one cluster have exactly the same hosts in their results, and such queries are usually almost identical. To find the minimum value of j that yields clusters that can be used for the spill evaluation, we did the clustering for $j = 1, 2, ...10$ and we manually examined the results in each case to see whether the clusters are appropriate for the spill evaluation. After such an evaluation, we determined that clusters with $j \geq 3$ or 4 are appropriate for the spill evaluation task, since such clusters contain queries that we believe that express the same advertiser intent. We selected $j = 4$ and the numbers of positive spill queries that we present are only conservative estimates of the values we would obtain with a looser clustering, e.g., $j = 3$.

The total number of clusters for $i = 2$ and $j = 4$ is 62,727 (note that each of 12M queries that we cluster may belong to many clusters). Moreover, the percentage of queries that do not belong to any cluster is approximately 27%.

In Figure 1 we show some cluster examples for $i = 2$ and $j = 4$. The first cluster contains different rewrites of a query about the comparison of Zune and iPod. The results of all of the cluster queries include pages from all of the 4 cluster hosts such as `electronics.howstuffworks.com`. The second cluster contains queries about electronic products such as hard disks and monitors that are sold by at least 4 online stores such as `www.newegg.com`. Finally, the last cluster contains the stock symbols of different companies such as AAPL for Apple or GOOG for Google. The result pages of all these queries contain pages from the 4 popular financial sites that we show in the table. In all three cases, we can argue that an advertiser that is interested in one of the cluster queries would probably be interested in the rest of the queries as well.

C. SPILL INTEGRATION IN THE QUERY SET OUTPUT COVER PROBLEM

Using the partition of queries to positive and negative spill, we can revisit the penalization of the spill term in the Query Set Output Cover problem. In particular, we can replace $|spill(a,R)|$ with a linear combination of positive and negative spill:

$$\beta|spill^{+}(a,R)| + (1-\beta)|spill^{-}(a,R)|, 0 \leq \beta \leq 1. \quad (11)$$

The parameter β is analogous to parameter γ and it handles the penalization weighting between the two spill terms. To penalize the negative spill more than the positive spill, its value must be in the range $\beta \in [0, 0.5]$.

The new objective requires changes in Algorithm 1. First, the variables $S[u]$ that keep track of the URL spills are replaced by $S_{pos}[u]$ and $S_{neg}[u]$. Similarly, variable $spill$ is replaced by $spill_{pos}$ and $spill_{neg}$. Second, the lines 7, 10, 12 and 15 must change to account for the different spill types. Since the rest of the changes are straightforward, we only focus on the greedy decision in line 12. The spill term $|S[u] - spill|$ in this line becomes $\beta|S_{pos}[u] - spill_{pos}| + (1-\beta)|S_{neg}[u] - spill_{neg}|$.

We have included this analysis for completeness, but we do not use the revised Algorithm 1 in our experiments. Our clustering only yields an approximation to the advertiser's intent and our data does not contain advertiser evaluations of spill queries that would be required to accurately estimate β. So in our current experiments we use the same penalization for both positive and negative spill. Such experiments provide conservative results about the actual performance of output URL bidding. If we decrease or remove the penalization of positive spill, then the selected URL in the greedy step of Algorithm 1 will yield less additional negative spill and, therefore, the total spill of the resulting output expression will be more relevant to the input query set.

D. KEYWORD AND OUPUT BIDDING

In this section we provide a qualitative comparative discussion between keyword and output bidding and we highlight the differences of the two models from the advertiser perspective.

The two models differ fundamentally in the way new campaigns are created. Say that an advertiser wants to create a campaign for the upcoming Olympic Games. In case of input bidding the advertiser has to simulate the user role and come up with queries that will be submitted by users interested in this event, e.g., event name, host city name, country team names, sports names, player names and so on. In case of output URL bidding the advertiser has to think of the URLs that are the most authoritative and popular sources of information for this event, e.g., the official games web site such as `www.london2012.com`, a popular news web site that covers the event such as `espn.go.com/olympics/` and so on. In the former case, the more accurately the advertiser can predict the user query behavior, the more successful the campaign will be. In the latter case, the more successful the search engine is in identifyingq the user intent and showing the most relevant results, the more successful the campaign will be (assuming that it is easy for the advertiser to identify the most authoritative URLs with trying few searches). Since neither the advertisers nor the search engines can be perfect in their roles, it seems that successful campaigns would require combining the two models. Results 5, 7 and 8 from our experiments confirm this intuitive claim.

After creating a campaign, the advertiser has to monitor its performance and make adjustments. The monitoring task is different in the two models and depeding on the campaign it may be easier for either of the models. In case of keyword bidding the advertiser has to follow the user query trend. A recent study on search engine caching [4] shows that an infinite query cache can achieve a maximum hit rate of 0.6. That means that there is a constant stream of new queries that the advertiser needs to predict to keep up with user query trends. In case of output bidding the advertiser has to monitor both the creation of new related URLs as well as changes in the search engine ranking. Regarding new URLs, a recent study [22] shows that two web graphs that are obtained one month apart and correspond to a very dynamic part of the web (news web sites) have in common 50% of their nodes/hosts and edges/hyperlinks. Note that this percentage should be much higher in a random part of the web graph. So, if we also take into account that search engines favor old web sites over new ones in their rankings [2], it seems that monitoring for new URLs in the search results is less challenging than monitoring for new queries. Regarding changes in the search engines, note that the goal of such changes is to improve results relevance. As long as they achieve this goal, output bidding campaigns can only benefit from such changes. For example, after a ranking algorithm improvement, URLs related to advertiser interest should appear in the results of fewer irrelevant and more relevant queries. Altough we need to validate this claim in future work, monitoring output bidding campaigns does not seem to require more advertiser effort compared to keyword bidding campaigns.

In our future work we plan to study algorithms and create tools that will help advertisers create and monitor their output bidding campaigns.

E. DATASET AND IMPLEMENTATION

E.1 Dataset

Our dataset contains 12,931,117 unique queries obtained from Yahoo! search engine as part of an one day query log from November 2009. For each query the log contains the URLs of the top-10 organic search results as well as the union of all ads that were shown to users who submitted the query. The organic search results included in total 62,666,514 distinct URLs that come from 7,185,392 distinct hosts. The number of different ads that appeared in the dataset is proprietary information that we cannot disclose.

E.2 Implementation

We implemented Algorithm 1 in Python and we used PyLucene to index the queries of each ad, the URLs of each query and the hosts of each query. Our implementation takes 0.2-10 hours to find the logical expressions for all 2,251 ads on a Linux Server with two Xeon 5335 processors and 64GB of memory. The running time is affected by the variation type and the value of γ. The longest running time corresponds to the generation of 2-mixed expressions with a high γ value and the shortest running time corresponds to the generation of 1-URL ads with a low γ value. Although the worst case algorithm performance does not depend on γ, in practice, the value of γ affects significantly the running time of the algorithm. This is a result of structural properties of the Queries-Result URLs bipartite graph, such as the power-law distribution of the URL node degrees [23]. With such a distribution a smaller spill penalization, i.e., a higher value of γ, results in the selection of a more popular URL, i.e., a URL u with higher degree $d(u)$, in every algorithm iteration and such selections increase the algorithm running time.