Shared Winner Determination in Sponsored Search Auctions

David J. Martin [#1], Joseph Y. Halpern [#2]

[#]*Computer Science Department, Cornell University*
Ithaca, NY, USA
[1]djm@cs.cornell.edu
[2]halpern@cs.cornell.edu

***Abstract*— Sponsored search auctions form a multibillion dollar industry. Search providers auction advertisement slots on search result pages to advertisers who are charged only if the end-user clicks on the advertiser's ad. The high volume of searches presents an opportunity for sharing the work required to resolve multiple auctions that occur simultaneously. We provide techniques for efficiently resolving sponsored search auctions involving large numbers of advertisers, with a focus on two issues: sharing work between multiple search auctions using shared aggregation and shared sort, and dealing with budget uncertainty arising from ads that have been displayed from previous auctions but have not received clicks yet.**

I. Introduction

With Internet search being a necessity for most Web users, search result pages have become a thriving advertising platform. The results of a search query are presented to the user as a web page that contains a limited number of slots for advertisements (typically between four and twenty). On each search result page, the major search engines, like Google and Yahoo!, sell these slots to advertisers via an auction mechanism that charges the advertiser only if a user clicks on the ad. 99% of Google's revenue and more than half of Yahoo!'s revenue comes from their sponsored search auctions [1]. And the market size is growing fast. By 2008, spending by US firms on sponsored search is expected to increase by $3.2 billion from 2006 and will exceed $9.6 billion, which was the amount spent on all of online advertising in 2004 [2]. Furthermore, 44% of the current search engine advertisers joined the market within the last two years [2]. With the increase in the market size and the high volume of searches in mind, our goal is to provide techniques for efficiently resolving sponsored search auctions involving large numbers of advertisers, with a focus on two important issues: sharing work between multiple search auctions, and dealing with budget uncertainty arising from previous auctions whose outcomes have not yet been decided. In the remainder of this introduction, we describe the problem and how we solve it in a little more detail.

When a user submits a search query, the search provider returns the search results along with a set of advertisements in designated advertisement slots on the search result page. These advertisement slots are sold to advertisers via an auction where advertisers specify a bid representing the maximum amount that they would pay for a click. Advertisers pay the search provider only if and when the user clicks on their ad.

When the user does click on an advertiser's ad, the price that the advertiser is charged is determined by a (publicly known) pricing rule used by the search provider. For example, with a first-price pricing rule, the advertiser is charged the amount he bid. In practice, more complex pricing rules, such as Vickrey pricing [3], [4] and generalized second-pricing (used by Google and Yahoo!) [1], [5], are used because they lead to desirable economic properties such as truthful bidding, local envy-freeness, etc. The pricing rules employed all satisfy the constraint that the price charged to an advertiser does not exceed his bid. More important for our purposes, all the pricing rules currently used proceed by first solving the *winner-determination problem* (e.g., [1], [5], [6], [4]).

In our setting, winner determination is the problem of assigning the k available ad slots on a search result page to the n interested advertisers so as to maximize the total expected amount of bids realized, subject to the constraint that no advertiser gets more than one slot.[1] An advertiser's bid of $b dollars is said to be *realized* if the advertiser's ad is displayed in some slot and the user clicks on the advertiser's ad. Thus, the total expected amount of bids realized depends on the advertisers' bids as well as on their *click-through rates*, where ctr_{ij}, the click-through rate of advertiser i in slot j, is the probability that a user will click on advertiser i's ad if it is in slot j. The winner-determination problem can be written as the following integer program:

$$\max_{x_{ij}} \sum_{i\in[n]} \sum_{j\in[k]} x_{ij} ctr_{ij} b_i$$

subject to

$$\begin{gathered}\forall\, i \in [n], j \in [k] \,.\, 0 \le x_{ij} \le 1 \\ \forall\, j \in [k] \,.\, 0 \le \textstyle\sum_{i=1}^{n} x_{ij} \le 1 \\ \forall\, i \in [n] \,.\, 0 \le \textstyle\sum_{j=1}^{k} x_{ij} \le 1,\end{gathered}$$

where we use the notation $[m]$ to denote the set $\{1, \ldots, m\}$, the variable x_{ij} represents whether or not advertiser i is assigned to slot j, and b_i denotes the value that advertiser i bids for a click. The objective function maximizes the expected return of the assignment (assuming that advertisers pay what they bid). The first set of constraints represent the condition

[1]The constraint that no advertiser gets more than one slot is imposed to prevent a rich advertiser from monopolizing all the slots.

that the x_{ij}s are Boolean variables (since this is an integer program). The second set of constraints represent the condition that no two advertisers can be assigned to the same slot. The third set of constraints represent the condition that no advertiser is assigned to more than one slot.

While mechanisms currently in use differ in what pricing rule they use after running winner determination, they all use winner determination as a first step. It is therefore very important to solve winner determination as quickly as possible, especially since winner determination needs to occur *before* a search result page is returned, so it contributes to the latency experienced by the user. As we show here, we can reduce the total latency for winner determination by sharing the winner-determination computation across multiple auctions. We show that by reusing some information, we can considerably save on computation time.

In order to share work between auctions, we exploit the fact that there are often related search queries that occur nearly simultaneously. For example, according to the SEO Book Keyword Tool [7], there were over 300,000 music-related searches per day in June 2008, giving an average of over 1 music-related searches every $1/3$ seconds on average. If we batched auctions into rounds consisting of $2/3$ second intervals (well within the limits of user tolerance studies [8]), then we would expect to see 2 music-related auctions per round. Now it is reasonable to believe that there will be many advertisers (e.g., music stores) who will be interested in advertising in both music-related searches. Our techniques take advantage of such shared data in the winner determination problems of multiple related auctions to reduce the amount of work required to find the winning bidders for each auction.

A. Outline

The rest of this paper proceeds as follows. Section II describes our approach to sharing the winner determination computation across multiple auctions using top-k aggregation to find the winning k advertisers for different bid phrases under a slightly restrictive assumption, called separability. We examine the effect of the algebraic properties of top-k aggregation on the complexity of finding an optimal sharing plan. We show that the problem is inapproximable and provide a reasonable heuristic for finding a shared plan. In Section III, we provide a technique for sharing work in a more general setting using a novel shared sort. In Section IV, we examine the issue of uncertain budgets arising from ads which have been recently displayed but have not yet received a click. We demonstrate a way to egregiously game the system if this issue is ignored and there is a high volume of search. We provide a principled approach to dealing with budget uncertainty, along with efficient techniques for determining the top k advertisers in the face of this uncertainty. We discuss related work in Section V, where we show how to adapt our techniques to the situation where the separability assumption does not hold. Finally, we conclude in Section VI.

ctr_{ij}	slot 1	slot 2
advertiser A	0.36	0.24
advertiser B	0.33	0.22
advertiser C	0.39	0.26

Fig. 1. Separable Click-Through Rates

	A	B	C
c_i	1.2	1.1	1.3

	slot 1	slot 2
d_j	0.3	0.2

Fig. 2. Advertiser-Specific and Slot-Specific Factors

II. Shared Aggregation and Winner Determination

Given the high volume of searches performed each day, several search queries arrive nearly simultaneously at any given time. This presents an opportunity for sharing the work of winner determination among several sponsored search auctions. In order to identify the work that can be shared across auctions, we need to first describe how winner determination is solved for an individual auction.

A. Separability and Winner Determination

The probability that a user clicks on an advertiser's ad depends on, among other things, the content of the ad and the slot in which the advertisement is displayed. (For example, studies have shown that ads are more likely to be clicked on if they are displayed in slots at the top of a vertical list of slots than if they were placed lower in the list [9].) The assumption made by existing systems [1], [5], [6], [4], called the *separability assumption*, is that the probability that a given ad receives a click when displayed in a given slot can be written as the product of two factors, one that depends only on the advertiser and the other that depends only on the slot position. This probability, denoted as ctr_{ij}, is called the *click-through rate* of advertiser i in slot j. The separability assumption then says that $ctr_{ij} = c_i \times d_j$ where c_i is the advertiser-specific factor and d_j is the slot-specific factor. Figure 1 shows an example of separable click-through rates that can be decomposed into advertiser-specific factors and slot-specific factors, as depicted in Figure 2.

Under the separability assumption, winner determination can be solved in time linear in the number of advertisers for any given auction. Since each ctr_{ij} is separable as $c_i \times d_j$, winner determination is equivalent to finding one-to-one mapping α from slots to advertisers so as to maximize $\sum_{j \in [k]} b_{\alpha(j)} c_{\alpha(j)} d_j$. Then α dictates the allocation of slots: slot j is assigned to advertiser $\alpha(j)$. Without loss of generality, assume that the slots are ordered such that slot j has the jth highest value of d_j. We can then solve the winner-determination problem by simply finding the advertisers with the top k values of $b_i c_i$ and setting $\alpha(j)$ to the advertiser with the j-highest value of $b_i c_i$. This requires a single scan over the

	A	B	C
b_i	14	15	10
$b_i c_i$	16.8	16.5	13

Fig. 3. Bids

$b_i c_i$s, keeping track of the top k advertisers. For the remainder of this paper, for ease of exposition, we assume separability, in accord with the earlier literature. However, we remark that in earlier work [10], we showed that winner determination could be done efficiently even without assuming separability. In fact, the techniques for sharing computation that we develop in this paper can be applied to the winner determination algorithm proposed in [10]. We return to this issue in Section V.

For example, consider the click-through rates defined by Figure 1. Now suppose the advertisers bids are as depicted in Figure 3. Then winner determination assigns slot 1 to advertiser A and slot 2 to advertiser B.

B. Shared Winner Determination

Having described how winner determination works for a single auction, we turn our attention to sharing the work of winner determination between multiple auctions that occur in the same round. The choice of granularity of a round is left to the system designer. While choosing a coarser granularity will lead to higher sharing between auctions (since more searches will occur per round), and thus greater overall efficiency, it will also increase the latency (the time the user has to wait before obtaining her search results). Studies have shown that users tolerate median latencies of up to 2.2 seconds without much adverse perception of search quality, but median latencies of about 3.6 seconds or more are considered too long [8].

Under the separability assumption, winner determination amounts to finding the advertisers with the top k values of $b_i c_i$ where k is the number of slots and b_i is advertiser i's bid and c_i is the advertiser-specific factor of advertiser i's click-through rate. Thus, if the same set of advertisers take part in two auctions in the same round, then slots would be awarded in the same way in both. However, not every bidder takes part in every auction. Advertiser can specify a set of *bid phrases*. If the search query does not match one of the advertiser's bid phrases, then his ad is not entered into the auction. In addition, advertisers can specify a *daily budget*. If an advertiser has already spent his budget for the day, then again he will not take part in the auction.

In this section, we ignore budget constraints; we deal with that in Section IV. In determining whether a query matches an advertiser's bid phrase, we assume that the two-stage method proposed in [11] is used, where the search query is first mapped into a lower-dimensional space of bid phrases and is then matched to the advertisers' bid phrases using exact match. Accordingly, if a bid phrase does indeed match some query, then we must find the advertisers with the top k values of $b_i c_i$ whose set of bid phrase contain the bid phrase. This is where we can share work between the different auctions.

For example, suppose that the search queries "hiking boots" and "high-heels" occur in the same round. There might be several general shoe stores that specify both queries as bid phrases. However there might be a few sports stores that specify "hiking boots" but not "high-heels", while a few high-end fashion accessory stores might only be interested in "high-heels" queries. Suppose there are 200 general shoe stores, 40 sports stores, and 30 upscale fashion stores. Finding the top k advertisers for each of the two phrases separately requires us to scan through 240 and 230 advertisers respectively. However, if we find the top k advertisers among the general shoe stores, the top k among sports stores, and the top k among the fashion stores (which requires looking at 200, 30, and 40 advertisers), we can then merge the first and second top k lists to find the top k advertisers interested in "hiking boots", and the first and third top-k lists to find the top k advertisers interested in "high-heels". Merging in this way allows us to scan 40% fewer advertisers.

This suggests the use of merging of two top-k lists as a primitive aggregation operation that we employ to build shared plans that successively aggregate the $b_i c_i$ values of all the advertisers so as to find the aggregates corresponding to the sets of advertiser interested in each bid phrase while minimizing the number of aggregate operations performed. Thus, the plan we build will be a DAG where each leaf node represents an advertiser, and each internal node has in-degree 2 and represents a top-k aggregation operator that aggregates the top k advertisers from the two upstream nodes.

One further issue that complicates sharing is that not all bid phrases occur in a given round. Thus a single shared plan may not be optimal in all rounds. Unfortunately, coming up with a new plan on the fly at every round based is not practical given the latency requirement of winner determination. Instead, we try to find a single plan offline that works well 'on average'. To formalize this, we assume that the event that a bid phrase occurs in a round is an independent Bernoulli trial whose probability is known. We call the probability that bid phrase q occurs its *search rate* and denote it as sr_q. We then try to find the plan involving pairwise top-k aggregation that computes the aggregate for each bid phrase, and minimizes the expected number of nodes *materialized* per round. A node is materialized in a given round if it is used to compute the result for a bid phrase that occurs in that round. In other words, a node is materialized if there is a path in the plan's DAG from that node to some node corresponding to a bid phrase query node. Therefore the probability of node v being materialized is $1-\prod_{q:v \rightsquigarrow v_q}(1-sr_q)$ where $v \rightsquigarrow q$ represents the statement that node v is used in the computation of the aggregate query corresponding to bid phrase q in the shared plan. Thus, by linearity of expectation, the total expected cost of a plan is

$$\sum_v \left(1 - \prod_{q:v \rightsquigarrow q} (1 - sr_q)\right).$$

C. Shared Aggregation

In this section, we examine the problem at the core of sharing winner determination: optimizing shared top-k aggregation plans. To this end, we develop a framework for shared aggregation using an abstract aggregation operator specified by a set of algebraic properties that the operator satisfies. We show that finding an optimal shared plan for our abstraction of the top-k aggregation operator is not only NP-hard, but is in fact inapproximable. The construction used in the proof motivates our heuristic for finding a good shared plan in the next section.

We start out by defining our notion of an abstract aggregation operator and its associated aggregate queries. An abstract aggregate operator is simply a binary function $\oplus$: $Z \times Z \rightarrow Z$ for some set of values Z (e.g., $\mathbb{Z}$, $\mathbb{N}$). This is sometimes known as a magma. Given the abstract operator $\oplus$, aggregation queries are represented by $\oplus$-*expressions* which are obtained by starting out with a set of variables X and closing off under the binary $\oplus$ operator. An example of an aggregation query is $(x \oplus y) \oplus z$, where x, y, z are variables that take values in Z. In our setting, each variable represents the bid of some advertiser, and the values of the variables change rapidly since advertisers are constantly updating their bids using external search engine optimizers [12] or automated bidding programs [10] in order to achieve complex advertising goals such as staying in a given slot during specific hours of the day, staying a certain number of slots above a competitor, dividing one's budget across a set of keywords so as to maximize the return-on-investment, etc. [12], [13], [14]. We therefore have to evaluate our aggregate queries at each round since the variables are constantly taking on different values.

Without using information about the algebraic properties of $\oplus$, we can only share work between queries in a rather limited manner by reusing the results of sub-expressions used to compute the queries. For example, we can share work between $x \oplus y$ and $(x \oplus y) \oplus z$ by re-using the value of $x \oplus y$ (which was computed for the first query) during the computation of the second. But if we take advantage of the various algebraic properties that $\oplus$ satisfies, we can increase the amount of shared computation. For example, if $\oplus$ is commutative then we can share work between the queries $x \oplus y$ and $(y \oplus x) \oplus z$ by aggregating the value of z with the value of the first query in order to compute the value of the second.

Let I_q be the set of advertisers interested in bid phrase q. Then an $\oplus$-expression representing the aggregate query for bid phrase q is $\oplus_{i \in I_q}\ b_i$ – we are implicitly using right-associativity by convention, not that it really matters – where b_i is the variable containing advertiser i's bid. Throughout this subsection, we assume that all bid phrases occur in every round with probability 1 (i.e., $sr_q = 1$ for each bid phrase q). The hardness results presented here therefore extend to the case when the sr_qs are arbitrary. Sharing winner determination then amounts to finding a shared top-k aggregation plan that produces, for each phrase q, the top-k aggregate of the bids of advertisers listed in X_q. Recall that the top-k aggregation operator is the binary function that takes in two k-lists (i.e., lists of size at most k) and outputs a k-list of the top k elements of the union of the two input lists. Notice that this operator is clearly associative, commutative, and idempotent (i.e., aggregating a list with itself returns the list itself). It also has an identity element, namely, the empty list which, when aggregated with any k-list, returns that list. We therefore abstract the top-k aggregator using an abstract aggregator $\oplus$ satisfying the following algebraic properties.

A1)	$\forall a . \forall b . \forall c . a \oplus (b \oplus c) = (a \oplus b) \oplus c$	(associativity)
A2)	$\exists e . \forall a . a \oplus e = e \oplus a = a$	(identity)
A3)	$\forall a . a \oplus a = a$	(idempotence)
A4)	$\forall a . \forall b . a \oplus b = b \oplus a$	(commutativity)

For convenience of notation, let $\mathcal{A} = \{A1, A2, A3, A4\}$. $\mathcal{A}$ defines the algebraic notion of a semilattice with identity element, and so our results in this subsection apply to any meet or join operator, such as $\min$, $\max$, Bloom filter unions, etc. We also point out that presence or absence of the identity axiom A2 does not have any affect on our complexity results. This is mainly due to the fact that we are aggregating variables, not constant elements, and therefore we cannot exploit the properties of the identity element since the variables may or may not contain the identity element at any given round.

We say that two $\oplus$-expressions e and e' are $\mathcal{A}$-*equivalent* iff $e = e'$ is provable in first-order logic plus $\mathcal{A}$. Now we can formally define the notion of a shared plan. Given a set E consisting of $\oplus$-expressions over X, an $\mathcal{A}$-*plan for* E is a DAG satisfying the following properties:

1) each node is labeled with an $\oplus$-expression and has in-degree either 0 or 2,
2) each node with in-degree 0 is labeled with a variable $x \in X$,
3) each node with in-degree 2 is labeled with an $\oplus$-expression $e \oplus e'$,
4) each $e \in E$ is $\mathcal{A}$-equivalent to the label of some node.

The *total cost of an* $\mathcal{A}$*-plan* is the number of nodes with non-zero in-degree in the graph (i.e., those nodes representing top-k aggregation operators). Now we can formally state the shared aggregation problem as follows. *Given a set* E *of* $\oplus$*-expressions over* X*, find the min-cost plan for computing each* $e \in E$.

We assume, without loss of generality, that no two $\oplus$-expressions in E are $\mathcal{A}$-equivalent and also that no $\oplus$-expression in E is $\mathcal{A}$-equivalent to any variable $x \in X$ since we can identify such expressions upfront and remove such duplicates. We define the *base cost* of an $\mathcal{A}$-plan to be $|E|$. Since every expression E must be the label of a non-leaf node of a plan for E, every plan for E has cost at least $|E|$. What is interesting is the cost of the plan over and above $|E|$. We define the *extra cost* of an $\mathcal{A}$-plan to be the total cost of the plan minus $|E|$. The nodes contributing to extra cost are the partial results that are used to compute the final set of aggregates. Note that minimizing the extra cost is equivalent to minimizing the total cost of an $\mathcal{A}$-plan. Later on, when we

discuss inapproximability, we will measure competitive ratio in terms of extra cost instead of total cost, since the base cost for all $\mathcal{A}$-plans for E is the same and is unavoidable.

First we state the following lemma that is easy to prove.

Lemma 1: Two $\oplus$-expressions over a set of variables X are $\mathcal{A}$-equivalent iff the set of variables appearing in the two expressions are equal. In particular, $e_1 \oplus e_2$ and e are $\mathcal{A}$-equivalent iff the set of variables appearing in e is equal to the union of the sets of variables appearing in e_1 and e_2.

Next, we show that finding an optimal shared plan is NP-hard for our abstraction of the top-k aggregator.

Theorem 2: Finding a min-cost $\mathcal{A}$-plan for E is NP-hard, where E is a finite set of $\oplus$-expressions over a finite set of variables X.

Proof: We reduce this to the *set-cover problem*, which is well-known to be NP-complete [15]. Recall that, in the set-cover problem, we are given a finite 'universal' set U, a finite collection $\mathcal{S}$ of subsets of U such that $\cup_{S \in \mathcal{S}} S = U$, and an integer k, and we must determine whether there is some $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \leq k$, and such that $\cup_{S \in \mathcal{S}'} S = U$.

Consider any instance of set cover. We can convert this into an instance of the problem of finding a minimum-cost $\mathcal{A}$-plan using the following construction. We create a variable for each element of the universal set. That is, we set $X = U$. For each $S \subseteq X$, we define a 'canonical' $\oplus$-expression e_S as follows. Let $<_X$ be an arbitrary strict ordering on the variables in X. If $S = \{x_1, \ldots, x_k\}$ is a nonempty set of variables in X, where $x_1 <_X \cdots <_X x_k$, then $e_S = x_1 \oplus \cdots \oplus x_k$. (Since $\oplus$ is associative by assumption, we can omit parentheses here.)

Now let the set of $\oplus$-expressions $E = \{e_U\} \cup \{e_S : S \in \mathcal{S}\}$. That is, we have an $\oplus$-expression corresponding to each set in $\mathcal{S}$ and one extra $\oplus$-expression corresponding to the universal set. Note that this construction can be done in polynomial time.

Now suppose that we have a polynomial-time algorithm for finding the min-cost $\mathcal{A}$-plan for E. Let G be the (DAG) plan returned by the algorithm. Let $\leq$ be the binary relation on nodes in G defined by $u < v$ iff G contains a directed (possibly zero-length) path from u to v. For each $e \in E$, let u_e be the node labeled with the $\oplus$-expression that is $\mathcal{A}$-equivalent to e. Note that checking whether two $\oplus$-expressions are $\mathcal{A}$-equivalent can be done in polynomial time by Lemma 1. Let $V = \{u : u \leq u_{e_U}\}$ and $W = \{u : \exists S \in \mathcal{S} \,.\, u \leq u_{e_S}\}$. That is, V is the set of all nodes that have a path to the node labeled e_U and W is the set of all nodes that have a path to a node labeled e_S for some $S \in \mathcal{S}$. So V induces an arborescence rooted at u_{e_U} that represents the plan's pairwise aggregation computation of e_U. Similarly the DAG induced by V represents the plan's computation of the $\oplus$-expressions in $\{e_S : S \in \mathcal{S}\}$. Let Z be the set of nodes in $V \cap W$ that have an edge into $W \setminus V$. The nodes in Z are the ones with paths leading both to u_{e_U} and u_{e_S} for some $S \in \mathcal{S}$. For each node $z \in z$, let S_z be the set in $\mathcal{S}$ such that $z = u_{e_{S_z}}$. Note that Z forms a cut of the arborescence induced by V since V and W have the same leaf nodes (namely, the nodes labeled by the variables in X). Therefore, by Lemma 1, $\{S_z : z \in Z\}$ is a set cover of U since e_U is formed by aggregating the nodes in Z since Z cuts the arborescence induced by V. Since the plan generated by the algorithm was minimal, this must be a minimal set cover of U, otherwise we could have replaced the nodes in $V \setminus W$ by aggregating the smaller set cover, which would have produced a plan with fewer nodes. ■

Finally, we extend the idea behind the construction in the previous proof to show that finding an optimal shared plan for our abstraction of the top-k aggregator is, in fact, hard to approximate to within less than a logarithmic factor of optimal.

Theorem 3: There is no polynomial-time algorithm that finds a shared plan whose extra cost is within a $\log n$ factor of optimum unless $P = NP$.

Proof: We follow the same construction as the proof of Theorem 2, except that we close the query expressions off under sub-expressions before adding the universal set query. This ensures that the only extra nodes we add are for computing the universal set query, which as we showed in the proof of Theorem 2 corresponds to finding a minimal set cover. Then the theorem follows directly from the fact that minimal set cover is not approximable to within a $\log n$ factor of optimal [16]. ■

As we mentioned previously, these complexity results apply to the case where the queries are probabilistic as well.

D. Algorithm

In the previous section we proved that finding an optimal shared plan for top-k aggregation between multiple auctions is inapproximable to within a $\log n$ factor even for the special case where all queries occur with certain probability. We now propose a heuristic for finding a shared aggregation plan for multiple probabilistic queries. Our approach consists of two stages: identifying fragments, and aggregating across fragments.

1) Identifying fragments: In the first stage, we group together all variables that occur in the same set of query expressions. We associate with each variable a bit string of length m, where the ith bit indicates whether or not the variable occurs in the i query expression. Then we group together all variables that are associated with the same bit string. These groups are equivalence classes of variables and are called *fragments* in [17]. Note that even though there are 2^m possible fragments, only $O(n)$ will be non-empty since there are n variables. We can safely aggregate elements within a fragment since no sharing occurs across fragments, since fragments are equivalence classes. This step itself provides some basic multiquery optimization since no fragment is computed twice. It is not hard to see that this step takes $O(mn \log n)$ time; the $\log n$ factor comes from having to index the fragments by bit string to identify groups of fragments. Alternatively, a hash table of bit strings could be used for grouping.

2) Aggregating across fragments: In the second stage, we use a greedy heuristic to complete the plan that was started out by aggregating together all the nodes within each fragments. We say that an $\mathcal{A}$-plan is *incomplete* if it does not compute all query expressions, i.e., if there is some query expression that

is not equivalent to the label of any node in the plan. We can always complete an incomplete plan by finding a set cover of the missing query nodes from the collection of existing nodes. Note that we are associating nodes with the set of variables mentioned in the $\oplus$-expression labeling the node according to Theorem 1. Also note that we use the term 'set cover' to mean a cover whose union exactly equals the target set instead of just being a superset of the target set. This usage of the term is made without loss of generality with respect to earlier complexity results.

Suppose for the moment, that all queries occur with probability 1 at each round. Then the optimal way to complete the plan without any further sharing would be to find a minimum set cover C of each of the missing queries and to aggregate together all the nodes in C using an arbitrary binary tree, using $|C| - 1$ nodes. Thus, the cost of completing the plan without further sharing is $\sum_q (|C_q| - 1)$, where C_q is the size of the minimal set cover for query node q. This motivates our greedy heuristic, which works as follows.

At every step, we find two nodes that would aggregate together to form a new node that would lead to the greatest decrease in $\sum_q |C_q|$ per unit extra cost of computing the new aggregate node. We call the decrease in $\sum_q |C_q|$ resulting from aggregating a pair of existing nodes the *coverage gain* of the pair. Note that the extra cost of creating a new aggregate node is 1 unless the aggregate is equivalent to a query expression, in which case the extra cost is 0 since the query node would have had to be computed anyway and would therefore have counted toward the base cost. If there are multiple pairs of nodes that would cover some previously uncovered query, then we pick the pair with the highest coverage gain. The intuition is that the faster we cover all the query nodes, the faster the plan gets completed, and hence the fewer the extra nodes that are required.

Unfortunately, as the reader might have noticed, we run into a problem if we try to carry out the heuristic as stated above. The issue is that in order to pick the pair of nodes with the highest coverage gain, we need to first calculate the minimum cover for each query node from the existing set of nodes. But minimum set cover is an NP-hard problem, and is in fact not approximable to within a $\log n + 1$ factor as we saw in Section II-C. Therefore we cannot use the real set cover in measuring coverage gain, so instead we use the cover prescribed by the greedy covering algorithm which works as follows. Until the target set is covered, repeatedly pick the feasible set that covers the maximum number of as-yet-uncovered elements. It is known that this greedy algorithm produces a cover within a $1 + \log n$ factor of optimal [18]. In fact, the greedy algorithm performs even better during the early decisions: the greedy algorithm achieves a competitive ratio logarithmic in $|S|$, where S is the largest cardinality set in the collection [18] (in our context, this is the size of the largest set associated with an existing node in the incomplete plan). We call the total size of the covers of all the query nodes as prescribed by the greedy covering algorithm the *greedy coverage*. Since we are always decreasing the minimum cover at each step, we run for at most $\sum_q |X_q|$ steps, where X_q is the set of variables mentioned in query q. Each step requires checking the newly created node with all existing nodes to form a pair that maximizes greedy coverage gain. So the total running time is polynomial in $\sum_q |X_q|$.

So far in this subsection, we have assumed that all queries appears at each round. However, as we described earlier, in our application, each query occurs independently with some probability. We therefore extend the algorithm to deal with this probabilistic setting by replacing the notion of coverage gain with *expected greedy coverage gain*. Expected greedy coverage gain of a pair of nodes is the decrease in expected total greedy coverage of queries (i.e., $\sum_q sr_q |C_q|$, where sr_q is the probability of the aggregate query q occurring) resulting from aggregating that pair of nodes. Thus, the algorithm favors the covering and sharing of the queries that are more probable over rare queries.

To summarize, our final algorithm works as follows:

1) First, group variables by the set of query expressions they appear in, and then aggregate the variables within each group.
2) Until the plan covers all query nodes, do the following:
   a) For each pair of nodes in the incomplete plan, compute the expected greedy coverage.
   b) If there exist some pairs of nodes that could be aggregated together to form a missing query node, then aggregate one such pair with the maximum expected greedy coverage.
   c) Otherwise pick any pair with maximum expected greedy coverage and aggregate them to form a new node.

The running time of the algorithm is polynomial in $\frac{\max_q sr_q}{\min_q sr_q} \sum_q |X_q|$ using an analysis similar to that for the deterministic queries case presented above. We observe that our algorithm performs within a constant factor of any polynomial-time algorithm (unless $P = NP$) in the inapproximable case described in the proof of Theorem 3. Initially, our algorithm adds aggregates that compute all the $\oplus$-expressions of E' since these aggregate nodes have zero extra cost. Once these nodes have been created, our algorithm then tries to find a greedy covering for the query node labeled e_U and therefore essentially runs greedy set cover, which is a $(1 + \log n)$-approximation to optimal.

We point out that the more certain the queries are, the more effective our sharing techniques will be. The intuition is that nodes that perform shared computation give more savings when they are used in more queries. Figure 4 shows an example of the savings provided in a set of 10 top-k queries over 20 advertisers. The queries were chosen by flipping coins to determine whether each advertiser would be in the list of top-k contenders, discarding duplicate queries.

III. Shared Sorting and Winner Determination

In the previous section, we provided techniques for sharing between auctions with where the advertiser-specific factor of

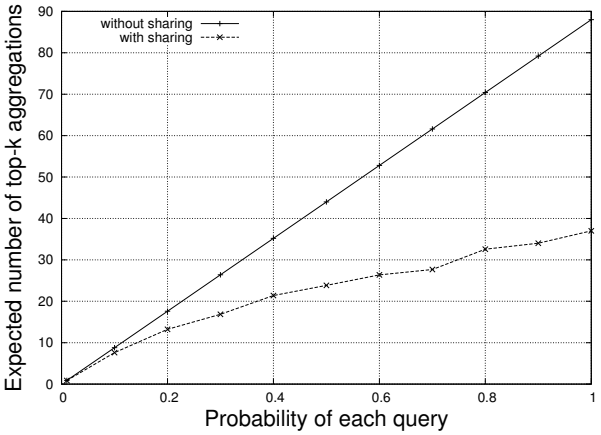

Fig. 4. Expected cost of plan vs. query probability

the click-through rate was identical across all the bid phrases. In reality, it seems quite reasonable that the same advertiser in the same slot position might have different click-through rates for substantially different bid phrases. For example, a book store that mainly sells books but also dabbles in selling movies and music might have a higher advertiser-specific factor when the bid phrase is "books" than when the bid phrase is "DVDs". If this is the case, we cannot directly share the top-k aggregation of the $b_i c_i$ values across bid phrases as we did in Section II-B since the value of c_i can be different for each phrase. Instead, we devote this section to examining how we can share work by exploiting the fact that the b_i values are shared across bid phrases.[2]

A. Threshold Algorithm

In order to share work between auctions, we use the well-known *threshold algorithm* [19] to find the top q advertisers for each bid phrase that occurs in a given round. In our context, the threshold algorithm works as follows. For a given bid phrase q, let c_i^q be the advertiser-specific factor of the click-through rate of advertiser i for bid phrase q. Let $j_1^q, j_2^q, \ldots$ be an ordering of advertisers who are interested in bid phrase q, ordered by descending values of c_i^q.[3] Let $i_1^q, i_2^q, \ldots$ be an ordering of advertisers who are interested in bid phrase q, ordered by descending values of b_i. The algorithm proceeds in stages. At each stage $s = 1, 2, \ldots$, the threshold algorithm incrementally maintains k indices in $\{i_{s'}^q : 1 \le s' \le s\} \cup \{j_{s'}^q : 1 \le s' \le s\}$ with the highest values of $b_i c_i^q$. It terminates early at the first stage s where all top k values are no less than the threshold defined by $b_{i_s^q} c_{j_s^q}$. It is well-known that the threshold algorithm is *instance optimal* for the class of algorithms that find the advertisers with the top k values of $b_i c_i^q$ without making "wild guesses" (i.e., the algorithms must not access an advertiser until that advertiser is encountered via a sequential scan of one of the lists). Instance optimality means that, for any input, the threshold algorithm finds the top k values within a constant factor of the time it takes the fastest algorithm that avoids wild guess on that input. Thus, we could solve the top-k problem for each bid phrase in an instance-optimal manner if we had a way of listing, on demand, the advertisers interested in phrase q, starting with the advertiser i with the highest b_i and proceeding in decreasing order of b_i values. This motivates the following problem that we call *shared merge-sort*.

B. Shared Sorting

Consider a sponsored search auction that matches some bid phrase q, and let I_q be the set of advertisers who are interested in k. For ease of exposition, we assume that each $|I_q|$ is a power of two; the discussion generalizes to arbitrary cardinalities in a straightforward way. The threshold algorithm as described above initially asks for the advertiser $i \in I_q$ with the highest value of b_i. It then asks for the advertiser from I_q with the next highest value of b_i, and then the next, and so on, until the threshold condition described above has been met. To supply the threshold algorithm with an advertiser at each stage, we construct a plan whose DAG is a balanced binary tree as used in a merge-sort of the set $\{b_i : i \in I_q\}$. Each leaf node is associated with a distinct b_i from this set. Rather than running the entire merge-sort upfront, we treat each non-leaf node as an on-demand operator that stores a left register and a right register, and sends the contents of the larger of the two registers upstream and then clears that register; if a register to be read from is empty, its value is pulled from its corresponding downstream node. This way, we don't do any extra work beyond the stage where the threshold condition is met. Each operator stores the sequence of values it has sent upstream. This will be used for caching results when operators are shared between multiple sort plans.

Now suppose that there is some other auction for another phrase q', and let $I_{q'}$ be the set of advertisers who are interested in q'. If $I_q \cap I_{q'} \neq \emptyset$, then we have already done some work in ordering advertisers who are interested in k. We would like to re-use some of this work when feeding advertisers to the threshold algorithm for phrase q'. Clearly, we can re-use the cached results of any operators below which all leaves correspond to advertiser in $I_q \cap I_{q'}$. This amounts to the problem of optimally sharing our on-demand merge operators in the merge-sort trees for multiple bid phrases. With each each merge operator v, we associate the set of advertisers I_v corresponding to the leaves below the operator. Then, according to the usual merge-sort tree restrictions, two operators u and v can be merged into a new merge operator w only if $I_u \cap I_v = \emptyset$ and $|I_u| = |I_v|$. The total number of times an operator v is invoked in the worst case is $|I_v|$. This happens when the threshold condition is never met and the entire set I_v is sorted. Since we do not model the distribution of values that the b_is take, we conservatively use this *full-sort cost* when evaluating the cost of shared plans. Thus, the expected full-sort cost of a merge-sort operator in a shared plan is $|I_v| \left(1 - \prod_{q: v \rightsquigarrow q} (1 - sr_q)\right)$, where sr_q is the probability

[2] If, in addition, we allow the values of b_i to vary across bid phrases, then there is no opportunity for sharing work between phrases at all since no data is shared between the phrases.

[3] We assume that the click-through rates are recalculated only occasionally and for the most part remain fixed. Therefore the ordering $j_1^q, j_2^q, \ldots$ can be treated as fixed and can be precomputed.

that bid phrase q appears in some auction, and $q \rightsquigarrow v$ denotes the property that operator v is used in bid phrase q's merge-sort tree in the shared plan. By linearity of expectation, the total expected full-sort cost of a shared merge-sort plan is

$$\sum_v |I_v| \left(1 - \prod_{q:v \rightsquigarrow q} (1 - sr_q)\right)$$

C. Algorithm

We propose the following simple bottom-up greedy heuristic for building a shared merge-sort plan that starts out with the leaf nodes, each corresponding to a distinct advertiser, and successively merges the two nodes that would lead to the largest savings in expected cost. When creating a new node w, we annotate it with the set of bid phrases Q_w whose merge-sort tree it contributes to. Initially, each leaf nodes v is annotated with $Q_v = \{q : v \in I_q\}$. At any point, we can merge nodes u and v into a new node w only if $Q_u \cap Q_v \neq \emptyset$, $I_u \cap I_v = \emptyset$, and $|I_u| = |I_v|$. We then set $Q_w = Q_u \cap Q_v$ and $I_w = I_u \cup I_v$. We pick the u and v such that the expected savings of merging them to create new node w is maximized. The expected savings from creating node w is given by

$$|I_w| * \sum_{i=1}^{n} \left[\left(\prod_{1 \leq j < i} (1 - sr_{q_j}) \right) sr_{q_i} \left(\sum_{j=i+1}^{n} sr_{q_j} \right) \right]$$

where $Q_w = \{q_1, \ldots, q_n\}$. Note that $\sum_{i=1}^{n} \left[\left(\prod_{1 \leq j < i} (1 - sr_{q_j}) \right) sr_{q_i} \left(\sum_{j=i+1}^{n} sr_{q_j} \right) \right]$ is simply the expected number of queries in Q_w that occur beyond the first.

IV. Dealing with Budget Uncertainty

In most existing systems, advertisers can specify a daily budget, which represents the maximum amount of money the advertiser is willing to spend per day. The search provider is required to respect this constraint, and must therefore never charge an advertiser more than his daily budget on any given day. In order to perform winner determination correctly, we need to take this budget into consideration. What makes this tricky is that the amount of budget remaining is often uncertain. With the high rate of searches, an advertiser may well be interested in a new auction before he has to pay for his winnings from a previous auction. Since advertisers pay for clicks only after a user clicks on their ad, if the user from the first auction has not yet clicked on the advertiser's ad by the time the second auction occurs, there will be uncertainty about the amount of budget that the advertiser has remaining, since the first user may still click on the ad at some time in the future.

Suppose we were to ignore the budget issue during winner determination and simply not charge the advertiser if the user clicks after the advertiser's budget has been depleted. Consider an advertiser who is interested in a popular keyword, such as music, whose budget is almost exhausted. Until he receives enough clicks to completely exhaust his budget, we would allow him to bid his remaining budget on every music-related search query that occurs. He may win m auctions, but only have enough money in his budget to pay for $m' < m$ clicks. If he gets more than m' clicks, payment for the extra clicks would be forgiven. Thus, the advertiser would get more than his budget's worth of clicks. This constitutes lost revenue, since the slots could have been assigned to competing advertisers who had less chance of depleting their budgets. We now propose a principled solution to this problem by taking into account the outstanding ads that are awaiting clicks, and computing appropriately throttled bids for advertisers who are likely to go over budget.

A. Throttling Bids

To start with, consider an advertiser i for whom there are no outstanding ads awaiting clicks from users. Denote the i's remaining budget, i.e., his daily budget minus the amount he has paid to the search provider for clicks that have already occurred, as β_i. Suppose that in the current round, the advertiser takes part in m_i auctions, and that his current bid for a click is b_i. Rather than using b_i directly as his bid, we use a modified bid $\hat{b}_i$ instead. If the advertiser can afford to pay his stated bid of b_i for each of the m_i auctions, then we take $\hat{b}_i$ to be b_i; otherwise, we use the highest possible bid that the advertiser could still afford to pay for each auction. In other words, we let $\hat{b}_i = \min(b_i, \beta_i/m_i)$.

Now suppose that there are some, say l_i, outstanding ads of advertiser i that are awaiting clicks. For each outstanding ad j, suppose the price for a click on that ad was determined to be π_j and the probability of that ad getting clicked (given the time elapsed since the ad was displayed) is ctr_j. We make no assumptions about the value of ctr_j, but we point out that it is reasonable to model ctr_j as decreasing over time, and furthermore, that it reaches 0 after a specified time limit has passed; this will enable us to discard outstanding ads that have received no clicks in a long time. Let X_j be the random variable for the amount eventually paid for ad j. For any $l \in \{1, \ldots, l_i\}$, let $S_l = \sum_{j=1}^{l} X_j$. Thus, the amount of budget remaining once the debts for outstanding ads have been cleared is $\max(0, \beta_i - S_{l_i})$. We would like to take $\hat{b}_i$ to be the highest possible bid that the advertiser could still afford once his debts for outstanding ads have been cleared. That is,

$$\hat{b}_i = \begin{cases} b_i & \text{if } S_{l_i} < \beta_i - m_i b_i \\ 0 & \text{if } S_{l_i} \geq \beta_i \\ (\beta_i - S_{l_i})/m_i & \text{otherwise} \end{cases}$$

or, written another way, $\hat{b}_i = \min(b_i, \max(0, \beta_i - S_{l_i})/m_i)$. However, since the values of the X_js are uncertain because the ads are still awaiting clicks, we use the expected value at the time of winner determination. That is, we let $\hat{b}_i = \mathrm{E}(\min(b_i, \max(0, \beta_i - S_{l_i})/m_i))$.

B. Computing Bounds for Throttled Bids

Let ω_l denote $\sum_{j=1}^{l} \pi_j$, where π_j is the price for a click on the jth outstanding ad. Note that $S_{l_i} \leq \omega_{l_i}$, since each X_j is either π_j with probability ctr_j, or else is 0 with the remaining

probability. Thus, if $\omega_{l_i} \leq \beta_i - m_i b_i$, then $\hat{b}_i = b_i$. Otherwise, if $\omega_{l_i} > \beta_i - m_i b_i$, we can compute $\hat{b}_i$ as follows. Note that $\mathrm{E}(\min(b_i, \max(0, \beta_i - S_{l_i})/m_i)) = \mathrm{E}(\min(m_i b_i, \beta_i - \min(\beta_i, S_{l_i})))/m_i$. Thus, in order to compute $\hat{b}_i$, we can compute the distribution of $\min(\beta_i, S_{l_i})$ and then take the expected value of $\min(m_i b_i, \beta_i - \min(\beta_i, S_{l_i}))$ over that distribution. This takes time $O(\min(2^{l_i}, \beta_i))$, assuming that β_i is written in the lowest denomination of currency. However, observe that during the winner-determination phase, we do not need the precise values of $\hat{b}_i$. We simply need the ability to compare $\hat{b}_i$ with $\hat{b}_{i'}$ for advertisers i and i' in order to find the top k advertisers. Of course, once winner determination is over, we will need the precise values of $\hat{b}_i$ for the winning advertisers in order to compute the prices for clicks. But there are only k winning advertisers at this point, so the amount of computation is a lot less than computing the precise $\hat{b}_i$ values for all n advertisers.

Now, in order to compare the $\hat{b}_i$ and $\hat{b}_{i'}$, we use Hoeffding bounds to compute successively tighter upper and lower bounds for $\hat{b}_i$ and $\hat{b}_{i'}$ until the upper bound is lower than the lower bound for the other at which point we can resolve the comparison test with certainty. In order to do this, notice that $\hat{b}_i$ can be rewritten as

$$b_i \Pr(S_{l_i} < \beta_i - m_i b_i) + \frac{1}{m_i} \mathrm{E}((\beta_i - S_{l_i}) 1_{\beta_i - m_i b_i \leq S_{l_i} < \beta_i})$$

We will denote upper and lower probability bounds as $\overline{\Pr}(\ldots)$ and $\underline{\Pr}(\ldots)$ respectively, and we denote upper and lower expectation bounds as $\overline{\mathrm{E}}(\ldots)$ and $\underline{\mathrm{E}}(\ldots)$ respectively. Let μ_l denote $\mathrm{E}(S_l) = \sum_{j=1}^{l} ctr_j \pi_j$ by linearity of expectation, and let σ_l denote $\sqrt{\mathrm{Var}(S_l)} = \sqrt{\sum_{j=1}^{l} ctr_j (1 - ctr_j) \pi_j^2}$. Using Hoeffding's inequality [20], which upper-bounds the probability that the sum of bounded independent random deviates from its expected value, we can derive the following bounds for $\Pr(S_l < x)$ for any $x > 0$,

$$\underline{\Pr}(S_l < x) = \begin{cases} 1 \text{ if } \omega_{l_i} \leq x, \\ 0 \text{ if } x < \mu_{l_i} \leq \omega_{l_i}, \text{ and} \\ \max(0.5, 1 - \exp(-2(x - \mu_{l_i})^2 / \sum_{j=1}^{l_i} \pi_j^2)) \\ \qquad \text{if } \mu_{l_i} \leq x < \omega_{l_i}; \end{cases}$$

and

$$\overline{\Pr}(S_l < x) = \begin{cases} 1 \text{ if } \mu_{l_i} \leq x \\ \min(0.5, \exp(-2(\mu_{l_i} - x)^2 / \sum_{j=1}^{l_i} \pi_j^2)) \\ \qquad \text{if } x < \mu_{l_i} \leq \omega_{l_i}. \end{cases}$$

Using these bounds, we can derive bounds for $\Pr(x \leq S_l < y)$ as $\underline{\Pr}(x \leq S_l < y) = \max(0, \min(1, \underline{\Pr}(S_l < y) - \overline{\Pr}(S_l < x)))$ and $\overline{\Pr}(x \leq S_l < y) = \max(0, \min(1, \overline{\Pr}(S_l < y) - \underline{\Pr}(S_l < x)))$. Now for $0 < x < y$, we can bound $\mathrm{E}(S_l 1_{x \leq S_l < y})$ from above and below by $x \Pr(x \leq S_l < y)$ and $y \Pr(x \leq S_l < y)$ respectively. Using the bounds that we have just derived, we can bound the value of $b_i \Pr(S_{l_i} < \beta_i - m_i b_i) + \frac{\beta_i}{m_i} \Pr(\beta_i - m_i b_i \leq S_{l_i} < \beta_i) + \frac{1}{m_i} \mathrm{E}(S_{l_i} 1_{\beta_i - m_i b_i \leq S_{l_i} < \beta_i})$ and hence that of $\hat{b}_i$.

If the bounds for $\hat{b}_i$ and $\hat{b}_{i'}$ as computed above are insufficient to decide the comparison, we can expand $\Pr(S_l < x)$ and $\mathrm{E}(S_l 1_{x \leq S_l < y})$ in terms of expressions involving S_{l-1}, π_l, and ctr_l to get tighter bounds. We do this repeatedly until the bounds are tight enough to decide the comparison. $\Pr(S_l < x)$ expands to

$$ctr_l \Pr(S_{l-1} < x - \pi_l) + (1 - ctr_l) \Pr(S_{l-1} < x)$$

and $\mathrm{E}(S_l 1_{x \leq S_l < y})$ expands to

$$\begin{array}{rl} & ctr_l \mathrm{E}(S_{l-1} 1_{x - \pi_l \leq S_{l-1} < y - \pi_l}) \\ + & ctr_l \pi_l \Pr(x - \pi_l \leq S_{l-1} < y - \pi_l)) \\ + & (1 - ctr_l) \mathrm{E}(x \leq S_{l-1} < y) \end{array}$$

We order the random variables X_j in increasing order of π_j. We expand out variables of high π_j values first, thus quickly eliminating their appearance in the Hoeffding bounds which as can be seen from the equations above leads to tighter bounds. Note that, in the worst case, the running time for getting a precise value for $\hat{b}_i$ is still $O(\max(2^{l_i}, \beta_i))$, but our technique allows us to terminate early once the bounds are tight enough for the purpose of comparison. Furthermore, we can cache the bounds for comparison with other $\hat{b}_{i'}$s and for computing the precise computation of $\hat{b}_i$ should advertiser i be one of the top k advertisers.

V. Related Work

Early work on multiquery optimization includes work done by Sellis [21] to provide shared plans for select, project, and joint queries. However, this work did not consider shared aggregation. Cocke looked at sharing work for computing expressions where the operators were commutative non-associative operators in the context of compiler optimization by finding global common subexpressions [22]. In contrast, our work focuses on operators that are associative as well as commutative.

There has recently been a lot of work done in the context of data streaming and sensor networks that is closely related to ours. For example, Dobra, Garofalakis, et al. introduce a technique for computing approximate aggregates by sharing work across multiple queries [23] transmitting 'sketches' of the data rather than the entire data. In our setting, it is important to find the exact aggregate values in order to ensure the desired economic properties of the auction, such as truthfulness and envy-freeness. Trigoni, Yao, et al. look at optimizing aggregates in sensor networks [24]. They focus on communication cost and use more coarse cost-model than ours. They consider a unit cost of sending a vector of aggregates whose length depend on the problem size. In contrast, we consider the cost of computing each individual aggregate, which is a more accurate computation cost model. Zhang, Koudas, et al. consider the problem of sharing aggregation over data streams in the Gigascope system where the queries are aggregates of group-bys of several attributes [25]. They use 'phantom' group-by aggregates that contain partial results for multiple queries and propose a greedy heuristic finding the optimal set of phantoms for count and sum aggregates. Krishnamurthy, Wu, and Franklin suggest the use of fragments [17] (i.e., grouping inputs by the set of selection predicates that they satisfy) that we use in the first stage of our algorithm.

However they did not take advantage of further algebraic properties of the aggregation operator as we do. Huebsch, Garofalakis, et al. consider the problem of sharing aggregate computation for distributed queries and classify aggregates based on whether or not they are linear and whether or not they are duplicate-insensitive [26]. Like [24], this work uses a coarser cost model than ours. Other work on multiquery aggregation includes that of Silberstein and Yang where they look at aggregation using a set of multicast trees in a network that satisfies certain assumptions on the relationship between the trees [27]. Our setting is different in that, rather than given a network, we have to design the optimal network between sources (inputs) and sinks (queries).

Non-Separable Click-Through Rates. Recent work goes beyond the traditional assumption of separable click-through rates [10]. Advertisers are allowed to bid on clicks, impressions, and purchases resulting from displaying their ad, and click-through and purchase rates are allowed to be non-separable. The technique proposed in [10] takes advantage of the fact that the number of slots is usually very small in comparison with the number of bidders. A complete bipartite graph is constructed with advertisers on one side and slots on the other. The edge between an advertiser and a slot is weighted by the expected realized bid that would obtained by assigning that advertiser to that slot. The graph is then pruned to a much smaller graph by considering only the advertisers with the k highest edges incident to each slot, where k is the number of slots. Then the maximum weight bipartite matching is found between these $O(k^2)$ advertiser and k slots using the well-known Hungarian algorithm [28]. Our work fits very well into this framework – we can use the shared top-k algorithms presented in this paper to find the top k advertisers for each slot in the graph-pruning step described above.

Finally, related to our work on uncertain budgets, Aggarwal and Hartline propose a related auction known as the knapsack auction, where bidders want to place items of varying sizes in a knapsack of a given capacity [29]. They suggest that this auction can be used to run a single auction to sell advertisement slots for the entire day where each advertiser's budget runs out after receiving exactly one click. In contrast to our approach, their auction fixes the outcome ex ante at the start of the day. Re, Suciu, et al. propose a technique they term 'multisimulation' to find the top-k most probable tuples in the result of a query to probabilistic database [30]. They do this by running Monte-Carlo simulations for all tuples and scheduling the simulations so as to quickly eliminate unlikely contenders.

VI. Conclusion

In this paper, we highlight the opportunity for sharing work when there is a high search volume by sharing the winner determination computation across multiple sponsored search auctions that occur simultaneously. We provided techniques for both separable and non-separable click-through rates even if the advertiser-specific factor is different across bid phrases in the case of separable click-through rates. We are working on a thorough experimental evaluation of our proposed heuristics. It would also be desirable to provide provable approximation bounds if possible.

We also demonstrate a way for advertisers to game the system when there is a high volume of search if the system ignores the issue of budget uncertainty. Our solution automatically throttles bids by takes into account ads that have been displayed recently but have not yet been clicked on. We showed how to solve winner determination efficiently by computing upper and lower bounds on throttled bids rather than computing their values precisely. As future work, we would like to explore in more detail how to schedule the refinement of these bounds to reduce the amount of work necessary to compare two throttled bids.

VII. Ongoing and Future Work

[10] propose a framework where advertisers submit bidding programs to bid on their behalf. These programs are run every time an auction occurs thus giving the advertisers much more dynamic control over their bids. In order to make informed decisions about how to bid, it would be useful for these programs to be able to compute quantities such as average (or maximum) bid placed on a given set of bid phrases (e.g., those bid phrases containing the word 'music'), or the total number of users who have searched for one of a set of bid phrases. These quantities can be computed using sum, average, and count aggregates over bid phrases. Often multiple advertisers will want to perform similar aggregates over similar sets of bid phrases, giving us the opportunity to share such aggregation. We therefore consider aggregates other than the top-k aggregate that we considered in Section II. However, rather than considering the shared aggregation problem for each particular aggregate, we take a more general approach and employ the abstract algebraic framework that we introduced earlier. As ongoing work, we study the relationship between algebraic properties of the aggregation operation in question and the complexity of finding the optimal shared aggregation plan. To this end, we consider the following algebraic properties of binary aggregation operator $\oplus$.

A1) $\forall a . \forall b . \forall c . a \oplus (b \oplus c) = (a \oplus b) \oplus c$ (associativity)
A2) $\exists e . \forall a . a \oplus e = e \oplus a = a$ (identity)
A3) $\forall a . a \oplus a = a$ (idempotence)
A4) $\forall a . \forall b . a \oplus b = b \oplus a$ (commutativity)
A5) $\forall a . \forall b . \exists! c . \exists! d . a \oplus c = d \oplus a = b$ (divisibility)

These axioms can been used to characterize various algebraic structures of interest, including semigroups ($\{A1\}$), monoids ($\{A1, A2\}$), groups ($\{A1, A2, A5\}$), Abelian groups ($\{A1, A2, A4, A5\}$), bands ($\{A1, A3\}$), semilattices ($\{A1, A3, A4\}$), quasigroups ($\{A5\}$), and loops ($\{A2, A5\}$). We have already seen axioms A1, A2, A3, and A4 from Section II. We focus mainly on those aggregates satisfying $A4$, since the most common and important aggregation operators that come up in our setting, and in database and stream settings in general are commutative. Such aggregates include sum, count, product, max, min, top-k, and Bloom-filter unions and intersections. Moreover, these aggregates can be combined

A1	A2	A3	A4	A5	Complexity
N	*	*	*	N	PTIME
N	N	N	*	Y	PTIME
N	Y	N	*	Y	PTIME
N	N	Y	*	Y	PTIME
N	Y	Y	*	Y	$O(1)$
Y	*	N	Y	N	NP-complete
Y	*	N	Y	Y	NP-complete
Y	*	Y	Y	N	NP-complete
Y	*	Y	*	Y	$O(1)$

Fig. 5. Complexity Results for Optimally Sharing Aggregation

with each other to compute other useful aggregates such as mean and variance.

Figure 5 summarizes our complexity results so far. Note that this includes a complete characterization for commutative aggregates. We do not yet have complexity bounds for the cases corresponding to lines 6 through 8, when $A4 = N$ instead of Y. Also missing are approximation algorithms for the NP-complete cases.

ACKNOWLEDGMENT

The first author is supported in part by NSF under Grants IIS-0534064 and IIS-0534404. The second author is in part supported by NSF under Grants ITR-0325453 and IIS-0534064, and by AFOSR under Grant FA9550-05-1-0055. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors. We thank Mingsheng Hong for several useful discussions on shared aggregation and for pointers to literature on multiquery optimization.

REFERENCES

[1] B. G. Edelman, M. Ostrovsky, and M. Schwarz, "Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords," NBER Working Paper No. W11765, November 2005. [Online]. Available: http://ssrn.com/abstract=847037

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

[3] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *Journal of Finance*, vol. 16, pp. 8–37, 1961.

[4] G. Aggarwal, A. Goel, and R. Motwani, "Truthful auctions for pricing search keywords," in *EC '06: Proceedings of the 7th ACM Conference on Electronic Commerce*. New York, NY, USA: ACM Press, 2006, pp. 1–7.

[5] H. R. Varian, "Position auctions," UC Berkeley Working Paper, 2006.

[6] G. Aggarwal, S. Muthukrishnan, and J. Feldman, "Bidding to the top: VCG and equilibria of position-based auctions," in *WAOA '06: Proceedings of the 4th Workshop on Approximation and Online Algorithms*. Berlin, Germany; Heidelberg, Germany: Springer, September 2006, pp. 15–28.

[7] SEO Tools, "SEO Book keyword tool," http://tools.seobook.com/keyword-tools/seobook, 2008.

[8] A. Sears, J. A. Jacko, and M. S. Borella, "Internet delay effects: how users perceive quality, organization, and ease of use of information," in *CHI '97: CHI '97 extended abstracts on Human Factors in Computing Systems*. New York, NY, USA: ACM Press, 1997, pp. 353–354.

[9] Nielsen/NetRatings, "Interactive advertising bureau (IAB) search branding study," Commissioned by the IAB Search Engine Committee, August 2004, available at http://www.iab.net/resources/iab_searchbrand.asp.

[10] D. J. Martin, J. Gehrke, and J. Y. Halpern, "Toward expressive and scalable sponsored search auctions." in *ICDE '08: Proceedings of the 24th IEEE International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 237–246.

[11] F. Radlinski, A. Broder, P. Ciccolo, E. Gabrilovich, V. Josifovski, and L. Riedel, "Optimizing relevance and revenue in ad search: A query substitution approach," in *SIGIR '08: Proceedings of the Conference on Research and Development in Information Retrieval*. New York, NY, USA: ACM Press, 2008, p. (to appear).

[12] B. Kitts and B. LeBlanc, "Optimal bidding on keyword auctions," *Electronic Markets*, vol. 14, no. 3, pp. 186–201, 2004.

[13] C. Borgs, J. Chayes, N. Immorlica, M. Mahdian, and A. Saberi, "Multi-unit auctions with budget-constrained bidders," in *EC '05: Proceedings of the 6th ACM Conference on Electronic Commerce*. New York, NY, USA: ACM Press, 2005, pp. 44–51.

[14] S. Muthukrishnan, M. Pál, and Z. Svitkina, "Stochastic models for budget optimization in search-based advertising," in *Internet and Network Economics*, ser. Lecture Notes in Computer Science. Berlin, Germany; Heidelberg, Germany: Springer, 2007, vol. 4858, pp. 131–142.

[15] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. New York, NY, USA: Plenum Press, 1972, pp. 85–103.

[16] C. Lund and M. Yannakakis, "On the hardness of approximating minimization problems," *Journal of the ACM*, vol. 41, no. 5, pp. 960–981, 1994.

[17] S. Krishnamurthy, C. Wu, and M. Franklin, "On-the-fly sharing for streamed aggregation," in *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM Press, 2006, pp. 623–634.

[18] D. S. Johnson, "Approximation algorithms for combinatorial problems," in *STOC '73: Proceedings of the 5th Annual ACM Symposium on Theory of Computing*. New York, NY, USA: ACM Press, 1973, pp. 38–49.

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

[20] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, no. 301, pp. 13–30, 1963.

[21] T. K. Sellis, "Multiple-query optimization," *ACM Trans. Database Syst.*, vol. 13, no. 1, pp. 23–52, 1988.

[22] J. Cocke, "Global common subexpression elimination," *SIGPLAN Notices*, vol. 5, no. 7, pp. 20–24, 1970.

[23] A. Dobra, M. N. Garofalakis, J. Gehrke, and R. Rastogi, "Sketch-based multi-query processing over data streams," in *EDBT '04: Proceedings of the 9th International Conference on Extending Database Technology*. Berlin, Germany; Heidelberg, Germany: Springer, 2004, pp. 551–568.

[24] N. Trigoni, Y. Yao, A. J. Demers, J. Gehrke, and R. Rajaraman, "Multi-query optimization for sensor networks," in *DCOSS '05: Proceedings of the 2005 International Conference on Distributed Computing in Sensor Systems*. Berlin, Germany; Heidelberg, Germany: Springer, 2005, pp. 307–321.

[25] R. Zhang, N. Koudas, B. C. Ooi, and D. Srivastava, "Multiple aggregations over data streams," in *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM Press, 2005, pp. 299–310.

[26] R. Huebsch, M. Garofalakis, J. M. Hellerstein, and I. Stoica, "Sharing aggregate computation for distributed queries," in *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM Press, 2007, pp. 485–496.

[27] A. Silberstein and J. Yang, "Many-to-many aggregation for sensor networks," in *ICDE '07: Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 986–995.

[28] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics*, vol. 2, pp. 83–97, 1955.

[29] G. Aggarwal and J. D. Hartline, "Knapsack auctions," in *SODA '06: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm*. New York, NY, USA: ACM Press, 2006, pp. 1083–1092.

[30] C. Ré, N. Dalvi, and D. Suciu, "Efficient top-k query evaluation on probabilistic data," in *ICDE '07: Proceedings of the 23rd IEEE International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 886–895.