Optimizing Query Rewrites for Keyword-Based Advertising

Azarakhsh Malekian*
Department of Computer Science
University of Maryland
College Park, MD 20742.
malekian@cs.umd.edu

Chi-Chao Chang Ravi Kumar Grant Wang
Yahoo!
701 First Ave
Sunnyvale, CA 94089.
{chichao,ravikumar,gjw}@yahoo-inc.com

ABSTRACT

We consider the problem of query rewrites in the context of pay-per-click search advertising. Given a three-layer graph consisting of queries, query rewrites, and the corresponding ads that can be served for the rewrites, we formulate a family of graph covering problems whose goals are to suggest a subset of ads with the maximum benefit by suggesting rewrites for a given query. We obtain constant-factor approximation algorithms for these covering problems, under two versions of constraints and a realistic notion of ad benefit. We perform experiments on real data and show that our algorithms are capable of outperforming a competitive baseline algorithm in terms of the benefit of the rewrites.

Categories and Subject Descriptors

F.2.m [**Theory of Computing**]: Analysis of Algorithms and Problem Complexity—*Miscellaneous*

General Terms

Algorithms, Economics, Experimentation, Theory

Keywords

Keyword-based advertising, Greedy algorithm, Submodularity, Query rewriting

1. INTRODUCTION

Pay-per-click search advertising continues to power the growth of online advertising. Yahoo! Sponsored Search, Microsoft AdCenter, and Google Adwords are examples of search advertising networks, commanding over seven billion dollars in annual revenues in 2007. Such networks are composed of affiliate search publishers and advertisers. At their core, these search advertising networks match ads to user queries. Advertisers bid on search keywords that are most likely to generate clicks and conversions for them. Publishers want relevant ads for their search users to maximize revenue on their sites; they are paid by the search network through revenue share (a fraction of the payment advertisers make for each click goes to them).

A major goal of search ad networks is to show relevant ads for a given query. In the short term, users will click on relevant ads and generate revenue for the search ad network. In the long term, users will recall that ads were relevant to them and thus continue to click on them. In both the long and short term, this creates revenue for the publishers and generates leads for the advertisers.

Although advertisers/ads bid on keywords, a relevant ad for a given query may not necessarily exist among the set of ads that have bid for that query. Indeed, that set may be empty, even though a relevant ad exists. For instance, an ad bidding on the keyword "wedding band" may be appropriate for the query "engagement ring". The traditional information retrieval problems of polysemy and synonymy occur in this setting.

Query rewriting. A common mechanism used to improve the relevance in information retrieval is query rewriting. At a high level, query rewriting outputs a list of queries (referred to as rewrites) that are related to a given query.

Query rewriting is a well studied problem (see Section 2). Most work on this topic, however, has primarily focused on generating relevant rewrites with respect to the original query using a variety of methods such as mining query logs and user sessions. These lines of work do not directly provide a framework to optimize for the relevance of the ads of the rewrites[1] subject to:

(1) constraints on the number of rewrites per query — typically the number of rewrites is a constant across all queries subject to system considerations such as fitting the rewrite hash table into the main memory of the ad servers, the maximum number of keys in the reverse index, indexing latency, etc;

(2) constraints on the number of queries for which a query rewrite can be used — if a rewrite is used too often, it can lead to the same ads being shown to users, which is undesirable;

(3) budget constraints of an ad — each advertiser has a limited budget and an ad that cannot be shown due to a consumed budget cannot provide relevance.

*Part of this work was done when the author was visiting Yahoo!

[1]This is mainly because query rewriting has been a widely-used tool in web search but not so much so in keyword advertising.

EC'08, July 8–12, 2008, Chicago, Illinois, USA.

Our contributions. In this paper we propose a combinatorial framework for optimizing query rewrites, taking the relevance and budget constraints of ads into account. We assume the existence of a query rewrite generator that outputs a list of candidate rewrites for a given query along with a score indicating the relevance of the rewrite with respect to the query. The framework exposes the set of all ads that can be served from the candidate rewrites, allowing for the definition of general ad benefit functions over any subset of these ads. We assume that the average traffic rate for each query and average budget of each ad for a fixed period of time is known.

At the heart of our formulation is a graph covering problem on a graph with three sets of vertices: queries, rewrites, and ads. The goal is to select rewrites from the second vertex set for each query in the first vertex set, so that the *benefit* of the ads adjacent to the rewrites is maximized subject to ad system and budget constraints; we will use a realistic notion of ad benefit that captures display real-estate and user experience constraints. All of the variants of this problem are NP-hard and so we focus on designing efficient approximation algorithms.

We look at two variants of system and budget constraints. In the first variant (called the *cardinality version*), the constraints specify upper bounds on both the number of rewrites a query can have as well as the number of queries for which a query rewrite can be used. These model the system constraints in an ad network: too many rewrites for a given query will slow down the time needed to serve an ad and using the same rewrite for too many queries will make the ads less diverse. For this variant, we give a greedy algorithm with an approximation ratio of $(e-1)/(2e-1) \approx 0.387$. This ratio is an improvement over 1/3 that can be obtained using existing results on greedy algorithms for matroid intersections; we believe this may be of independent interest. For the special case of a single query, we obtain a stronger approximation ratio of $1 - 1/e \approx 0.632$.

In the second variant (called the *weighted version*), we model the constraints of an ad's budget. We assume that the traffic for a query for a fixed period of time is known. The goal here is again to select a set of rewrites that have the maximum benefit subject to the constraint that no query can have too many rewrites. The key difference is that an ad can only contribute benefit for traffic up to its budget. For this problem we give a greedy algorithm with an approximation ratio of 1/4.

We also conduct experiments to measure the performance of some of our algorithms. For the case of determining rewrites for a single query, we compare the ad benefit of our greedy algorithm to the ad benefit of a baseline algorithm that is a variant of the k-nearest neighbor algorithm. Our experimental results show that while query rewrites suggested by the greedy algorithm achieve similar relevance compared to the baseline, they significantly outperform the baseline in terms of the ad benefit of the rewrites.

1.1 Why rewrite queries for keyword advertising?

If our main objective is to serve the most relevant set of ads for a query, it is conceivable to estimate the relevance of every ad with respect to all known keywords a priori (e.g., offline, say, with a machine-learned relevance ranking model) and build a keyword-ad index mapping keywords to their most relevant ads. Given this, why is query rewriting related at all to keyword advertisement? We offer three motivating points.

(1) The advertiser bidding landscape for a keyword is very dynamic and fast changing. Advertisers manually or automatically distribute their spend throughout the day by turning on and off their ads. Yahoo!, for example, offers a maximum of 15 min delay before ads effectively go on- or off-line. With such tight system requirements, it is more practical to add (or remove) a list from only one keyword (the one they are bidding on) rather than hundreds or even thousands (as it could potentially be the case if the indexing is based on an *a priori* computation of keyword-ad relevance).

(2) Search ad networks need cost-effective experimentation capabilities. Experimenting with various query rewrite algorithms using live A/B testing is far easier and cheaper than having to set up a few clusters of ad indices.

(3) Search advertising networks evolve from simple entities offering hundreds or thousands of search keywords to be purchased by advertisers to massive agencies and networks offering millions of keywords as well as "packages" or "bags" of keywords. Not surprisingly, proprietary ad serving systems go from in-memory hash tables and MySQL databases to possibly full-fledged search engines with time. Along the way, due to legacy reasons, query rewriting based on keyword clustering, keyword graph mining, etc, are viable techniques to improving ad relevance and coverage.

Organization. The rest of the paper is organized as follows. In Section 2, we discuss some related work. In Section 3, we state our combinatorial formulation and provide some background material. Section 4 contains the results for the cardinality version and Section 5 contains the results for the weighted version. Section 6 contains the experimental results. Finally, Section 7 contains concluding remarks. The Appendix discusses small variants of the two problem versions considered.

2. RELATED WORK

Query rewriting. There is a vast amount of literature on clustering and mining of search logs to generate query suggestions for improving web and paid search results. Jones et al [13] used query reformulation sessions to pre-compute similar queries and phrases with affinity scores. For an incoming query, these tables are consulted in order to generate candidate rewrites; ranking of the rewrites is achieved using a machine-learned function. Recently, Zhang et al [21, 20] have improved its performance using click logs and active learning techniques and with additional features such as web search results page co-occurrence and advertiser cobiddedness.

Other classes of work revolve around exploring the context and structure of query and click logs to cluster related queries. For example, Beeferman and Berger [2] applied agglomerative clustering techniques to bipartite click graphs using a simple set overlap distance function. Antonellis et al [1] build upon Simrank [12], a measure of structural-context similarity developed for personalized web graphs, to identify similar queries. Another method commonly employed is latent semantic analysis based on SVD [5]. SVD allows the rearrangement of the term–document space to reflect

the major associative patterns; typically, these methods are computationally expensive.

The algorithms presented in this paper complement all these techniques by optimizing the selection of the rewrites by taking the ad benefit into account.

Submodularity and greedy algorithms. Submodularity has been studied in more depth in recent years due to its applications in combinatorial auctions, e.g., the submodular welfare problem [16], [14], generalized assignment problems [8], etc. The greedy approach is a natural tool to solve maximization problems with a submodular objective function. Nemhauser and Wolsey [18] showed that greedy approach gives an $e/(e-1)$-approximation for maximizing a non-decreasing submodular function over a uniform matroid. Nemhauser, Wolsey, and Fisher [19] considered this problem over the independence system. They showed that if the independence system is the intersection of M matroids, the greedy algorithm gives an $M+1$ approximation. Recently, Goundan and Schulz [9] generalized both these results and showed that if an α-approximate incremental oracle is available, then the greedy solution is a $e^{1/\alpha}/(e^{1/\alpha}-1)$ approximation for maximizing a non-decreasing submodular functions over a uniform matroid and an $\alpha M+1$ approximation for the intersection of M matroids. Feige, Mirrokni, and Vondrak [7] gave a general framework for solving the non-monotone submodular problems.

3. FORMULATION

In this section we formalize the problem framework and provide the necessary notation and technical background. Consider a three-layer graph G with vertex set (Q, W, A) and edge sets $E_Q \subseteq Q \times W$ and $E_A \subseteq W \times A$. Here, Q represents all the queries, W represents all possible rewrites for the queries Q, and A represents the set of potential ads. The edge $(q, w) \in E_Q$ means that w is a possible rewrite for q and the edge $(w, a) \in E_A$ means that ad a can be shown for rewrite w. The general goal is to suggest a subset A' of ads that maximizes certain "benefit," subject to certain "constraints." As we discussed before, this goal will be achieved instead by suggesting query rewrites for each query, i.e., a subset $E'_Q \subseteq E_Q$ of edges such that each query has at least one rewrite, that will lead to the most beneficial subset A' of ads.

Notation. We use the following notation.

$\Gamma(Q')$: For a subset $Q' \subseteq Q$ of queries, $\Gamma(Q') \subseteq W$ denotes the set of rewrites that are adjacent to some query in Q'.

$\Gamma(W')$: For a subset $W' \subseteq W$ of rewrites, $\Gamma(W') \subseteq A$ denotes the set of ads that adjacent to some rewrite in W'.

$\beta_q(a)$: For each $q \in Q$ and $a \in A$, $\beta_q(a) \geq 0$ captures the *benefit* of showing ad a for query q.

$\beta_{q,d}(A')$: For $A' \subseteq A$, the benefit provided by the d most beneficial ads in A' (called *max d-benefit*) is defined as

$$\beta_{q,d}(A') = \max_{A_s \subseteq A', |A_s|=d} \beta_q(A_s).$$

$B_{q,d}(A')$: Given $A' \subseteq A$, $B_{q,d}(A')$ is the set of d ads in A' that provides the max d-benefit.

Similarly, $\beta_{q,d}(W')$ is the benefit obtained from the set of top d ads belonging to $\Gamma(W')$ according to the benefit function $\beta_q(\cdot)$, where $W' \subseteq \Gamma(\{q\})$ and likewise, $B_{q,d}(W')$ is the set of ads that are contributing in $\beta_{q,d}(W')$. Finally, $\beta_d(E'_Q)$ is the benefit due to the set of query, rewrite pairs: $\sum_{q|(q,\cdot)\in E'_Q} \beta_{q,d}(\{w \mid (q,w) \in E'_Q\})$.

Note that *max d-benefit* captures realistic constraints that limit the number of ads shown, including screen real-estate, user experience, etc. Given this framework, we now describe the two flavors of benefit and constraints that will be used in our study. A constraint that is common to both versions is the following:

(a) given $K > 0$, there are no more than K rewrites for each query, i.e., for each $q \in Q$, we have $|\{w \mid (q,w) \in E'_Q\}| \leq K$.

Next, we describe the specific constraint of each variant as well as the desired objective function.

Cardinality version. In the cardinality version of the problem, we are given $d > 0$, and a function $D : W \to \mathbb{R}^{\geq 0}$. Our constraint is then:

(b) there are no more than $D(w)$ queries for which w is a rewrite, i.e., for all $w \in W$, we have $|\{q \mid (q,w) \in E'_Q\}| \leq D(w)$.

The objective is to find E'_Q, subject to constraints (a) and (b), such that the d-benefit $\beta_d(E'_Q)$ is maximized.

Weighted version. In the weighted version of the problem, we also consider the budget of each ad and the traffic that each query receives. Again, the objective in this problem is to select rewrites so that the benefit of the ads adjacent to the rewrites is maximized. The added constraint here is that each ad has a budget and can only contribute benefit up to its budget — if a query has t units of traffic and has a budget of ℓ, with $t > \ell$, the benefit of an ad can only be obtained ℓ times. The exact setup we assume is the following. Given a set of query rewrites, i.e., a subset of edges in E_Q, an *ad allocator* will *optimally allocate* (with respect to the traffic of each query and the budget of each ad) d ads that are covered by the rewrite set to each query. The goal is to suggest the rewrites in a way that the total benefit that an ad allocator can obtain will be maximized. Let the benefit gained from showing an ad a for t units of traffic for a query q be $t \cdot \beta_q(a)$.

Formally, we are given two functions $T : Q \to \mathbb{R}^+$ and $L : A \to \mathbb{R}^+$. The constraint is then:

(c) each ad a can be shown for at most $L(a)$ units of traffic, i.e., assuming that queries $q_1, \ldots, q_\ell$ are shown ad a for $T_1, \ldots, T_\ell$ units of traffic, $\sum_{i=1}^{\ell} T_i \leq L(a)$.

Background. Now, we will introduce some general concepts that will be useful in developing algorithms for the max benefit problems.

Definition 1 (Non-decreasing submodularity). *Let U be a finite set. A function $f : 2^U \to \mathbb{R}$ is* non-decreasing *and* submodular *if*

1. $f(0) = 0$,
2. $f(X) \leq f(Y)$ *when* $X \subseteq Y \subseteq U$.
3. $f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y)$, $\forall X, Y \subseteq U$, *or equivalently,*

3'. $f(X \cup \{u\}) - f(X) \geq f(Y \cup \{u\}) - f(Y)$, $\forall X \subseteq Y \subseteq U$.

A natural way to optimize covering problems, where the function is non-decreasing and submodular, is the *greedy* approach. Start with an empty set and iteratively build the solution. At each iteration, select the element with the highest incremental benefit to the current solution and add it to the current solution.

THEOREM 2 ([19]). *The greedy algorithm produces a* $(1-1/e)$*-approximation to covering problems with non-decreasing submodularity property.*

4. ALGORITHMS FOR CARDINALITY VERSION

In this section we obtain approximation algorithms for the cardinality version. First, we consider the case when there is a single query and obtain a simple greedy algorithm. Next, we focus on the general case of multiple queries where we have to optimize simultaneously over all the queries.

4.1 Warmup: Single query case

Note that since there is a single query, constraint (b) is vacuous. The query rewrites can therefore be simply represented as $W' \subseteq W$ and constraint (a) becomes a single constraint $|W'| \leq K$. Also, $\beta_d(E'_Q) = \beta_{q,d}(\Gamma(W'))$. For the remainder of the section, we will use these conventions.

The approximation algorithm we describe is a greedy algorithm. At each step of the greedy algorithm, the query rewrite that gives the maximum incremental d-benefit will be selected. Let $Q = \{q\}$.

Algorithm SINGLE-QUERY-GREEDY

Set $W' \leftarrow \emptyset$.
While $|W'| \leq K$ do,
  Find $w \in W \setminus W'$ that maximizes $\beta_{q,d}(\Gamma(W' \cup \{w\}))$.
  Set $W' \leftarrow W' \cup \{w\}$.
Return W'.

We show that Algorithm SINGLE-QUERY-GREEDY gives a $(1-1/e)$-approximation to the cardinality version for the single query case. We do this by showing that the objective function has the non-decreasing submodularity property. The approximation guarantee then follows by appealing to Theorem 2.

THEOREM 3. *The max d-benefit function is non-decreasing and submodular.*

PROOF. The non-decreasing property is immediate. We use the characterization (3') in Definition 1 to show submodularity. In our case, we have $f(W') = \beta_{q,d}(\Gamma(W'))$ for $W' \subseteq W$. We need to show that for a given $w \in W$ and $W_1, W_2 \subset W$ with $W_1 \subseteq W_2$:

$$f(W_1 \cup \{w\}) - f(W_1) \geq f(W_2 \cup \{w\}) - f(W_2).$$

First, we observe that if $a \in \Gamma(w) \setminus \Gamma(W_2)$ and $a \in B_{q,d}(W_2 \cup \{w\})$, then $a \in B_{q,d}(W_1 \cup \{w\})$. Let X be the set of ads covered by w that are added to $B_{q,d}(W_2 \cup \{w\})$, i.e.,

$$X = \{a \in B_{q,d}(W_2 \cup \{w\}) \setminus B_{q,d}(W_2) : a \in \Gamma(w)\}.$$

Let $Y \subseteq B_{q,d}(W_2)$ be the set of ads such that $|Y| = |X|$ with the lowest sum $\sum_{a \in Y} \beta_q(a)$, i.e., the least $|X|$ beneficial ads in $B_{q,d}(W_2)$. Similarly, let $Z \subseteq B_{q,d}(W_1)$ be the set of ads such that $|Z| = |X|$ and with the lowest sum $\sum_{a \in Z} \beta_q(a)$, i.e., the least $|X|$ beneficial ads in $B_{q,d}(W_1)$. We have

$$f(W_2 \cup \{w\}) - f(W_2) = \sum_{a \in X} \beta_q(a) - \sum_{a \in Y} \beta_q(a).$$

From the above mentioned fact, $X \subseteq B_{q,d}(W_1 \cup \{w\})$. Therefore,

$$f(W_1 \cup \{w\}) - f(W_1) \geq \sum_{a \in X} \beta_q(a) - \sum_{a \in Z} \beta_q(a).$$

Since $W_1 \subseteq W_2$, we have $\sum_{a \in Z} \beta_q(a) \leq \sum_{a \in Y} \beta_q(a)$. This shows that the increase in benefit caused by adding w to W_1 is at least that of adding w to W_2, establishing (3'). □

It is easy to see that Algorithm SINGLE-QUERY-GREEDY runs in time $O((\sum_{w \in W} |\Gamma(W)|) \cdot K(\log d))$. In Appendix A, we consider a slightly different constraint — given $L > 0$, we need $|A'| \leq L$, i.e., we require that query rewrites cover at most L ads — and show a $(1/2)(1-1/e)$-approximation with this constraint.

4.2 The general case

We now consider the cardinality version with constraints (a) and (b). To simplify notation, given a solution E'_Q, let R_q be the set of rewrites for query q as given by E'_Q, i.e., $R_q = \{w \mid (q,w) \in E'_Q\}$. Thus, $\beta_d(E'_Q) = \sum_{q \in Q} \beta_{q,d}(R_q)$. The approximation algorithm is once again greedy-based.

Algorithm GENERAL-GREEDY

Set $E_g \leftarrow \emptyset$.
While $\exists e \in E_Q$ that is unmarked
  Find an unmarked candidate edge $e = (q^*, w) \in E_Q$
    that maximizes the incremental d-benefit:
  $\sum_{q \in Q \setminus \{q^*\}} \beta_{q,d}(R_q) + \beta_{q^*,d}(R_{q^*} \cup \{w\})$.
  If (q^*, w) does not violate the constraints, i.e.,
    if $|R_{q^*}| < K$ and $|\{q : (q,w) \in E_g\}| < D(w)$,
  add (q^*, w) to E_g.
  Mark (q^*, w).
Return E_g.

It turns out that our problem can be formulated as maximizing a submodular function over the intersection of two matroids. It is known that the greedy algorithm provides a tight $1/(M+1)$ approximation for maximizing a submodular function over the intersection of M matroids [19]. For our problem, this implies that the greedy algorithm provides a $1/3$ approximation. However, using the structure of our objective function, which is a sum of submodular functions, we improve the approximation ratio to $(e-1)/(2e-1)$ using a careful charging argument; this improvement might be of independent interest.

THEOREM 4. *Algorithm* GENERAL-GREEDY *provides an* $(e-1)/(2e-1) \approx 0.387$ *approximation for the cardinality version.*

PROOF. Denote by $E_g = \{e_1, \ldots, e_m\} \subseteq E_Q$ the set of edges chosen by Algorithm GENERAL-GREEDY and by $E_o = \{e'_1, \ldots, e'_n\}$ the set of edges in the optimal solution.

We first add all the edges in $E_o \setminus E_g$ to E_g. We will charge the increase in benefit coming from these new edges to the edges in E_g. Note that in Algorithm GENERAL-GREEDY, each edge in E_Q is considered exactly once to be added to E_g. We now describe the charging scheme.

Consider the iteration in which an edge $e'_i = (q, w) \in E_o \setminus E_g$ is selected as a candidate by the greedy algorithm, i.e., it

is the edge that maximizes the incremental d-benefit. Since e'_i was not added to the greedy solution, this means that either we have already selected K rewrites for q (meaning $|R_q| = K$) or w has been chosen as a rewrite $D(w)$ times (meaning $|\{q : (q, w) \in E_g\}| = D(w)$).

Partition the set of edges in $E_o \setminus E_g$ into two groups, S_Q and S_W. S_Q consists of those edges that were not added to the greedy solution because the endpoint in Q was tight ($|R_q| = K$), and S_W consists of those edges that were not added because the endpoint in W was tight. If an edge was tight for both sides, it is only in S_W.

First consider an edge $(q, w) \in S_W$. When (q, w) was chosen as a candidate in Algorithm General-Greedy but was not added to E_g, w had already been suggested as a rewrite for $D(w)$ other queries. Since the selection of the edges is greedy, each of the previous $D(w)$ edges adjacent to w added more incremental d-benefit at the time they were added to E_g. Since the d-benefit function is submodular, the incremental benefit from adding (q, w) to E_g is less than the incremental benefit from adding the previous $D(w)$ edges adjacent to w at the time they were added. Since the number of edges in S_W adjacent to w is at most $D(w)$, we can create a one-to-one mapping between the incremental benefit coming from S_W and the incremental benefit from the edges adjacent to w chosen by Algorithm General-Greedy.

Next consider the edges in S_Q. We charge the incremental benefit for all the edges in S_Q adjacent to a vertex q at the same time. Consider a fixed vertex q. Let the edges in E_g incident to q be $E_K = \{e_1, \ldots, e_K\}$ (where $e_1, \ldots, e_K$ was the order that the edges were added) and the edges in S_Q adjacent to q be $E_{K'} = \{e'_1, \ldots, e'_{K'}\}$, with $K' \leq K$. We will show that the benefit obtained from E_K is at least $1-1/e$ of the benefit obtained from $E_{K'}$. The key observation is that when each edge $(q, w) \in E_{K'}$ was a candidate to be added to E_g, the edge was only tight on the Q side, i.e., there were already K edges adjacent to q in E_g. Again, this implies that each of the edges in E_K added more incremental benefit than $E_{K'}$. Let the total benefit obtained from $E_{K'}$ be OPT_q and the incremental benefit from $e_1, \ldots, e_K$ be $s_1, \ldots, s_K$. Since $e_1, \ldots, e_K$ were chosen greedily we can conclude that:

$$s_1 \geq \frac{OPT_q}{K}, s_2 \geq \frac{OPT_q - s_1}{K}, \ldots, s_K \geq \frac{OPT_q - \sum_{i=1}^{K-1} s_i}{K}.$$

The total benefit obtained by E_K can be represented as $\sum_{i=1}^{K} s_i$. It is easy to see that

$$\sum_{i=1}^{K} s_i \geq \left(1 - \left(1 - \frac{1}{K}\right)^K\right) OPT_q \geq \left(1 - \frac{1}{e}\right) OPT_q.$$

Since the benefit of the edges in S_W can be charged bijectively to edges in E_g, and the benefit of the edges in S_Q is no more than $e/(e-1)$ of the benefit of the edges in E_g, we can conclude that $\beta_d(E_g) \geq (e-1)/(2e-1)OPT$. □

5. ALGORITHMS FOR WEIGHTED VERSION

In this section we consider the weighted version. Again our goal is to suggest K rewrites for each query to maximize the total benefit. The benefit is computed based on the d ads that are shown for each query. An ad can be shown for a query if it is covered by one of the suggested rewrites for that query. However in this scenario we have some new constraints as well. The parameters that will come into play are the traffic of a query and the budget of an ad. Let $T(q)$ represent the traffic for $q \in Q$ for a fixed period of time and let $L(a)$ denote a limit on the amount of traffic that can be allotted for an ad for the same period. We will refer to $L(a)$ as a budget constraint. Now, we discuss how to compute the benefit in this scenario for each set of suggested rewrites. We assume an *optimal ad allocator*, which takes the rewrites for the queries and allocate at most d ads to each query (an ad a can be allocated to a query q if there is some rewrite w for q such that $(w, a) \in E_A$).

The ad allocator allocates ads to queries so as to maximize the total ad benefit, but subject to budget constraints. This means that the total traffic of the queries for which an ad has been allocated can be more than the budget constraint of an ad. There are two sensible ways to make this precise:

(1) *Integral ad allocator*: If an ad a is allocated to a query q, then either all of the traffic $T(q)$ is assigned to that ad, or none of it. This means that there could possibly be some ϵ amount of budget $L(a)$ left for an ad after allocation since the traffic $T(q)$ of each query q is bigger than ϵ.

(2) *Fractional ad allocator*: Here, if an ad a is allocated to a query q, then a *fractional* amount of the traffic $T(q)$ can be assigned to that ad.

In the weighted version, the suggested rewrites will define which ads can be shown for each query and then the *ad allocator* will select which ads to show for each query. In this section we consider the optimal *integral ad allocator* model. In Appendix B, we consider a slightly stringent constraint on the budget, which makes the problem extremely hard, and hence less appealing. In Appendix C, we describe algorithms where we assume an optimal *fractional ad allocator*.

Our method for selecting the rewrites in this scenario is again based on the greedy approach.

Algorithm Greedy-Benefit

Let S be the set of rewrites.
Construct a bipartite graph G with vertex sets Q, A and edge set E. Add an edge $(q, a) \in E$ iff $\exists e = (q, w) \in S$ so that $a \in \Gamma(w)$.
Sort the edges in E by $\beta_q(a)$.
Add feasible edges in the sorted order above and accumulate the total benefit of S.

Algorithm Greedy-Budget

Set $E_g \leftarrow \emptyset$.
For each $e = (q, w) \in E_Q$, (approximately) compute the benefit of $E_g \cup \{e\}$ using Algorithm Greedy-Benefit.
Select e such that $E_g \cup \{e\}$ has the maximum benefit.
Mark e.
Add e to E_g if q has at most $K-1$ rewrites in E_g.
Repeat the above steps until all the edges in E_Q are marked.

It can be shown that selecting the edge $e \in E_Q$ that maximizes the benefit in each stage of Algorithm Greedy-Budget is NP-hard.

Theorem 5. *Selecting the edge $e \in E_Q$ with maximum incremental benefit is NP-hard.*

Proof. We reduce 3-partition to this problem. Consider an instance of 3-partition problem with $n = 3m$ elements in which the value of each element is between $B/4$ and $B/2$ and the total value of all the elements is mB. The goal is to partition the set into triples with total valuation B each.

Now create an instance of the query-ad graph as follows. There are n queries and m ads. An edge exists between a query and an ad with benefit equal to the weight of the element that the ad stands for. Now, let $d = 3$, i.e., we only show 3 ads for each query and let the total allowed budget for each ad be equal to B. It can be seen that there is an optimal query-ad assignment iff there is a 3-partition of the original set. □

Finally, we show that Algorithm Greedy-Budget returns a solution with benefit at least 1/4 of the optimal. The argument has two parts. We first show that Algorithm Greedy-Benefit computes a 1/2 approximation to the optimal benefit. Next we show that if we added the edge with optimal benefit in each round, Algorithm Greedy-Budget achieves a 1/2 approximation. Putting the two results together gives us a 1/4 approximation.

Lemma 6. *Consider a set S of rewrites. The benefit of S computed by Algorithm* Greedy-Benefit *is at least half of the optimal benefit that can be obtained from S.*

Proof. As before, construct the bipartite graph G' with vertices (Q, A) and edges E'. Add an edge (q, a) to E' iff $\exists (q, w) \in S$ for which $a \in \Gamma(w)$. We show that if we compute the benefit in this graph greedily, it is at least half of the optimal benefit obtainable from that set. Consider an edge (x, y) that is in S but not chosen in our solution. This means that at the stage that it was considered (and marked) in the greedy solution, at least one of the endpoints was tight. In other words, either d ads are already allocated to x or y received at least $L(y)$ amount of traffic. In both cases, the weight of the edges that are already assigned to the endpoints are higher since we chose the edges greedily. If x is tight, we charge the weight to any of the d edges. Note that each of these edges share the same query, so they also share the same amount of traffic. If the tight end point is y, we will charge that edge to the same share of traffic that is already assigned to y. Since for all the edges assigned to y their weight for the unit traffic that covered is higher, the benefit obtained from each unit of traffic of this edge can be charged to the benefit of one unit of traffic of that edge. Since we charge each unit traffic of each edge selected in the greedy solution at most twice, the returned solution by the greedy is at least half of the optimal benefit obtainable from S. □

Using the above result we can show that the following.

Theorem 7. *Algorithm* Greedy-Budget *achieves a* 1/4 *approximation for the weighted version.*

Proof. Let S be a set of rewrites. We will refer to the approximate benefit of S computed by Algorithm Greedy-Benefit as the *greedy-benefit* of S. The optimal benefit of S will be referred to as *optimal-benefit*.

In this proof, we use the submodularity property to charge the optimal solution for the *greedy-benefit* to the selected greedy rewrite pairs. Consider the set of rewrites that are suggested by algorithm Greedy-Budget (call it E_g) and then add all the rewrites that are suggested by the optimal solution (call it E_o). We will update the total benefit obtained using the *greedy-benefit*. Consider an edge $e' \in E_o \setminus E_g$. Since $e' \notin E_g$, this means that it would increase the *greedy-benefit* less than all the selected rewrites in the greedy solution at the time they were chosen. This means that we can charge the increase in *greedy-benefit* from adding e' to the current solution set to one of those edges. Note that the only constraint that bans a rewrite from being selected is the constraint on the number of rewrites K for a query. Thus, we conclude that we can make a one-to-one correspondence between the rewrites adjacent to $q \in Q$ that are in $E_o \setminus E_g$ and the rewrites adjacent to q that are in $E_g \setminus E_o$. This means that the greedy-benefit obtained from $E_g \cup E_o$ is at most twice the greedy-benefit of E_g. By using Theorem 6 we know that the optimal benefit from $E_g \cup E_o$ is at most twice its greedy-benefit. The optimal benefit from $E_g \cup E_o$ is at least OPT. That means that the optimal benefit of E_g, which is larger than its greedy benefit, is at least 1/4 of the optimal solution. □

6. EXPERIMENTAL RESULTS

We evaluated the performance of Algorithm Single-Query-Greedy through two experiments. In the first, we compared the relevance of the rewrites selected by Algorithm Single-Query-Greedy with the rewrites selected by a baseline algorithm (Algorithm Baseline). The relevance of these rewrites was determined by a set of human editors. In the second experiment, we compared the d-benefit of Algorithm Single-Query-Greedy to the d-benefit of the Algorithm Baseline.

Algorithm Baseline selects rewrites using the same graph $(\{q\}, W, A)$ that is used by Algorithm Single-Query-Greedy. To select the rewrites for a given query q, Algorithm Baseline computes a similarity measure between between all pairs (q, w), where $w \in W$. The similarity measure Baseline uses is Pearson correlation, which has been used in many other settings as a similarity measure (e.g. [3, 10, 1]). Pearson correlation is defined on two random variables X, Y with means μ_X, μ_Y and standard deviations σ_X, σ_Y as:

$$p(X, Y) = \frac{\mathsf{E}((X - \mu_X)(Y - \mu_y))}{\sigma_X \sigma_Y}.$$

To compute the Pearson correlation $r(q_1, q_2)$ between two queries, we identify a query q with a random variable X_q defined as follows — with probability $1/|\mathcal{A}|$, $X_q = \beta(a)$ if $(q, a) \in E$, and 0 otherwise. Algorithm Baseline selects k rewrites with the highest Pearson correlation with q. The idea is that queries with high similarity to q are relevant.

Our results show that while the rewrites selected by Algorithm Single-Query-Greedy are similar in relevance to the rewrites found by Algorithm Baseline, the former significantly outperforms the latter when comparing the d-benefit of the selected rewrites. We describe our results in detail in the next two sections.

6.1 Editorial relevance

In this experiment we selected rewrites for 1,170 queries. The 1,170 queries were chosen randomly from a search log. We compared the relevance of the top five rewrites selected by Algorithm Single-Query-Greedy and the top five rewrites of Algorithm Baseline. We ran Algorithm Single-Query-Greedy with the constraint that $|W'| \leq 5$ while attempting to maximize 10-benefit, i.e., the benefit of the top 10 ads.

The relevance of the rewrites was judged on a four point scale identical to the scale used in [13, 1]:

(1) Precise match: A near-certain match — the rewrite precisely matches the user's intent.

Scale	BASELINE	SINGLE-QUERY-GREEDY
1	24%	21%
1-2	72%	72%
1-3	91%	91%

Table 1: Relevance comparison: Algorithm BASELINE vs. Algorithm SINGLE-QUERY GREEDY

(2) Approximate match: A probable, but inexact match with user intent.

(3) Marginal match: A distant, but plausible match to a related topic.

(4) Clear mismatch: A clear mismatch.

The results of the experiment are summarized in Table 1.

Algorithm BASELINE does place a larger fraction of the rewrites it selects into category **(1)**, i.e., precise match; otherwise, the relevance of the two rewriting systems is equivalent.

6.2 Ad benefit

In this experiment we compared the d-benefit of the rewrites selected by Algorithm SINGLE-QUERY-GREEDY to the rewrites selected by Algorithm BASELINE. We computed rewrites for a set of $89,000$ queries chosen randomly from a search log.

One choice of the benefit $\beta(a)$ of an ad for a given query q is the estimated click-through rate for the (q, a) pair. We used this notion of benefit in this experiment and describe briefly how it was computed. Recall the graph $(\{q\}, W, A)$. For each edge $(w, a) \in E_A$, we have a historical estimate of the click-through (CTR) rate for the query-ad pair. We would like $\beta(a)$ to be proportional to the CTR for the pair (q, a), but the CTR of the pair (w, a) will not necessarily be a good estimate. This is because some queries in W will be more relevant to q than others. This motivates our definition of $\beta(a)$ in these experiments:

$$\beta(a) = \frac{\sum_{w \in \Gamma(a)} r(q, w) \cdot \mathsf{CTR}(w, a)}{\sum_{w \in \Gamma(a)} r(q, w)},$$

where $r(q, w)$ is the Pearson correlation defined above. In short, $\beta(a)$ is the similarity-weighted average of its CTRs.

6.2.1 *Comparing* SINGLE-QUERY-GREEDY *and* BASELINE

We compared the d-benefit of Algorithm SINGLE-QUERY-GREEDY and Algorithm BASELINE for different values of K (the constraint on the number of rewrites allowed for each query) and d (which specifies the number of ads that contribute to the benefit). To compare the performance of BASELINE and SINGLE-QUERY-GREEDY for a specific setting of K and d, we ran Algorithm SINGLE-QUERY-GREEDY with K and d to select a set W_G of K rewrites, and computed $\beta_d(\Gamma(W_G))$, i.e., the benefit of the top d ads adjacent to W_G. Recall that Algorithm BASELINE is not parameterized on either K or d. For BASELINE, we chose the set W_B of the top K rewrites ranked by BASELINE and computed $\beta_d(\Gamma(W_B))$.

The results of this experiment for different values of K and d are displayed in Table 2. In each cell, we give the percentage gain of SINGLE-QUERY-GREEDY over BASELINE, i.e., $100 \times$(SINGLE-QUERY-GREEDY $-$ BASELINE)/BASELINE. For small K, the gain of SINGLE-QUERY-GREEDY over BASELINE is significant. As K increases, the gain lessens. This is because the number of potential rewrites for many queries is less than K (i.e., $|W| \leq K$). For these queries, there is no difference between the rewrites suggested by SINGLE-QUERY-GREEDY or BASELINE. This observation is verified by Table 3. In this table, we break down the percentage gain in Table 2 by $|W|$. The large percentage gains come when $|W|$ is large relative to K, the number of rewrites we select. This occurs when the set of rewrites to choose from is quite large. Many of the rewrites are likely to be relevant and choosing the ones with the most beneficial ads gives large gains.

We also noticed that for $K > d$, SINGLE-QUERY-GREEDY computes the optimal solution. Indeed, for $K > d$, the optimal solution is easy to characterize — it is just the queries that are adjacent to the d ads with the greatest benefit. This means that the percentage gain of SINGLE-QUERY-GREEDY over BASELINE for $K > d$ is just an indicator of how well BASELINE performs compared to the optimal solution.

	d				
	2	4	6	8	10
$K = 1$	197.2%	201.9%	208.6%	214.9%	220.5%
$K = 2$	147.3%	156.7%	164.3%	171.4%	177.6%
$K = 4$	83.5%	96.4%	104.2%	109.9%	115.1%
$K = 8$	45.4%	49.9%	54.2%	58.5%	62.1%
$K = 16$	22.3%	24.1%	25.5%	27.0%	28.4%
$K = 32$	10.2%	10.9%	11.3%	11.8%	12.2%
$K = 64$	3.4%	3.6%	3.8%	4.0%	4.2%
$K = 128$	0.9%	1.0%	1.1%	1.1%	1.2%

Table 2: d-benefit: Percentage gain of Algorithm SINGLE-QUERY-GREEDY over Algorithm BASELINE.

7. CONCLUSIONS

We have formulated the problem of selecting rewrites to obtain the most beneficial set of ads. We give simple greedy algorithms for two realistic versions of the problem and prove that they obtain solutions whose score is within a constant-factor of the optimal solution. Experimentally, we compare the rewrites selected by the greedy algorithms to rewrites selected by a baseline algorithm in two categories: relevance and benefit. The results show that the rewrites selected by the greedy algorithm are nearly as relevant as the rewrites selected by the baseline but are much more beneficial, especially when the number of possible rewrites/ads is large relative to the number we wish to display.

Acknowledgments

We thank Saeed Alaei, Samir Khuller, and Mohammad Mahdian for various discussions.

8. REFERENCES

[1] I. Antonellis, H. G. Molina, C-C. Chang. Simrank++: Query rewriting through link analysis of the click graph. Stanford Computer Science Department Technical Report, 2007.

[2] D. Beeferman, A. Berger. Agglomerative clustering of a search engine query log. In *Proc. 6th KDD*, pages 407–416, 2000.

	range of $\|W\|$			
K, d	$[1, 2]$	$[3, 8]$	$[9, 32]$	$[33, \infty]$
1, 2	16.5%	130.0%	290.1%	486.1%
1, 4	17.3%	135.9%	299.1%	462.4%
1, 6	17.9%	140.5%	310.1%	460.1%
1, 8	18.3%	143.6%	320.1%	464.1%
1, 10	18.6%	146.0%	328.8%	469.9%
2, 2	0.0%	72.7%	212.0%	400.2%
2, 4	0.0%	81.5%	225.0%	390.8%
2, 6	0.0%	87.1%	237.1%	390.0%
2, 8	0.0%	91.1%	248.4%	394.5%
2, 10	0.0%	94.0%	258.6%	400.5%
4, 2	0.0%	17.2%	116.0%	262.3%
4, 4	0.0%	20.2%	130.4%	287.1%
4, 6	0.0%	22.9%	140.7%	294.1%
4, 8	0.0%	24.9%	149.1%	297.9%
4, 10	0.0%	26.5%	156.8%	302.4%
8, 2	0.0%	0.0%	49.2%	170.5%
8, 4	0.0%	0.0%	52.1%	178.9%
8, 6	0.0%	0.0%	55.4%	187.3%
8, 8	0.0%	0.0%	58.9%	196.5%
8, 10	0.0%	0.0%	62.3%	203.2%
16, 2	0.0%	0.0%	10.5%	101.8%
16, 4	0.0%	0.0%	10.9%	104.5%
16, 6	0.0%	0.0%	11.3%	106.5%
16, 8	0.0%	0.0%	11.8%	109.1%
16, 10	0.0%	0.0%	12.2%	112.0%
32, 2	0.0%	0.0%	0.0%	48.0%
32, 4	0.0%	0.0%	0.0%	48.3%
32, 6	0.0%	0.0%	0.0%	48.2%
32, 8	0.0%	0.0%	0.0%	48.5%
32, 10	0.0%	0.0%	0.0%	49.0%
64, 2	0.0%	0.0%	0.0%	13.2%
64, 4	0.0%	0.0%	0.0%	13.2%
64, 6	0.0%	0.0%	0.0%	13.5%
64, 8	0.0%	0.0%	0.0%	13.7%
64, 10	0.0%	0.0%	0.0%	14.0%
128, 2	0.0%	0.0%	0.0%	3.2%
128, 4	0.0%	0.0%	0.0%	3.3%
128, 6	0.0%	0.0%	0.0%	3.5%
128, 8	0.0%	0.0%	0.0%	3.6%
128, 10	0.0%	0.0%	0.0%	3.7%

Table 3: d-benefit: Percentage gain of Algorithm SINGLE-QUERY-GREEDY **over Algorithm** BASELINE **broken down by $|W|$.**

[3] J. Breese, D. Heckerman, C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proc. 14th UAI*, pages 43–52, 1998.

[4] S. Cucerzan, R. W. White. Query suggestion based on user landing pages. In *Proc. 30th SIGIR*, pages 875–876, 2007.

[5] S. C. Deerwester, S. T. Dumais, T. K. Lancaster, G. W. Furnas, R. A. Harshman. Indexing by latent semantic analysis. In *JASIS*, 41(6):391–407, 1990.

[6] D. Fain, J. Pedersen. Sponsored search. In *Bulletin of the American Society for Information Science and Technology*, 2005.

[7] U. Feige, V. Mirrokni, J. Vondrak. Maximizing non-monotone submodular functions. In *Proc. 48th FOCS*, pages 461–471, 2007.

[8] L. K. Fleisher, M. X. Goemans, V. S. Mirrokni, M. Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *Proc. 17th SODA*, pages 611–620, 2006.

[9] P. Goundan, A. Schulz. Revisiting the greedy approach to submodular set function maximization. In *Manuscript*, 2007.

[10] J. Herlocker, J. Konstan, A. Borchers, J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proc. 22nd SIGIR*, pages 230–237, 1999.

[11] D. S. Hochbaum, A. Pathria. Analysis of the greedy approach in covering problems. *Unpublished*, 1994.

[12] G. Jeh, J. Widom. Simrank: A measure of structural-context similarity. In *Proc. 8th KDD*, pages 538–543, 2002.

[13] R. Jones, B. Rey, O. Madani, W. Greiner. Generating query substitutions. In *Proc. 15th WWW*, pages 387–396, 2006.

[14] S. Khot, R. Lipton, E. Markakis, A. Mehta. Inapproximability results for combinatorial auctions with submodular utility functions. In *Proc. 1st WINE*, pages 92–101, 2005.

[15] S. Khuller, A. Moss, J. Naor. The budgeted maximum coverage problem. *IPL*, 70(1):39–45, 1999.

[16] B. Lehmann, D. J. Lehmann, N. Nisan. Combinatorial auctions with decreasing marginal utilities. In *Proc. 3rd EC*, pages 18–28, 2001.

[17] V. Mirrokni, M. Schapira, J. Vondrak. Tight information-theoretic lower bounds for welfare maximization in combinatorial auctions. *This proceedings.*

[18] G. L. Nemhauser, L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. In *Math. OR*, 3(3):177–188, 1978.

[19] G. L. Nemhauser, L. A. Wolsey, M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Math. Prog.*, 14:265–294, 1978.

[20] W. V. Zhang, X. H. Fei, B. Rey, R. Jones. Query rewriting using active learning for sponsored search. In *Proc. 30th SIGIR*, pages 853–854, 2007.

[21] W. Zhang, R. Jones. Comparing click logs and editorial labels for training query rewriting. In *Workshop on Query Log Analysis, 16th WWW*, 2007.

APPENDIX

A. SINGLE-QUERY, WITH CONSTRAINT ON NUMBER OF ADS

In this section we consider the single-query version with a different constraint:

(b') select at most L ads by picking one rewrite at a time.

This case turns out to be interesting since we lose submodularity. Despite this, it is possible to modify the greedy algorithm slightly to yield a constant factor approximation.

Suppose we have $L' < L$ ads before the last rewrite. Then, for the last rewrite, only $L-L'$ ads can contribute to the max benefit objective. We assume that if q is the last rewrite, then the $L - L'$ ads are chosen uniformly at random from $\Gamma(q)$. Since we treat the last chosen rewrite slightly different than the others, we alter the notation. Instead of Q, we will use (Q, q) as a parameter in all functions that shows the value of that function considering q as the last added element. We use $E(\beta_d(\Gamma(Q, q)))$ to define the expected value for $\beta_d(\Gamma(Q, q))$ for a given Q, q. Unlike the case of the constraint (b), we show that the non-decreasing and submodularity property does not hold with constraint (b').

LEMMA 8. *The single-query version with constraint (b') is neither submodular nor non-decreasing.*

PROOF. First, we consider the non-decreasing property. Suppose $Q = \{q\}$ and $\Gamma(q) = \{a_1, \ldots, a_L\}$ where $d = L$, $\beta(a_i) = m$ and m is a large integer. It can be seen that $\beta_d(Q) = Lm$. Consider q' where $\Gamma(q') = \{a'_1, \ldots, a'_L\}$ and $\beta(a'_i) = \epsilon$ where $\epsilon \approx 0$. We have $\Gamma(Q, q') = \{a_1, \ldots, a_L, a'_1, \ldots, a'_L\}$. Since $|\Gamma(Q, q')| > L$, by our assumption, L of the covered ads would be chosen uniformly at random. Now $E(\beta_d(Q, q')) = L\frac{m}{2} < Lm$, showing it can be decreasing.

To show that submodularity does not hold, consider $Q_1 = \{q_1\}$ and $Q_2 = \{q_1, q_2\}$. Also assume that $\Gamma(q_1) = \{a_1, \ldots, a_L\}$, $\Gamma(q_2) = \{a'_1, \ldots, a'_L\}$ and $\beta(a_i) = \beta(a'_i) = m$. It can be seen $E(\beta_d(\Gamma(Q_1))) = E(\beta_d(\Gamma(Q_2))) = Lm$. Now,

$$E(\beta_d(\Gamma(Q_1, q'))) = L\frac{m}{2} < L\frac{2m}{3} = E(\beta_d(\Gamma(Q_2, q'))).$$

This shows that gain of adding q' to Q_2, $-\frac{Nm}{3}$, is greater than $-\frac{Nm}{2}$, the gain of adding q' to Q_1. In other words, submodularity does not hold. □

Thus, the performance of the vanilla greedy algorithm is unbounded. However, we show how a slight modification to the greedy algorithm can give a $(1/2)(1 - 1/e)$-approximation for max d-benefit with constraint (b'). For a given Q, let $\text{size}(Q) = \sum_{q \in Q} |\Gamma(q)|$. The general framework of the algorithm is as follows.

Algorithm MODIFIED-GREEDY

Set $Q'_1 = \emptyset$.
While $\Gamma(Q'_1) \leq L$ do,
  Find $q \in \mathcal{Q} \setminus Q'$ that maximizes

$$\frac{E(\beta_d(\Gamma(Q'), \Gamma(q))) - \beta_d(\Gamma(Q'))}{\min(\text{size}(q), L - \text{size}(Q'))},$$

  breaking ties arbitrarily.
  Set $Q'_1 \leftarrow Q'_1 \cup \{q\}$.
Set $Q'_2 \leftarrow \arg\max_{q \in \mathcal{Q}} B_d(\Gamma(q))$.
If $\beta_d(\Gamma(Q'_2)) > \beta_d(\Gamma(Q'_1))$, then return Q'_2.
Otherwise return Q'_1.

We show that the MODIFIED-GREEDY algorithm gives a $(1/2)(1-1/e)$-approximation for the max d-benefit problem with the constraint (b').

THEOREM 9. *The* MODIFIED-GREEDY *algorithm is a* $(1/2)(1-1/e)$*-approximation for the single-query version with the constraint (b').*

PROOF. The proof consists of the following steps.

LEMMA 10. *For a given $q \in \mathcal{Q}$ and $Q' \subseteq \mathcal{Q}$, if we select a subset $Q_\ell \subseteq \Gamma(q)$ of size ℓ uniformly at random, then*

$$\frac{E(\beta_d(\Gamma(Q') \cup Q_\ell\})}{\ell} \geq \frac{\beta_d(\Gamma(Q' \cup \{q\}))}{|\Gamma(q)|}.$$

PROOF. Consider an ad $a \in B_d(Q' \cup \{q\}) \setminus B_d(Q')$. Then,

$$\Pr[a \in Q_\ell] = \frac{\ell}{|\Gamma(q)|}.$$

This will show that $E(\beta_d(Q' \cup Q_\ell)) - \beta_d(Q')$ is at least $\ell/|\Gamma(q)|$ of the expected increase when we add the whole set. Therefore, $(1/\ell) \cdot E(\beta_d(\Gamma(Q') \cup Q_\ell)))$ is at least $(1/|\Gamma(q)|) \cdot \beta_d(\Gamma(Q') \cup \{q\})$. □

Assume that the vertices selected by the greedy solution are $q_1, \ldots, q_n$. Let $Q_i = \{q_1, \ldots, q_i\}$. Let Δ_i be the expected gain by adding q_i to Q_{i-1}. Also let Q^* be the selected set by the optimal solution, excluding the last choice. Now we can show that.

LEMMA 11.

$$\Delta_i \geq \frac{\min(|\Gamma(q_i)|, L - \text{size}(Q_{i-1}))}{L}(\beta_d(\Gamma(Q^*)) - \beta_d(\Gamma(Q_{i-1}))) \quad (1)$$

PROOF. At the time q_i is chosen, it has the highest ratio of the (expected) gain over the size (expected) among all $q \in Q \setminus Q_{i-1}$. Also by adding all $o' \in Q^* \setminus Q_{i-1}$, the total benefit will increase to $\beta_d(\Gamma(Q^*))$. Therefore, the increase is at least $\beta_d(\Gamma(Q^*)) - \beta_d(\Gamma(Q_{i-1}))$. Let $Q^* \setminus Q_{i-1} = \{r_1, \ldots, r_m\}$. We know that

$$\sum_{j=1}^{m} \Delta_{r_j} \geq \beta_d(\Gamma(Q^*)) - \beta_d(\Gamma(Q_{i-1})).$$

Also we know that $\sum_{i=1}^{m} |\Gamma(r_i)| \leq L$. The goal is to show that, for q_i, the ratio of the expected gain over the size is at least

$$\gamma = \frac{\beta_d(\Gamma(Q^*)) - \beta_d(\Gamma(Q_{i-1}))}{L}.$$

By the choice of greedy algorithm, q_i has the highest ratio. If the inequality does not hold, it means that for all $r_1, \ldots, r_m$ the gain ratio is also less than γ. Lemma 10 showed that the expected partial gain ratio of a vertex is at least its complete gain ratio. So we have for each r_i, $\Delta(r_i) \leq \gamma \cdot |\Gamma(r_i)|$. Summing this bound over all the r_i's, we have

$$\sum_{i=1}^{m} \Delta(r_i) \leq \beta_d(\Gamma(Q^*)) - \beta_d(\Gamma(Q_{i-1})),$$

which is a contradiction. □

By using Lemma 11 and an induction as in [15], we can show the following (proof omitted).

LEMMA 12. *We have* $\beta_d(\Gamma(Q_i))$

$$\geq \left(1 - \prod_{j=1}^{i}\left(1 - \frac{\min(|\Gamma(q_i)|, L - \text{size}(Q_{i-1}))}{L}\right)\right)\beta_d(\Gamma(Q^*)).$$

We can now complete the proof of the theorem. We use Lemma 12. Let q_m be the last chosen query in the modified greedy solution. We know that

$$\sum_{i=1}^{m-1} |\Gamma(q_i)| + L - \text{size}(Q_{m-1}) = L.$$

Since

$$\prod_{j=1}^{m}\left(1 - \frac{\min(|(\Gamma(q_i)|, L - \text{size}(Q_{i-1})}{L}\right)$$

would be maximized when all the multiplicands have the same value, we have

$$1 - \prod_{j=1}^{i}\left(1 - \frac{\min(|\Gamma(q_i)|, L - \text{size}(Q_{i-1}))}{L}\right) \geq (1 - (1 - 1/m)^m) \geq 1 - 1/e.$$

This means that the expected gain from MODIFIED-GREEDY is at least $(1 - 1/e)$ of the expected gain from the optimal solution, excluding the last set. If $\tilde{Q}$ is the last chosen set, then the total maximum d-benefit in the optimal solution is at most $\beta_d(\Gamma(Q^*)) + \beta_d(\Gamma(\tilde{Q}))$. In MODIFIED-GREEDY, we compare the max d-benefit of the solution given by the pure greedy step and the solution that contains only one query (call it q') with the highest max d-benefit and return the one with the highest benefit. Since $\beta_d(\Gamma(q')) \geq \beta_d(\Gamma(\tilde{Q}))$, in the worst case the MODIFIED-GREEDY solution is at least $(1/2)(1 - 1/e)$ of the optimal solution.

Let $N_Q = \sum_{w \in W} \Gamma(w)$. If we assume the number of selected queries based on the greedy approach is K', the running time of this algorithm is $O(N_Q K' \log d)$ which in the worst case is $O(N_Q L \log d)$. Selecting $q \in Q$ with the largest $\beta_d(q)$ is $O(d \cdot N_Q)$. Assuming $d \leq L$, the MODIFIED-GREEDY algorithm runs in time $O(N_Q K \log d)$.

B. HARD BUDGET CONSTRAINTS

In this variant a rewrite (q, w) can be suggested only if none of the ads that can be covered by w is out of budget. We show that this hard budget constraint makes the problem as hard as independent set.

THEOREM 13. *The weighted version with hard budget constraint is as hard as independent set.*

PROOF. Consider an instance $G = (V, E)$ of the independent set problem. We can create an instance of our problem by creating a query for each vertex and an ad for each edge. Now, the query is connected to an ad if the corresponding edge was adjacent to that vertex. Also the weight of the (query, ad) pairs is set in a way that the total covered weight for each query is exactly one. Now if we place the restriction of 1 on the number of times each ad can be covered, the problem is equivalent to finding the maximum size independent set in the original graph. □

C. OPTIMAL FRACTIONAL AD ALLOCATOR

In this model we assume that the optimal ad allocator can break the traffic and suggest different d ads for each portion of the traffic. The only difference with the old approach is in the incremental oracle. The new algorithm works as follows.

Algorithm SOFT-BUDGET

For each $e = (q, q') \in E_Q$, compute how much it would increase the total benefit. Here, we call the current set of suggested rewrites $R \subset E_Q$. To compute the increase obtained by adding e we use the following procedure:

$R' = R \cup \{e\}$. Solve the following linear program:

$$\max \sum_{q \in Q}\sum_{a \in A} T(q) \cdot x_{a,q} \cdot \beta_q(a)$$

subject to

$$\sum_{(a,q) \in R'} x_{a,q} \leq d, \qquad \forall q \in Q$$

$$\sum_{(a,q) \in R'} x_{a,q} T(q) \leq L(a)$$

$$0 \leq x_{a,q} \leq 1, \qquad \forall a \in A, q \in Q.$$

Find the unmarked pair $e' = (q, q') \in E_Q$ that will return the maximum benefit in the above linear program.

Add e' to R if the number of chosen rewrites in R for q is less than K.

Mark e.

Repeat the above steps until each $e \in E_Q$ is marked.

THEOREM 14. *There exists an allocation of ads to query traffic that obtains the same amount of benefit that is obtained by the above LP solution.*

PROOF. For each $q \in Q$, choose d ads $A' = a_1, \ldots, a_d \subset A$ with the highest $x_{a_i,q}$ values. Initialize $\alpha = 0$. Start increasing α gradually and decreasing $x_{a_i,q}$ at the same rate. Allocate α fraction of $T(q)$ to ads in A'. Increase α until $\exists a' \in A \setminus A'$ with $x_{a',q} > \min_i(x_{a_i,q}) - \epsilon$. Update A'. (Select the ads with highest $x_{a',q}$.) For all the ads in $a' \in A \setminus A'$ we can see that $x_{a',q} \leq \min_i(x_{a_i,q}) - \epsilon$. This means that $x_{q,a} \leq 1 - \alpha, \forall q \in Q, a \in A$. Also we know that $\sum_{a \in A} x_{a,q} \geq d(1 - \alpha)$. This means we should still have d non-zero $x_{q,a}$ variables. □

Since the incremental oracle acts optimally, using the similar charging argument as before, we can show that the given algorithm is 1/2-approximation for the fractional ad allocator.