ABSTRACT

Title of dissertation: Combinatorial Problems in Online Advertising

Azarakhsh Malekian, Doctor of Philosophy, 2009

Dissertation directed by: Professor Samir Khuller
Department of Computer Science

Electronic commerce or eCommerce refers to the process of buying and selling of goods and services over the Internet. In fact, the Internet has completely transformed traditional media based advertising so much so that billions of dollars of advertising revenue is now flowing to search companies such as Microsoft, Yahoo! and Google. In addition, the new advertising landscape has opened up the advertising industry to all players, big and small. However, this transformation has led to a host of new problems faced by the search companies as they make decisions about how much to charge for advertisements, whose ads to display to users, and how to maximize their revenue. In this thesis we focus on an entire suite of problems motivated by the central question of "Which advertisement to display to which user?".

Targeted advertisement happens when a user enters a relevant search query. The ads are usually displayed on the sides of the search result page. Internet advertising also takes place by displaying ads on the side of webpages with relevant content. While large advertisers (e.g. Coca Cola) pursue brand recognition by advertisement, small advertisers are happy with instant revenue as a result of a user following their ad and performing a desired action (e.g. making a purchase). Therefore, small advertisers are often happy to get any ad slot related to their ad while large advertisers prefer contracts that will guarantee that their ads will be delivered to enough number of desired users. We first focus on two problems that come up in the context of small advertisers.

The first problem we consider deals with the allocation of ads to slots considering the fact that users enter search queries over a period of time, and as a result the slots become available gradually. We use a greedy method for allocation and show that the online ad allocation problem with a fixed distribution of queries over time can be modeled as maximizing a continuous non-decreasing submodular sequence function for which we can guarantee a solution with a factor of at least $(1-1/e)$ of the optimal.

The second problem we consider is *query rewriting* problem in the context of keyword advertisement. This problem can be posed as a family of graph covering problems to maximize profit. We obtain constant-factor approximation algorithms for these covering problems under two sets 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 due to rewrites.

We next consider two problems related to premium customers, who need guaranteed delivery of a large number of ads for the purpose of brand recognition and would require signing a contract. In this context, we consider the allocation problem with the objective of maximizing either revenue or fairness.

The problems considered in this thesis address just a few of the current challenges in e-Commerce and Internet Advertising. There are many interesting new problems arising in this field as the technology evolves and online-connectivity through interactive media and the internet become ubiquitous. We believe that this is one of the areas that will continue to receive greater attention by researchers in the near future.

Combinatorial Problems in Online Advertising

by

Azarakhsh Malekian

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2009

Advisory Committee:
Professor Samir Khuller, Chair/Advisor
Professor Lawrence Ausubel
Professor David Mount
Professor Neil Spring
Professor Aravind Srinivasan

Dedication

This thesis is dedicated to my parents Mahindokht and Abdolali. I would not have been able to accomplish this without their endless love and earnest support. I owe them for all of my achievements

Acknowledgments

This thesis could not have been accomplished without the assistance of many teachers, colleagues and friends.

I would first like to thank my advisor, Professor Samir Khuller for his guidance and support throughout my graduate career and during the completion of this thesis. Samir let me have the freedom to pursue my interests and work on problems that were more appealing to me in algorithm design. At the same time, he helped me clarify my ideas and patiently advised me on how to improve my skills. I am also extremely grateful to the members of my committee, Professor Lawrence Ausubel, Professor David Mount, Professor Neil Spring and Professor Aravind Srinivasan for taking the time to be in my thesis committee and for giving me very useful comments. I am in debt to all of them for their advices during the writing of my thesis and throughout the years I was in graduate school.

I was lucky to be a part of a friendly department. I should thank all my friends, staff and officemates for providing such a nice atmosphere. Special thanks to my friend and collaborator Saeed with whom we worked on a lot of problems and learned a lot. Also I should thank my colleagues Julian Mestre, Yoo Ah kim and Mohammad Toossi for sharing their insightful ideas and Co-authoring papers with me. I should also thank Professor Amol Deshpande for sharing his ideas and co-authoring a paper with me. I owe my gratitude to kind staff of computer science department, specially Jennifer Story and Fatima Bangura for all their help and support.

During my years at UMD, I visited Yahoo! and Microsoft Research. These visits opened up my mind into new fascinating problems and gave me the opportunity to collaborate with many brilliant researchers. Special thanks to Ravi Kumar and Mohammad Mahdian, Grant Want and Chi Chao Chang and Erik Vee from Yahoo! and Adam Kalai, Christian Borgs and Jennifer Chayes from Microsoft Research. I also should thank Professor Jason Hartline for all his useful advices.

Studying all these years in graduate school, far from home, would not have been possible without the support and sympathy of all my friends. Many thanks to all my dear friends specially my dear Baharak.

The work in this dissertation was done under the support of the NSF grant CCF-0430650 which I am duly grateful.

Before coming to University of Maryland, I spent my undergrad at Sharif University of Technology. I would like to thank Professor Mohammad Ghodsi who introduced me theoretical computer science and helped me build up my background.

Last but not the least, my dear parents and family receive my most heartful gratitude for their endless love. I want to thank my sister Azadeh and my brother Roozbeh for their support and love all these years.

This thesis is dedicated to my parents Mahindokht and Abdolali whom I cannot think of a proper way of thanking them compared to all their endless support and love. I owe them for all of my achievements.

Table of Contents

List of Tables

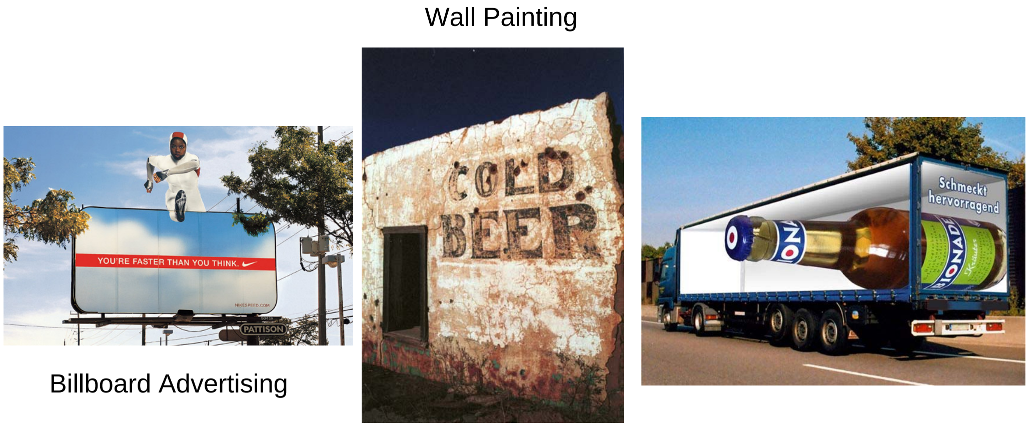
List of Figures

Chapter 1
Introduction

Advertising is a key factor in marketing; since it benefits both sellers and buyers. From the seller's point of view, advertising is a means of introducing new products to the world and describing their new features and also what makes them unique compared to other products available in the market. At the same time, it can help buyers decide which products match their needs the best, since they learn about more options to choose from. For example, consider a company that has an opening for a position. By advertising the position, the company gets more interested applicants and can possibly find a better match for its position. Furthermore, advertising helps applicants not to miss an opportunity simply by being unaware of all relevant open positions. Advertising is usually done many forms of media. It can be in the form of displaying advertisement as a wall painting, a TV commercial, a newspaper ad, etc. (Figure 1.1).

In the past few years, advertising over the Internet has overtaken many of the "traditional" media outlets because of the shift toward people's usage of the Internet for entertainment, communication, and as the main resource for information. In addition, the new features of advertising over the Internet make it more and more desirable for advertisers. First of all, Internet advertising lets the advertisers show their ad to targeted users of a random sample of the population. One way of targeted advertising instead, is by showing the ad only to a subset of users who are searching for a specific keyword. For example a travel agency would rather to show their ad to a person who is searching for something related to traveling such as plane tickets and hotels. instead of a random user. Most search engine companies are showing ads based on search terms we enter into the search engine. Another way is by targeting ads based on factors such as gender, age, geographic location and other general factors.

Another advantage of Internet advertising is cost. Considering the cost against the reach of interested audience, it is relatively cheaper than other media. The nature of the medium allows consumers to research and purchase products and services at their own convenience. At the same time, it is much easier and cheaper to collect accurate statistical data on the effectiveness of the advertisement over the audience and it can help advertisers to find the best way of advertising and best subgroup of the audience to target. The advertisers can choose different methods for paying for ads: pay per impression in which an advertiser will be charged when her ad is shown to a user, pay per click in which an advertiser is charged only when a user



Wall Painting

Billboard Advertising

Figure 1.1: Traditional methods for advertising

clicks on her ad, or pay per action in which the advertiser is charged only when a user performs a desired action (e.g., purchases the product). By choosing pay per click or pay per action, advertisers can quickly measure the effectiveness by counting the clicks or actions. However such measurements cannot be achieved through mass media or billboard advertising, where an individual will at best be interested, and then decide to obtain more information about the product at a later time.

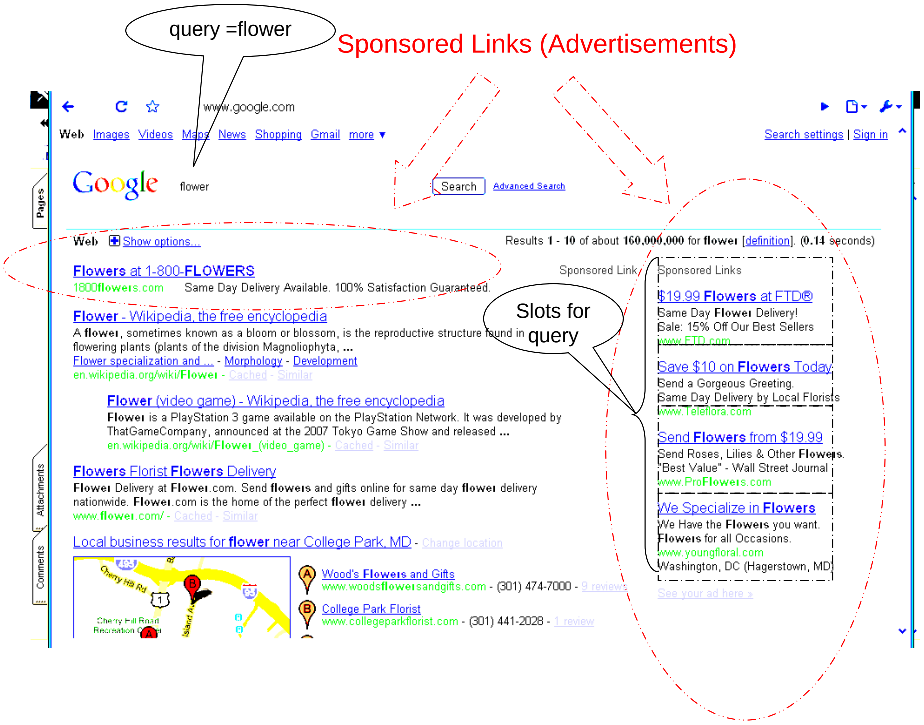
The focus of this dissertation is online targeted advertising, and some of the optimization problems that Internet advertising companies face. We consider problems with the main objective of profit maximization for these companies as well as consumer satisfaction. What makes these problems unique from the combinatorial perspective is the volume of trades that can happen simultaneously. Furthermore, the speed of running the actions, the number of participants in each trade and the anonymity of the participants are other reasons for making the problems distinctive. Because of these special properties, the usual solutions might not be feasible or even a good solution anymore.

From the perspective of the Internet advertising companies, such as Google or Yahoo!, online advertising can be categorized to two main classes:

- Non-Guaranteed Delivery
- Guaranteed Delivery

Non-Guaranteed Delivery is the traditional way of online advertising over the Internet. In this variant, advertisers ask to show their ad for a slot, for queries searched by Internet users one by one (query is a set of keywords that a user is interested in). The main property of this variant is that the allocation and payment is computed online for each query separately (Figure 1.2).

Various auction formats might be used for doing the allocation and charging the advertisers. Almost all of them have the following format: Advertisers submit bids to the auction to win a slot on a search result page. The auction has the property that larger bids will have a higher chance of getting top slots. Each advertiser is charged based on the number of clicks by users on their ads and the position of the slot they get. The bid value submitted by each advertiser defines their valuation for a single click. Each advertiser can also submit their daily or monthly budget and ask the search engine to include them in the upcoming auctions for their desired keyword as long as they are within their budget limit. As the most popular method, we describe the *Generalized Second Price* auction, currently used in Google AdWords and Yahoo! Search Marketing. In this method, the search engine (e.g., Google) runs a separate auction for each incoming query. This is an internal process and advertisers submit their keyword, their bid and budget only once to the search engine.


query =flower
Sponsored Links (Advertisements)
Slots for query
www.google.com
Web Images Videos Maps News Shopping Gmail more ▾
Search settings | Sign in
Google
flower
Search
Advanced Search
Pages
Attachments
Comments

Web ➕ Show options...

Results **1** - **10** of about **160,000,000** for **flower** [definition]. (**0.14** seconds)

Sponsored Link

Flowers at 1-800-**FLOWERS**
1800**flowers**.com Same Day Delivery Available. 100% Satisfaction Guaranteed.

Flower - Wikipedia, the free encyclopedia
A **flower**, sometimes known as a bloom or blossom, is the reproductive structure found in flowering plants (plants of the division Magnoliophyta, **...**
Flower specialization and ... - Morphology - Development
en.wikipedia.org/wiki/**Flower** - Cached - Similar

Flower (video game) - Wikipedia, the free encyclopedia
Flower is a PlayStation 3 game available on the PlayStation Network. It was developed by ThatGameCompany, announced at the 2007 Tokyo Game Show and released **...**
en.wikipedia.org/wiki/**Flower**_(video_game) - Cached - Similar

Flowers Florist **Flowers** Delivery
Flower Delivery at **Flower**.com. Send **flowers** and gifts online for same day **flower** delivery nationwide. **Flower**.com is the home of the perfect **flower** delivery **...**
www.**flower**.com/ - Cached - Similar

Local business results for **flower** near College Park, MD - Change location

A Wood's **Flowers** and Gifts
www.woods**flowers**andgifts.com - (301) 474-7000 - 9 reviews

B College Park Florist
www.collegeparkflorist.com - (301) 441-2028 - 1 review

Sponsored Links

$19.99 **Flowers** at FTD®
Same Day **Flower** Delivery!
Sale: 15% Off Our Best Sellers
www.FTD.com

Save $10 on **Flowers** Today
Send a Gorgeous Greeting.
Same Day Delivery by Local Florists
www.Teleflora.com

Send **Flowers** from $19.99
Send Roses, Lilies & Other **Flowers**.
"Best Value" - Wall Street Journal
www.Pro**Flowers**.com

We Specialize in **Flowers**
We Have the **Flowers** you want.
Flowers for all Occasions.
www.youngfloral.com
Washington, DC (Hagerstown, MD)

See your ad here »

Figure 1.2: Advertisement for the search query "flower"

The search engine assigns a weight to each (advertiser, keyword) pair based on the expected relevance. The weight is independent of the advertiser's submitted bid. In the auction, advertisers are ranked based on the product of their weight and their bid. Next, each advertiser is charged equal to the minimum bid that she could submit and still win the same slot.

Another category of online advertising that is more popular for banner advertisement is **Guaranteed Delivery Advertisement**. In the past, most banner advertisements were sold through negotiation between advertisers and publishers (web sites) that could result in inefficient outcomes and higher costs. In recent years, there has been some effort on automating this process, called *Guaranteed Delivery*. *Guaranteed Delivery* was introduced and implemented in major search engines like Yahoo! and Google for dealing with banner advertising. In guaranteed delivery, each advertiser specifies a collection of host web pages (publishers) based on the relevance to her product. They also specify the desired total number of impressions on these pages, and a maximum price that they are willing to pay per impression. The system (search engine) will then select a subset of these advertisers as winners and sign a contract with them. The search engine maps each winner to a set of impressions on pages from her desired collection. The distinguishing feature of guaranteed delivery as opposed to non-guaranteed delivery mechanism is that, the system has to satisfy a minimum required demand of the winning advertisers (or in another variant, the system should pay extra penalty as monetary payment in case it can not satisfy all the winning advertiser's demand). Such guarantees are essential in markets in which main purpose of advertising is to develop brand recognition. In addition, the allocation and payments are query independent and fixed for each contract. The winners are chosen as the advertisers and queries enter the system.

In this dissertation, we consider two problems from each class. We start by *Non-Guaranteed Delivery*. In this class we will consider the following problems:

- Online Ad Allocation Problem
- Query Rewriting Problem

There are different ways for matching advertisers and incoming queries. The main objective in both problems is to maximize the revenue of the search engine and increase the relevance of ads to queries. By increasing the relevance of the ads to queries, in the short term, users will click on relevant ads and generate revenue for the search engine. In the long term, users will recall that ads were relevant to them and thus continue to click on them. In both long and short term, this creates revenue for the publishers and generates leads for the advertisers.

Formally, the problem descriptions are as follows:

Online ad Allocation: The first problem that we study in *Non-Guaranteed Delivery* is *Online Ad Allocation* problem. Suppose that for each pair of advertiser and keyword, we know the expected payment by the advertiser for showing her ad for that keyword (this can be computed using historical data). Taking into account the budget limit of each advertiser and also the number of available impressions for each keyword, how should an Internet company assign advertisements for keywords to maximize their profit? We consider this problem in a combinatorial setting and give more details on this problem later.

Query Rewriting Problem: Next problem in this category is called the *Query Rewriting Problem.* Although advertisers bid on keywords, a relevant ad for a given query may not necessarily exist among the set of ads that have bid on that query. Indeed, the set could be empty, even though a relevant ad may exist. For instance, an ad for the keyword "wedding band" may be appropriate for the query "engagement ring". 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. However traditional existing methods for query rewriting in informational retrieval cannot be applied to keyword advertising because of system limitations and user and advertiser requirements. We describe the challenges and problem formulation later on in this chapter. The objective of this problem is again revenue maximization and increasing the relevance of queries to the selected ads.

In the context of *Guaranteed Delivery* we look at the following problems:

- Fair Contract Allocation
- Online Allocation of Display Advertisement

In the second problem, we consider the objective of search engine revenue maximization, but this time in the *Guaranteed Delivery* setting. In the first problem, however, the main focus is on advertiser satisfaction. As we argued earlier, in the long term, this increases the profit.

Fair Contract Allocation: In the *Guaranteed Delivery* setting, advertisers and users are mediated by a publisher (e.g., an online newspaper, a search engine, etc.). The advertiser buys a contract for a certain number of impressions (user visits to the publisher's page) and declares interest in a subset of user types called buckets (e.g., "girl, New York"). The goal of the publisher is to satisfy the demands by placing an ad from the advertiser on the web page visited by a user, if the user (i.e., the impression) belongs to the advertiser's bucket. Fair contract allocation is the problem of choosing the most balanced allocation among all the feasible allocations (a feasible allocation meets all the advertiser imposed constraints). If the publisher only needs to satisfy the contract's requirement, assigning a sufficient number of impressions to an advertiser as long as they belong to the advertiser's bucket is a feasible solution. Unfortunately, such an assignment can be unfair and unrewarding to the advertiser. To illustrate this, consider a girl's toy store whose poorly specified bucket reads "new York, females." If the publisher unintentionally serves up only impressions from middle-aged women in New York for this advertiser, then the latter is left irate! (One might blame the advertiser for not specifying the bucket precisely, as say, "new york, females, young adults", but in practice, it is never fully possible for any advertiser to specify the desired buckets to the finest conceivable granularity.) There is a tacit assumption by the advertisers that the impressions assigned are as "fair" as possible. In the *Fair Contract Allocation* problem we are trying to solve this problem: Given a set of impressions (i.e., the supply) and contracts (with demands and buckets), how do we find a feasible assignment of impressions to contracts that is as *fair* as possible? Answering this question involves formulating what fairness precisely means in this context. Given the large number of advertisers (typically, in the hundreds of thousands) and the astronomical number of impressions (typically, in the hundreds of millions) in an online setting, we insist on a solution that is *efficient*, in both time and space, and that yields insights into the structure of the allocation problem itself. In particular, we desire an allocation algorithm that is practical and combinatorial and whose allocation can be stored *succinctly*, ideally, using space linear in the number of impressions and contracts as opposed to the naive quadratic storage solution. Of course, this succinct representation should let us reconstruct the allocation along every edge in a time-efficient manner.

Online Allocation of Display Advertisements: As we said before, in *Guaranteed Delivery* advertisement, advertisers arrive online and each advertiser asks for a certain number of impression for a fixed period of time. Furthermore, the number of available impressions per day is bounded and can be estimated based on historical data. Assuming that after accepting an advertiser's request, the search engine is obliged to satisfy its whole demand, the problem that the search engine needs to solve is to decide which advertisers to accept in order to maximize the profit. This has to be done in an online fashion as the advertisers arrive. Later in this section we describe the problem in more details and also discuss our results for this problem.

Until now, we gave an overview of the problems that will be considered in this dissertation and the high level motivation for each problem. In the following section we formulate each problem in more detail.

1.1 Problem Formulation

In this section, we present a more detailed formulation for each problem and at the end of each part, a summary of our results for that problem.

1.1.1 Online Ad Allocation Problem

Assuming that for each query, the search engine can show d ads simultaneously, the online ad allocation problem can be defined as follows:

Problem Definition Let M be the set of ads and N the set of distinct query types (keywords), with $|M| = m$ and $|N| = n$. Let p_{ij} be the expected payment of the advertiser to the search engine for showing ad i for a query of type j. The expected payment may be computed based on the relevance of the ad to the keyword, the bid of the advertiser for that keyword and possibly other parameters (we assume it is given to us). Each ad i has a budget B_i for a given time period T. The goal is to assign incoming queries to ads as they arrive in a way that maximizes the profit of the search engine in a given time period. Here we make the assumption that the types of incoming queries are i.i.d random variables drawn from a fixed but possibly unknown distribution $\{q_j\}$ where q_j is the probability of a query being of type j ($\sum_j q_j = 1$). Also we assume that the expected payments (p_{ij}) are small compared to budgets (B_i). Note that in a sequence of r queries the expected number of queries of type j is rq_j. We would like to express this as a function of time so we define a virtual time, based on the number of queries that have arrived so far. In terms of our virtual time the expected number of queries arriving in a period of length Δt of type j is $\Delta t q_j$. Let T be the end of the time period in terms of the virtual time. So the problem is to find an allocation that maximizes the revenue of the search engine in time $[0, T)$.

Challenges Consider the offline version of the problem in which we know the queries in advance. The problem can be solved using LP rounding and get a solution close to the optimal (with the approximation ratio very close to 1 assuming that

$p_{ij} \ll B_i$). Now consider the online version of the problem in which we know the distribution q_j. Again LP rounding can be used to get a solution with an expected value very close to the optimal expected value. However, there are two issues with the online version. The first one is that we cannot use LP if we do not know the distributions and the second one is that due to the huge size of the input, it is not possible to use LP rounding in practice for this problem.

Our Result We consider a greedy algorithm and we show that its expected profit is at least $1 - \frac{1}{e}$ of the optimal. One important advantage of the greedy algorithm is that it does not depend on the distribution of the queries, and it is easy and fast to compute in real time even with huge input data. As such it is being used in practice. The suggested solution framework is more general and can be used for solving other online maximization problems for which the objective function satisfies the sequence submodularity property with prior distribution assumption.

1.1.2 Query Rewriting Problem

In the *Query Rewriting problem*, the objective is to suggest a set of queries that are related to a given query. The suggested list of queries should help us collect a good set of ads for each keyword. At the same time, we should take into account the constraints that are imposed by the system. We start this section by investigating the challenges that we need to deal with when solving query rewriting for keyword based advertising before giving the exact problem formulation.

Challenges In general, we have the following constraints:

1. constraints on the number of rewrites per query — typically the number of rewrites is 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 contribute to the revenue.

Next we describe different variants of query rewriting formulation that we study in this dissertation.

Problem Formulation 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 (Figure 1.3). 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. Most of the variants of this problem are NP-hard in general (by a reduction from Max K-coverage that is known to be NP-hard [38]) 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.

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.

Our Results For the *cardinality version*, we propose 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 *weighted version*, we propose another greedy algorithm with an approximation ratio of $1 - \frac{1}{e^{1-\frac{1}{e}}}$. To analyze this variant we use the same method that we used for online ad allocation problem.

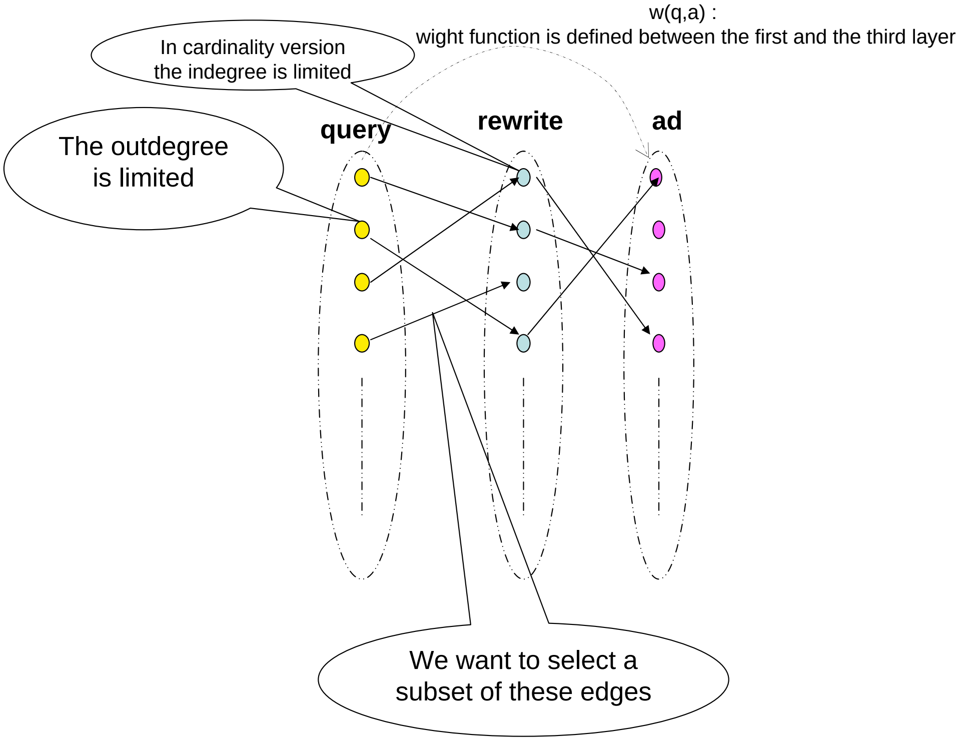


Figure 1.3: Three layer graph containing queries, rewrites and ads

We have also conducted 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.

1.1.3 Online Ad allocation for Display advertisement

As discussed in the online ad allocation for *Non-Guaranteed Delivery*, a search engine chooses a subset of contracts to maximize its revenue. However as in the rest of the problems, there are some challenges that we need to deal with:

Challenges Advertisers ask for a certain number of impressions for a certain period of time. The demand is unknown ahead of the time since they are arriving online. Since we are in the *Guaranteed Delivery Setting*, the demand of an advertiser should be completely satisfied or the advertiser will not pay anything to the search engine. Furthermore, as in the other problems the number of available impressions per day is limited for each keyword.

Problem Formulation The basic model is as follows: Advertisers arrive online and request a contract. Upon arrival, advertiser k reveals all her information including the number of impressions she is interested in per day and the start and end of the time period that she wants her ads to be displayed. The payment is according to the total number of impressions that will be allocated to an advertiser (if and only if her total demand will be satisfied). The publisher then decides on the spot to accept or reject the contract. We also assume that the total number of available impressions per day is fixed. Contracts can be dropped without penalty (i.e., the publisher can accept a contract and drop it later if a better one arrives). However in this case the publisher cannot charge the advertiser anything if the contract is dropped. The goal is to maximize social welfare (or revenue) under an *adversarial arrival* of advertisers.

Our Results Under these assumptions, we will show that for an arbitrary sequence of advertisers, there is no online algorithm that can have a worst case ratio smaller than n where n is the total number of available impressions per day. We then present an online greedy algorithm with a worst case analysis and we show that the obtained profit from our algorithm is only a constant factor worse than the best possible online algorithm. Since the lower bound on the best possible algorithm in the adversary setting is quite pessimistic, we simulated the performance of our greedy algorithm on actual data from Yahoo!'s display advertisement business. We found that our algorithm performs very well in practice as the observed performance was quite close to the optimal as opposed to the worst case lower bound.

1.1.4 Fair Contract Allocation:

When advertisers purchase a guaranteed contract, they often define a set of publishers as their possible future hosts and ask for a minimum total number of impressions for these pages. Assuming that the search engine has already decided which set of advertisers to accept, there are many different feasible allocations that satisfy all the demands. The following is some of the challenges that we face:

Challenges One way of allocating impressions to contracts is to allocate the cheapest impressions to the contracts and leave the rest, for being auctioned between advertisers interested in non guaranteed delivery. However the problem with this strategy is that guaranteed sales are premium products and as a result, assigning the cheapest impressions is not a good strategy. The implicit assumption of the advertisers is that they will receive a "fair" sample of their desired buckets. A more desirable approach is to try to assign the most representative sample of the desirable impression in each contract to the advertiser. By representative, we mean the distribution of the type of the ads in the sample, should represent the distribution of the ads in the original set. However, it may not be possible to assign a fully representative sample to all the contracts because the total number of available impressions for each keyword is limited. The situation is even worse when the interesting keyword is very valuable. Another constraint is that the number of advertisers and the number of impressions are quite large. In an online setting, we need to find solution which is *efficient* in both time and space.

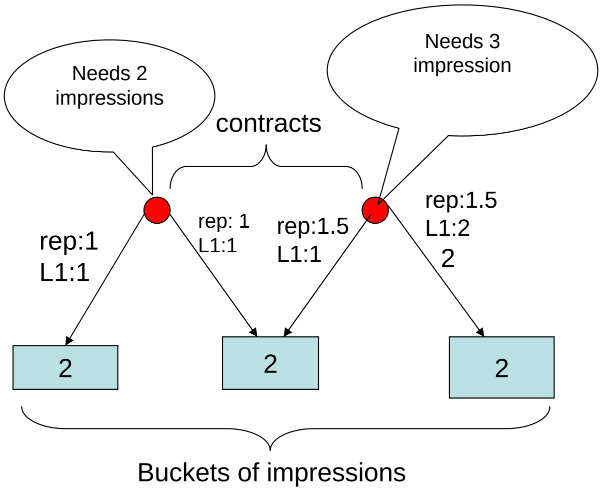
Problem Formulation The problem can be formulated as follows: Given a set of impressions (i.e., the supply) and a set of contracts with their desired number of impressions and the types of impressions they are interested in, find a feasible assignment of impressions to contracts so that for each contract, its demand is completely satisfied and at the same time, the allocation is as *fair* and representative as possible? Answering this question involves formulating what fairness, precisely means in this context. Given the large number of advertisers (typically, in the hundreds of thousands) and the astronomical number of impressions (typically, in the hundreds of millions) in an online setting, we desire an allocation algorithm that is practical and whose allocation can be stored *succinctly*, ideally, using space linear in the number of impressions and contracts as opposed to the naive quadratic storage. Of course, this succinct representation should let us reconstruct the complete allocation along every pair of contracts and impression buckets in a time-efficient manner. The price of each impression is computed based on historical data.

Our Results We consider the general problem of fair allocation in a bipartite supply-demand setting. Our formulation is combinatorial and captures the notion of deviation from fairness by a natural and general form of a penalty function. While this formulation admits a convex programming solution (assuming the penalty function is convex), it is undesirable in practice because of performance considerations and therefore we seek more efficient solutions. For the case of L_1 penalty functions we obtain a simple combinatorial algorithm for the fair allocation problem. By L_1 penalty function, we mean that we want to find a feasible allocation of impressions to contracts while minimizing the absolute distance of this allocation from the most representative allocation for each contract (Figure 1.4).

Our solution is based on solving a min-cost flow problem on a bipartite graph, which can be done very efficiently. By using a powerful dual formulation stemming from our combinatorial treatment of allocation and constraining the flow to be unique in a certain way, we also show how to precompute and store a *linear* amount of information such that the allocation along *any* edge in the bipartite graph can be approximately answered in *constant* time, under mild assumptions on the input instances. This space-efficient reconstruction method might be of independent interest in contexts beyond fair allocation.

We also prove two additional properties of our formulation. First is robustness, where we show how to upper bound the performance loss when the supply estimates are only approximately known. Second is extensibility, where we show an even simpler greedy approximation algorithm when some of the demand constraints are relaxed.

Finally, we extend our combinatorial solution to any convex function. This involves solving a convex cost flow, which once again is more efficient than solving a general convex program.



rep: completely representative allocation
L1: The feasible solution selected by using L1 metric

Figure 1.4: It is not feasible to give each contract its completely representative allocation

1.2 Roadmap

This dissertation is organized as follows. In chapter 2, we present the related work on all problems that will be discussed in this dissertation. In chapter 3, we will present the analysis of greedy method for *Online Ad Allocation* problem and also present a general method for analyzing greedy methods for online problems with some specified properties [5]. Then, we will present different models and solutions for each model for *Query Rewriting* problem in chapter 4 [40]. In chapter 5, we turn the focus to *Guaranteed Delivery* advertisement and consider the fair allocation problem and present an impact solution for the solution that is closest (minimum L_1 distance function) to the completely representative allocation for each advertiser [4]. Finally, in chapter 6, we consider online ad allocation problem again. However, this time we look at *Guaranteed Delivery* [3]. We conclude with an overview of the future work in chapter 7. In Figure 1.5, the main structure of this dissertation is presented.

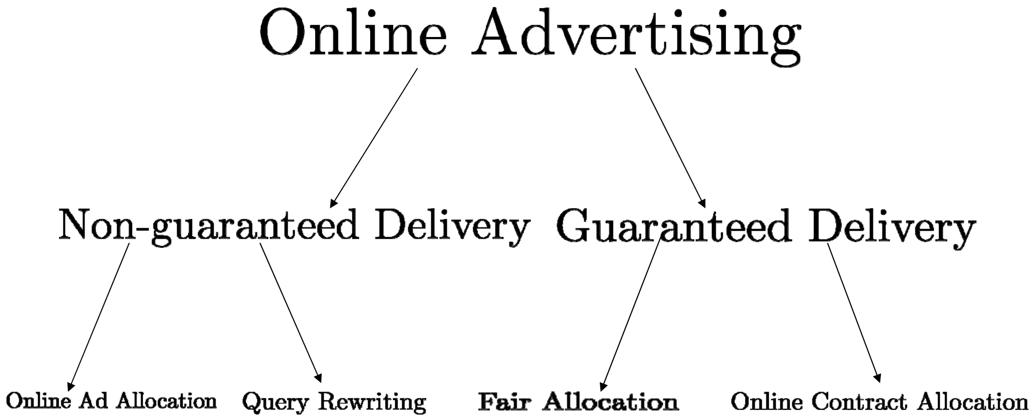


Figure 1.5: Organization

Chapter 2
Background and Related Work

The solutions and analysis for both problems in the *Non-guaranteed delivery* category are heavily based on the properties of a class of functions called submodular functions. We start this chapter by introducing this property and survey existing results on this topic. After that, we present related work for each problem.

2.1 Submodular Functions

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

Both maximization and minimization of submodular functions have been studied in recent years due to its application to combinatorial auctions (e.g., the submodular welfare problem [39, 37]), generalized assignment problems [24], etc. Minimizing a submodular function given by a value oracle can be done in polynomial time. However, this is not true for maximizing submodular functions ([32],[31]). An algorithm is called *efficient* if its running time is polynomial in the size of its input. Submodular maximization problems are typically in the class of NP-hard problems. When dealing with an NP-hard problem, in order to get an efficient algorithm, we need to relax the optimality of the objective function. An algorithm A for a given maximization problem (F, c) is called an α approximation if it is efficient and the objective value for the solution that is obtained by A for any instance, is at least α times of the objective of the optimal solution. The natural approximation algorithm that is suggested for maximizing a non-decreasing submodular function is a greedy approach. The greedy approach works as follows: Start with an empty set and iteratively build the solution. At each iteration, select the element with the highest incremental profit to the current solution and add it to the current solution. Nemhauser et al ([45]) showed that greedy approach gives a $1 - \frac{1}{e}$ approximation for maximizing non-decreasing submodular functions with a cardinality constraint. Nemhauser and Wolsey, also showed that in general a greedy approach gives a $1 - \frac{1}{e}$-approximation for maximizing a non-decreasing submodular function over a uniform matroid. Nemhauser, Wolsey, and Fisher [46] 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 [29] generalized both these results and showed that if an α-approximate incremental oracle is available, then the greedy solution is a $1 - \frac{1}{e^{1/\alpha}}$ approximation for maximizing a non-decreasing submodular function over a uniform matroid and an $\alpha M + 1$ approximation for the intersection of M matroids. In some recent work, Feige, Mirrokni and Vondrak designed different constant factor approximation for maximizing non-negative submodular functions without any monotonicity assumptions (They present a 1/3 approximation algorithm based on local search and a 2/5 approximation based on randomized rounding.)[23]. furthermore, Vondrak [**?**], presented a general method to derive inapproximability results for submodular functions.

In the rest of this chapter, we present the previous results on each of the problems that will be studied in this thesis.

2.2 Online Allocation Problem

There is a considerable amount of literature on auctions for online advertising in the economics and computer science community. In the online allocation problem, the goal is to decide which ads to show for each incoming query so that the the obtained profit from the advertisers is maximized. Several papers have studied this problem ([43], [18]). Mehta et al [43] presented a deterministic algorithm with a competitive ratio of $(1 - \frac{1}{e})$ in the worst case model. It can be shown that the competitive ratio for the greedy algorithm is $\frac{1}{2}$ in the worst case analysis. Later, Goel et al [27] showed that the competitive ratio of the greedy approach in the random permutation model as well as the i.i.d model is $(1 - \frac{1}{e})$ and in fact, the analysis is tight. Their proof is partly based on the techniques used for the online bipartite matching problem[36].

The offline variant of ad allocation problem is NP-hard as well[6, 24]. The first approximation algorithm for the problem was given by Garg, Kumar and Pandit[25] who gave a $\frac{2}{1+\sqrt{5}} = .618$ factor approximation. Andelman and Mansour [6] improved the factor to $1 - \frac{1}{e}$ approximation. Azar et.al. [8] obtained a $\frac{2}{3}$-factor for the general offline ad allocation problem. All these works are based on LP relaxation. Finally, Goel and Chakrabarty ([15]) and Srinivasan ([49]) independently gave a $\frac{3}{4}$ approximation for this problem and the methods were based on LP rounding. In addition, Chakrabarty and Goel showed that it is NP hard to approximate the offline variant to a factor better than $\frac{15}{16}$[15].

2.3 Query Rewriting

Query rewriting problems have been studied in the context of *Informational Retrieval* for a long time [35, 52, 53]. 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 [35] rely on user query reformulation sessions to precompute 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 [52, 53] have improved its performance using click logs and active learning techniques and with additional features such as web search results page co-occurrence.

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 [11] apply agglomerative clustering techniques to bipartite click graphs using a simple set overlap distance function. Antonellis et al [7] build upon Simrank [34], 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 Singular Value Decomposition [17]. It constructs a large matrix of term-document association data, forming a semantic space wherein terms and documents that are closely associated are placed near one another. Singular value Decomposition (SVD), allows the rearrangement of the space to reflect the major associative patterns. These based methods are computationally expensive. The algorithms presented in this thesis complement all these techniques by optimizing the selection of the rewrites by taking the “ad benefit” into account.

2.4 Display Advertisement

The last two problems considered in this thesis are related to display advertisement. We start by giving a summary of existing work in this area. Most of the recent literature related to online advertising is focused on studying slot ad auctions from the game theoretic perspective [20]. There has been some recent work on display advertisement and guaranteed delivery. Feige et al. [22], studied guaranteed delivery for display advertisement with penalties. In this model, for each accepted contract, either the whole demand requested in the contract should be satisfied or the search engine will pay extra penalties for the unsatisfied portion of the demand. They showed that there is no constant approximation for their problem and present a bicriteria algorithm. Furthermore, they proved a structural approximation result for the adaptive greedy algorithm. The problem of advanced booking with costly cancelation have also been studied from a game-theoretic point of view[16, 9].

2.5 Fair Allocation

Vee, Vassilvitskii, and Shanmugasundaram in [50] and also McAfee and Papineni [42] first studied the online allocation with forecast problem, where given an approximation of the online supply, the goal is to create an efficiently reconstructible plan for performing some form of fair allocation. They focus on the efficiency and sampling aspect of the problem and consider only the *strictly* convex version, which makes it amenable to using fixed point criteria such as KKT conditions for non-linear optimization. Ghosh et al. [26] studied the problem of representative allocation for display advertising when there are both spot markets and guaranteed contracts; they propose a solution where guaranteed contracts are implemented by randomized bidding in spot markets. Our solution is mainly based on network flow and its dual. There is a large amount of literature on the network flow problem (e.g., [2]). The closest work to our method is the push-relabel algorithm of Goldberg and Tarjan [28]; they introduced a method for computing the maximum flow problem without using augmenting paths. The reconstruction of the min-cost flow instance is based on the dual variables of the min-cost flow solution and has some similarities to push-relabel algorithm. Primal-dual methods have been largely used as a tool to find approximation algorithms for various problems (e.g., [10, 1]). Recently, Devanur et al. [19] and Jain and Vazirani [33] used primal-dual methods and KKT conditions for solving market equilibria problems.

2.6 Online Resource Allocation

In the last problem studied in this thesis, we deal with the online allocation of ads for display advertisement. The problem is closely related to *Dynamic Storage Allocation.* An offline variant of this problem has been studied as well [13]. In *Dynamic Storage Allocation*, the objective is to pack axis aligned rectangles in a horizontal strip of minimum height by sliding the rectangles vertically but not horizontally. Another closely related problem is the *Resource Allocation Problem.* This problem is very similar in nature to *Dynamic Storage Allocation.* The difference is that in this problem instead of rectangles we have jobs and instead of the horizontal strip we have some constant amount of resources that are available for a period of time. Each job is supposed to be executed at a pre-specified period of time and each day it will consume some constant amount of resource. The objective is to find a feasible schedule of jobs so that the total revenue will be maximized. The problem that we need to solve is the online variant of *Resource Allocation*. To the best of our knowledge, our algorithm is the first online algorithm for this problem. Here we present the setting from the work by Woeginger and some of its results[51]. In this work, the setting is as follows:

- Only one impression per round is available.
- All of the supply is exhausted by one advertiser at any given time.
- Payment is received once the contract is fully completed.
- Contracts can be dropped without penalty.

The goal is to maximize social welfare (or revenue) under an *adversarial arrival* of advertisers. This problem is called *Online Interval Scheduling* as well. It is known that the general case of the problem does not have a competitive ratio.[51]. This motivated the search for special cases that define real practical problems and at the same time have a constant competitive ratio. Woeginger [51] defined three special cases:

C-benevolent: In this class, the weight(or profit) of each interval only depends on its length. Furthermore, the weight function is strictly monotonically increasing, continuous and convex function.

D-benevolent: In this class, the weight function is a monotonically non-increasing function.

unit interval: In this class, all the intervals have unit length.

Woeinger showed that for all the above classes greedy approach is a 4 approximation algorithm and more interestingly he showed that the approximation ratio is tight for *C-benevolent* functions and *unit interval* class. [51]. For *D-benevolent* functions, he proved that the lower bound on the competitive ratio for any deterministic algorithm is 3. Seiden [48], presented a randomized algorithm for both classes of C-benevolent and D-benevolent functions with a competitive ratio of 3.73206. Miyazawa and Erlebach [44], considered unit intervals and gave a randomized 3-competitive algorithm for the special case where the sequence of arriving intervals has monotonically non-decreasing weight, as a function of the arrival times. They also designed a lower bound of $\frac{5}{4}$ for C-benevolent functions. Finally, Epstein and Levin [21], gave a lower bound of 1.693 for any randomized algorithm for C-benevolent functions and a subset of D-benevolent functions, and gave a lower bound of 1.5 for a larger subset of D-benevolent functions. They also gave a randomized algorithm for this setting with the approximation ratio of 3.22745.

Chapter 3
Non-guaranteed Delivery: Online Ad Allocation Problem

In this chapter and the next chapter, the focus is on *Non-guaranteed Delivery Advertisement*. Non-guaranteed delivery is the traditional way of advertising in sponsored search. In this chapter we consider the following problem:

Online Ad Allocation Problem: Assuming that for each query, the search engine can show d ads simultaneously, the online ad allocation problem can be defined as follows: We have m ads and n distinct keywords (query types). Let M be the set of ads and N the set of query types. Let p_{ij} be the expected payment of the advertiser to the search engine for showing ad i for a query of type j. The expected payment may be computed based on the click-through rate of the ad (click through rate is obtained by dividing the number of users who clicked on an ad on a web page by the number of times the ad was delivered (impressions)), the relevance of the ad to the keyword, the bid of the advertiser for that keyword and possibly other parameters. Each ad i has a budget B_i for a given period T. The goal is to assign incoming queries to ads as they arrive in a way that maximizes the profit of the search engine in a given time period. Here we make the assumption that the types of incoming queries are i.i.d random variables drawn from a fixed but possibly unknown distribution q_j where q_j is the probability of a query being of type j ($\sum_j q_j = 1$). We assume that the expected payments (p_{ij}) are small compared to budgets (B_i). Note that in a sequence of r queries the expected number of queries of type j is rq_j. We would like to express this as a function of time so we define a virtual time, based on the number of queries that have arrived so far. In terms of our virtual time the expected number of queries arriving in a period of length Δt of type j is $\Delta t q_j$. Throughout the rest of this chapter we will omit the word "virtual" and always use "time" to refer to virtual time unless explicitly stated otherwise. Let T be the end of the time period in terms of the virtual time. So the problem is to find an allocation that maximizes the revenue of the search engine in time $[0, T)$.

As explained before we cannot use LP for two reasons:

- We do not know the distributions
- Due to the huge size of the input it is not possible to use LP rounding in practice for this problem.

In this chapter, we consider a greedy algorithm and we show that its expected performance is at least $1 - \frac{1}{e}$ of the optimal. One important advantage of the greedy algorithm is that it does not depend on the distribution of the queries, and it is easy and fast to compute in real time even with huge input data. As such it is being used in practice. The suggested solution framework is more general and can be used for solving online maximization problems for which the objective function satisfies the generalized submodularity, we call it sequence submodularity property, with prior distribution assumption. This property will be described in detail in the rest of this chapter and will be used as a tool for analyzing the problem in the next chapter as well. Note that the revenue that is generated as a result of displaying an ad for a query depends on wether the ad has enough budget left and therefore depends on the previous allocations. So the total revenue as a function of queries/allocations, depends on the order of queries/allocation and its domain is a sequence of queries/allocations. We show that this function is a sequence-submodular function.

3.1 Sequence Submodularity

In this section, we show how to extend the notion of *submodularity* to functions defined over sequences and analyze the greedy algorithm for maximizing such functions subject to a maximum length constraint on the solution sequence. We call this property, sequence submodularity. A large class of combinatorial problems can be modeled in this framework specially those that involve time. We will see two of its applications in this thesis.

Sequence submodularity can be defined as follows: Let S be a finite set and $u(H)$ be a real valued function defined over discrete or continuous sequences [1] of elements of S. We extend the notion of *submodularity* defined by Nemhauser [46] for set functions to the more general class of sequence function (a set function is a special case of a sequence function in which the order and frequency of elements does not matter). We consider the problem of finding H that maximizes $u(H)$ subject to $|H| \leq T$ for a given $T \in \mathbb{R}^+$ in which $|H|$ denotes the length of sequence H. What we will show in this section is that if u is non-decreasing, submodular and differentiable (only for continuous sequences), a greedy algorithm can find H that achieves $1 - \frac{1}{e}$ of the optimal maximum when we have access to an incremental oracle or $1 - \frac{1}{e^\alpha}$ when we have access to an approximation incremental oracle with the approximation factor of $\alpha \in [0, 1]$. As a general application of this framework, whenever we have a finite set of actions from which we can choose an action and run it for some arbitrary duration it

[1] A continuous sequence of length $T \in \mathbb{R}^+$ is a mapping from $[0, T)$ to S

can be modeled as a continuous sequence and the utility over time can be interpreted as a sequence function.

However in previous works, the submodularity property is defined only on functions over sets. Nevertheless, there are problems in which the goal is to choose a sequence of actions to maximize some utility function defined over that sequence. In some of these problems, the order of actions matters. Also, sometimes, the actions are continuous and each action is used for a specified duration. Such problems cannot be modeled using a submodular set function. The objective of this section is to characterize the conditions that are necessary for sequence functions so that we can obtain the same conclusions about the behavior of a greedy approach over this class of functions. A series of operations with the property that each operation is performed for some specified duration can be seen as a continuous sequence. What we will show is that if a sequence function has the three properties of being "non-decreasing", "sequence submodular" and "differentiable", a greedy approach always achieves a solution that is at least $(1-\frac{1}{e})$ of the optimal solution for the maximization problem subject to a constraint on the maximum length of the solution sequence. We will see in the following sections that the online ad allocation problem with a fixed distribution of keywords over time can be modeled as maximizing a continuous non-decreasing submodular sequence function for which we can guarantee that the greedy approach achieves at least $(1-\frac{1}{e})$ of the optimal

3.2 Definitions

We start by defining a set of notions that we will use in the rest of this chapter.

Discrete Sequence: Let S be a finite set. Any $A = (s_1, \cdots, s_k)$ where $k \in \mathbb{N} \cup \{0\}$ and $s_i \in S$, is called a discrete sequence of elements of S ($k = 0$ is the empty sequence). We also denote the set of all finite discrete sequences of S by $\mathbb{H}^D(S)$ which is formally defined as:

$$\mathbb{H}^D(S) = \{A = (s_1, \cdots, s_k) | k \in \mathbb{N} \cup \{0\}, s_i \in S\} \tag{3.2.1}$$

Notice that a discrete sequence actually defines a discrete function from $\{1, \cdots, k\}$ to S and any such discrete function can be represented using a discrete sequence. We denote the value of the function defined by discrete sequence A at point x by $A(x)$.

Continuous Sequence: Let S be a finite set. Any $A = ((s_1, \Delta t_1), \cdots, (s_k, \Delta t_k))$ where $k \in \mathbb{N} \cup \{0\}$ and $s_i \in S$ and $\Delta t_i \in \mathbb{R}^+$, is called a finite continuous sequence of elements of S. We also denote the set of all finite continuous sequences of S by $\mathbb{H}^C(S)$ which is formally defined as:

$$\begin{aligned}\mathbb{H}^C(S) = \{A = ((s_1, \Delta t_1), \cdots, (s_k, \Delta t_k))| \\ k \in \mathbb{N} \cup \{0\}, a_i \in S, \Delta t_i \in \mathbb{R}^+\}\end{aligned}$$

Notice that a continuous sequence actually defines a function from $[0, \sum_{i=1}^{k} \Delta t_i)$ to S in which any $x \in [\sum_{j=1}^{i-1} \Delta t_j, \sum_{j=1}^{i} \Delta t_j)$ is mapped to s_i. Also notice that any function from $[0, T)$ to S in which the output changes a finite number of times when the input changes continuously from 0 to T can also be represented using a finite continuous sequence. We denote the value of the function defined by continuous sequence A at point x by $A(x)$.

Sequence Function: Let S be a finite set. Any function $u : \mathbb{H}^D(S) \rightarrow \mathbb{R}$ is called a sequence function (discrete). Also, any function $u : \mathbb{H}^C(S) \rightarrow \mathbb{R}$ is called a sequence function (continuous).

Length of a Sequence: We denote the length of a sequence A by $|A|$ which we define next. For any discrete sequence $A = (s_1, \cdots, s_k)$ we define $|A| = k$. For any continuous sequence $A = ((s_1, \Delta t_1), \cdots, (s_k, \Delta t_k))$ we define $|A| = \sum_{i=1}^{k} \Delta t_i$.

Equivalence of Sequences: We say two sequences A and B are equivalent and denote that by $A \equiv B$ if they represent the same sequence that is if and only if they have the same length and their corresponding functions have the same value at every point in their domain. The formal definition is given next.

If A and B are two discrete sequences, then $A \equiv B$ if and only if $|A| = |B|$ and for $\forall i \in \{1, \cdots, |A|\} : A(i) = B(i)$.

If A and B are two continuous sequences, then $A \equiv B$ if and only if $|A| = |B|$ and $\forall x \in [0, |A|) : A(x) = B(x)$.

Concatenation of Sequences: We denote the concatenation of two sequences A and B by $A \bot B$.

Refinement of a Sequence: We denote the portion of a discrete sequence A in $[x, y]$ by $A_{[x,y]}$ and also the portion of a continuous sequence A in $[x, y)$ by $A_{[x,y)}$ which we formally define as the following.

For a discrete sequence $A = (s_1, \cdots, s_k)$, if the intersection of $[1, k]$ and $[x, y]$ is empty we define $A_{[x,y]}$ to be the empty sequence. Otherwise suppose $[f, l]$ is the intersection of the two, then we define $A_{[x,y]} = (s_f, \cdots, s_l)$.

For a continuous sequence $A = ((s_1, \Delta t_1), \cdots, (s_k, \Delta t_k))$, if the intersection of $[0, |A|)$ and $[x, y)$ is empty we define $A_{[x,y)}$ to be the empty sequence. Otherwise suppose $[f, l)$ is their intersection then we define:

$$\begin{aligned} A_{[x,y)} = ((s_p, \Delta t_p - \delta), (s_{p+1}, \Delta t_{p+1}), \cdots \\ \cdots, (s_{q-1}, \Delta t_{q-1}), (s_q, \Delta t_q - \delta')) \end{aligned} \tag{3.2.2}$$

where $q, l \in \mathbb{N}$ and $\delta, \delta' \in \mathbb{R}^+ \cup \{0\}$ are chosen such that:

$$\sum_{i=1}^{p-1} \Delta t_i \leq f < \sum_{i=1}^{p} \Delta t_i \tag{3.2.3}$$

$$\sum_{i=1}^{q-1} \Delta t_i < l \leq \sum_{i=1}^{q} \Delta t_i \tag{3.2.4}$$

$$\delta = f - \sum_{i=1}^{p-1} \Delta t_i \tag{3.2.5}$$

$$\delta' = \sum_{i=1}^{q} \Delta t_i - l \tag{3.2.6}$$

Domination of Sequences: We say sequence A is dominated by sequence B and we show that by $A \prec B$ if we can cut out parts of B to get A. Next we give a formal definition.

If A and B are discrete sequences then $A \prec B$ if and only if A is a subsequence of B.

If A and B are continuous sequences then $A \prec B$ if and only if there exist $m \in \mathbb{N}, 0 \leq x_1 < x_2 < \cdots < x_{2m} \leq |B|$ such that:

$$A \equiv B_{[x_1,x_2)} \perp \cdots \perp B_{[x_{2m-1},x_{2m})} \tag{3.2.7}$$

Marginal Value of a Sequence Function: For a sequence function $u : \mathbb{H}(S) \rightarrow \mathbb{R}$ we define $u(B|A) = u(A \perp B) - u(A)$ where $A, B \in \mathbb{H}(S)$.

In this chapter, $\emptyset$, will denote the empty sequence. We will also use $\mathbb{H}(S)$ instead of $\mathbb{H}^C(S)$ and $\mathbb{H}^D(S)$ when a proposition applies to both discrete sequences as well as continuous sequences.

3.2.1 Submodular Non-decreasing Sequence Functions

In this part, we define the class of *s*ubmodular non-decreasing sequence functions. In the next sections we provide a greedy heuristic for maximizing such functions subject to a given maximum length for the solution sequence.

Let S be a finite set and $u : \mathbb{H}(S) \to \mathbb{R}$ be a sequence function. We define the following conditions:

Condition 1 (Non-Decreasing)**.** *A sequence function u is* non-decreasing *if:*

$$\forall A, B \in \mathbb{H}(S) : A \prec B \Rightarrow u(A) \leq u(B) \tag{3.2.8}$$

$$u(\emptyset) = 0 \tag{3.2.9}$$

Condition 2 (Sequence-Submodularity)**.** *A sequence function u is* sequence-submodular *if:*

$$\forall A, B, C \in \mathbb{H}(S) : A \prec B \Rightarrow u(C|A) \geq u(C|B) \tag{3.2.10}$$

Condition 3 (Differentiability)**.** *This condition only applies to continuous sequence functions. Note that we use the term* "continuous sequence function" *to signify that the argument to the function is a continuous sequence and not the function itself, however the* differentiability *condition that we define next is a property of the function. A continuous sequence function $u : \mathbb{H}^C(S) \to \mathbb{R}$ satisfies the differentiability condition if for any $A \in \mathbb{H}^C(S)$, $u(A_{[0,t)})$ is continuous and differentiable with a continuous derivative with respect to t for $t \in [0, \infty)$ except that at a finite number of points it may have different left and right derivatives and thus a non-continuous derivative.*

3.3 Greedy Heuristic (Discrete)

Here we provide a greedy heuristic for maximizing non-decreasing submodular sequence functions (discrete). Let S be a finite set and $u : \mathbb{H}^D(S) \to \mathbb{R}$ be a non-decreasing submodular sequence function. Consider the problem of finding a sequence $H \in \mathbb{H}^D(S)$ that maximizes u subject to $|H| \leq T$ for a given $T \in \mathbb{N}$. Also suppose that $O \in \mathbb{H}^D(S)$ where $O = (r_1, \cdots, r_T)$ is the optimal solution to this problem.

Lemma 3.3.1. *For any $A, B \in \mathbb{H}^D(S)$ there exist $s \in S$ such that $u(s|A) \geq \frac{1}{|B|}u(B|A)$*

Proof. Suppose $B = (s_1, \cdots, s_k)$, using the definition of u we have:

$$\begin{aligned} u(B|A) &= u(H^{i-1} \perp O) - u(H^{i-1}) && (3.3.1) \\ &= u(H^{i-1} \perp O_{[1,T]}) - u(H^{i-1}) && (3.3.2) \\ &= \sum_{j=1}^{T} u(H^{i-1} \perp O_{[1,j]}) - \sum_{j=0}^{T-1} u(H^{i-1} \perp O_{[1,j]}) && (3.3.3) \\ &= \sum_{j=1}^{T} (u(H^{i-1} \perp O_{[1,j]}) - u(H^{i-1} \perp O_{[1,j-1]})) && (3.3.4) \\ &= \sum_{j=1}^{T} u(O_{[j,j]} | H^{i-1} \perp O_{[1,j-1]}) && (3.3.5) \\ &= \sum_{j=1}^{k} u(s_j | A \perp B_{[1,j-1]}) && (3.3.6) \end{aligned}$$

The sum on the right hand side of (3.3.6) consist of k terms, so there should be at least one term which is above or equal to the average of the terms. That means there should be an index $1 \leq j' \leq k$ such that (3.3.7) holds.

$$\begin{aligned} u(s_{j'} | A \perp B_{[1,j'-1]}) &\geq \frac{1}{k} u(B|A) && (3.3.7) \\ u(s_{j'} | A) &\geq \frac{1}{|B|} u(B|A) && (3.3.8) \end{aligned}$$

Combining (3.3.7) with Condition 2 because $A \prec A \perp B_{[1,j'-1]}$ we get (3.3.8) which completes the proof.

□

We use the Lemma 3.3.1 to prove the following theorem:

Theorem 1. *For sequence $H \in \mathbb{H}^D(S)$ where $H = (s_1, \cdots, s_T)$ and $\alpha \in [0, 1]$ if:*

$$\forall i \in \{1, \cdots, T\}, \forall s \in S : u(s_i | H_{[1,i-1]}) \geq \alpha \; u(s | H_{[1,i-1]}) \tag{3.3.9}$$

then:

$$\frac{u(H)}{u(O)} \geq 1 - \frac{1}{e^{\alpha}} \tag{3.3.10}$$

Proof. According to Lemma 3.3.1 we argue that for any $H = (s_1, \cdots, s_T)$ and α for which (3.3.9) holds, (3.3.11) must also hold.

$$u(s_i|H_{[1,i-1]}) \geq \frac{\alpha}{T} u(O|H_{[1,i-1]}) \tag{3.3.11}$$

$$u(s_i|H_{[1,i-1]}) \geq \frac{\alpha}{T} (u(O \perp H_{[1,i-1]}) - u(H_{[1,i-1]})) \tag{3.3.12}$$

$$u(s_i|H_{[1,i-1]}) \geq \frac{\alpha}{T} (u(O) - u(H_{[1,i-1]})) \tag{3.3.13}$$

$$u(H_{[1,i]}) - u(H_{[1,i-1]}) \geq \frac{\alpha}{T} (u(O) - u(H_{[1,i-1]})) \tag{3.3.14}$$

$$u(H_{[1,i]}) \geq \frac{\alpha}{T} u(O) + (1 - \frac{\alpha}{T}) u(H_{[1,i-1]}) \tag{3.3.15}$$

In order to derive (3.3.13) from (3.3.12) we have used Condition 1 to infer that $u(O \perp H_{[1,i-1]}) \geq u(O)$.

$$u(H_{[1,T]}) \geq \left(1 - (1 - \frac{\alpha}{T})^T\right) u(O) \tag{3.3.16}$$

$$u(H) \geq \left(1 - \left((1 - \frac{\alpha}{T})^{\frac{T}{\alpha}}\right)^{\alpha}\right) u(O) \tag{3.3.17}$$

$$u(H) \geq \left(1 - \frac{1}{e^{\alpha}}\right) u(O) \tag{3.3.18}$$

Notice that (3.3.15) defines a recurrence relation which can be solved to get (3.3.18) which completes the proof. □

The condition of Theorem 1 is simply saying that $H = (s_1, \cdots, s_T)$ should be chosen by choosing each s_i locally such that $p(s_i|H_{[1,i-1]})$ is at least α times its optimal local maximum. Setting $\alpha = 1$ means we can compute the locally optimal s_i conditioned on $s_1, \cdots, s_{i-1}$. Based on the previous intuition we present greedy algorithm 1 to find H.

The greedy algorithm 1 starts with an empty sequence H^0 and then builds the complete sequence by finding at iteration i the s_i that gives the highest increase in the value of u when appended to the end of the current sequence or more formally the s_i that maximizes $u(s_i|H^{i-1})$ (or equivalently maximizes $u(H^{i-1} \perp s_i)$). Also note that in Algorithm 1, at the step where we find s_i that maximizes $u(s_i|H^{i-1})$. We may not be able to find the locally optimal s_i and instead we may only be able to find s_i for which $u(s_i|H^{i-1})$ is at least α times its locally optimal maximum.

```
$H^0 \leftarrow \emptyset$ ;
for $i = 1$ to $T$ do
    find $s_i$ that maximizes $u(s_i|H^{i-1})$ ;
    $H^i \leftarrow H^{i-1} \perp s_i$ ;
end
$H \leftarrow H^T$ ;
```

Algorithm 1: Greedy for discrete case

Theorem 2. *For any non-decreasing submodular function u and any given $T \in \mathbb{N}$, greedy algorithm 1 can be used to find a sequence that produces a value of u which is at least $1 - \frac{1}{e^\alpha}$ times of the optimal. In particular if we can locally find the optimal at each iteration the resulting sequence gives a value of u which is at least $1 - \frac{1}{e}$ of the global optimal.*

Proof. The proof of Theorem 2 trivially follows from theorems 1 and 1. □

3.4 Greedy Heuristic (Continuous)

In this section we provide an equivalent of the greedy heuristic of section 3.3 for the continuous version. Let S be a finite set and $u : \mathbb{H}^C(S) \to \mathbb{R}$ be a differentiable non-decreasing submodular sequence function. Consider the problem of finding a continuous sequence $H \in \mathbb{H}^C(S)$ that maximizes u subject to $|H| \leq T$ for a given $T \in \mathbb{R}^+$. Also suppose that $O \in \mathbb{H}^C(S)$ where $O = ((r_1, \Delta w_1), \cdots, (r_k, \Delta w'_k))$ is the optimal solution.

We define $\dot{u}_s(\delta|A)$ where $s \in S$, $\delta \in \mathbb{R}^+$ and $A \in \mathbb{H}$ as the following:

$$\begin{aligned}
\dot{u}_s(\delta|A) &= \frac{d}{d\delta} u((s,\delta)|A) && (3.4.1)\\
&= \frac{d}{d\delta}\left(u(A \perp (s,\delta)) - u(A)\right) && (3.4.2)\\
&= \frac{d}{d\delta} u(A \perp (s,\delta)) && (3.4.3)
\end{aligned}$$

We also define $\dot{u}_s(\delta|A)$ at $\delta = 0$ as the following:

$$\dot{u}_s(0|A) = \lim_{\delta \to 0^+} \dot{u}_s(\delta|A) \qquad (3.4.4)$$

Note that (3.4.1) is always defined because we are assuming that u satisfies the Condition 3 and (3.4.3) can be written as $\frac{d}{d\delta} u((A \perp (s,\infty))_{[0,|A|+\delta)})$. Also note that according to Condition 3 $\dot{u}_s$ is a continuous function over $\mathbb{R}^+$ except at a finite number of points.

Corollary 1. *For any $A \in \mathbb{H}^C$ like $A = ((s_1, \Delta t_1), \cdots, (s_k, \Delta t_k))$ let $A^i = ((s_1, \Delta t_1), \cdots, (s_i, \Delta t_i))$ then all of the following hold:*

$$u((s, \delta)|A) = \int_0^{\delta} \dot{u}_s(x|A)dx \tag{3.4.5}$$

$$u((s, \delta_2)|A\perp(s, \delta_1)) = \int_{\delta_1}^{\delta_2} \dot{u}_s(x|A)dx \tag{3.4.6}$$

$$u(A) = \sum_{i=1}^{k} \int_0^{\Delta t_i} \dot{u}_{s_i}(x|A^{i-1})dx \tag{3.4.7}$$

Proof. (3.4.5) and (3.4.6) trivially follow from (3.4.1) and (3.4.7) follows from the definition of marginal values. □

Lemma 3.4.1. *For any $A, B \in \mathbb{H}^C$ such that $A \prec B$ and any $s \in S$, we have $\dot{u}_s(\delta|A) \geq \dot{u}_s(\delta|B)$ for any $\delta \in \mathbb{R}^+ \cup \{0\}$ except at a finite number of points.*

Proof. The proof is by contradiction. Suppose there are $A, B \in \mathbb{H}^C$ such that $A \prec B$ and $s \in S$ and $\delta \in \mathbb{R}^+$ for which $\dot{u}_s(\delta|A) < \dot{u}_s(\delta|B)$. If either $\dot{u}_s(\delta|A)$ or $\dot{u}_s(\delta|B)$ is non-continuous at δ then this is one of the finite number of points that are exceptions in Lemma 3.4.1. Otherwise since they are both continuous at δ there should be a small neighborhood around δ in which $\dot{u}_s(\delta|B)$ is greater than $\dot{u}_s(\delta|A)$. More formally:

$$\exists \epsilon \in \mathbb{R}^+, \forall x \in [\delta - \epsilon, \delta + \epsilon] : \dot{u}_s(x|A) < \dot{u}_s(x|B) \tag{3.4.8}$$

Now we show that (3.4.8) can never happen:

$$u((s, \epsilon)|A\perp(s, \delta - \epsilon)) = \int_{\delta-\epsilon}^{\delta} \dot{u}_s(x|A) \tag{3.4.9}$$

$$u((s, \epsilon)|A\perp(s, \delta - \epsilon)) < \int_{\delta-\epsilon}^{\delta} \dot{u}_s(x|B) \tag{3.4.10}$$

$$u((s, \epsilon)|A\perp(s, \delta - \epsilon)) < u((s, \epsilon)|B\perp(s, \delta - \epsilon)) \tag{3.4.11}$$

Notice that $A\perp(s, \delta - \epsilon) \prec B\perp(s, \delta - \epsilon)$ and therefore (3.4.11) contradicts Condition 2 which says u is a submodular sequence function. It shows that our assumption of $\dot{u}_s(\delta|A) < \dot{u}_s(\delta|B)$ leads to contradiction which completes the proof. □

Corollary 2. *For any $A \in \mathbb{H}^C(S)$, and any $\delta \in [0, \infty)$, $\dot{u}_s(\delta|A)$ is a monotonically non-increasing function in δ. That is $\delta_1 < \delta_2 \Rightarrow \dot{u}_s(\delta_1|A) \geq \dot{u}_s(\delta_2|A)$.*

Proof. The proof is similar to the proof of Lemma 3.4.1. □

The following lemma in the equivalent of Lemma 3.3.1 for the continuous case.

Lemma 3.4.2. *For any $A, B \in \mathbb{H}^C(S)$ there exist $s \in S$ such that $\dot{u}_s(0|A) \geq \frac{1}{|B|}u(B|A)$*

Proof. Suppose $B = ((s_1, \Delta t_1), \cdots, (s_k, \Delta t_k))$ and let $B^i = ((s_1, \Delta t_1), \cdots, (s_i, \Delta t_i))$. Using the definition of u and (1) we have:

$$u(B|A) = \sum_{i=1}^{k} \int_0^{\Delta t_i} \dot{u}_{s_i}(x|A \bot B^{i-1})dx \tag{3.4.12}$$

We argue that there should be some $1 \leq i \leq k$ for which there exist some $\delta \in [0, \Delta t_i)$ such that $\dot{u}_{s_i}(\delta|A \bot B^{i-1}) \geq \frac{1}{|B|}u(B|A)$ otherwise that means the term inside the integral on the right hand side of (3.4.12) is always less than $\frac{1}{|B|}u(B|A)$ which means the sum of the integrals would be less that $u(B|A)$ which contradicts the (3.4.12). Suppose for i' and δ' (3.4.13) holds.

$$\dot{u}_{s_{i'}}(\delta'|A \bot B^{i'-1}) \geq \frac{1}{|B|}u(B|A) \tag{3.4.13}$$

$$\dot{u}_{s_{i'}}(\delta'|A) \geq \frac{1}{|B|}u(B|A) \tag{3.4.14}$$

$$\dot{u}_{s_{i'}}(0|A) \geq \frac{1}{|B|}u(B|A) \tag{3.4.15}$$

We can infer (3.4.14) from (3.4.13) by using Lemma 3.4.1. Applying Corollary 2 to that we get (3.4.15) which completes the proof. □

Next we present our main result for this section.

Theorem 3. *For any sequence $H \in \mathbb{H}^C(S)$ where $H = ((s_1, \Delta t_1), \cdots, (s_k, \Delta t_k))$ and $|H| = T$ and $\alpha \in [0, 1]$, if:*

$$\forall t \in [0, T), \forall s \in S : \frac{d}{dt}u(H_{[0,t)}) \geq \alpha \ \dot{u}_s(0|H_{[0,t)}) \tag{3.4.16}$$

then:

$$\frac{u(H)}{u(O)} \geq 1 - \frac{1}{e^\alpha} \tag{3.4.17}$$

Proof. Using Lemma 3.4.2 we have (3.4.18). Combining that with (3.4.16) we get (3.4.19). Using the definition of marginal values and using Condition 2 we get (3.4.21) which is a differential equation.

$$\forall t \in [0,T)\ \exists s \in S:\ \dot{u}_s(0|H_{[0,t)}) \geq \frac{1}{|O|}u(O|H_{[0,t)}) \tag{3.4.18}$$

$$\forall t \in [0,T):\ \frac{d}{dt}u(H_{[0,t)}) \geq \frac{\alpha}{T}u(O|H_{[0,t)}) \tag{3.4.19}$$

$$\forall t \in [0,T):\ \frac{d}{dt}u(H_{[0,t)}) \geq \frac{\alpha}{T}(u(O\perp H_{[0,t)}) - u(H_{[0,t)})) \tag{3.4.20}$$

$$\forall t \in [0,T):\ \frac{d}{dt}u(H_{[0,t)}) \geq \frac{\alpha}{T}(u(O) - u(H_{[0,t)})) \tag{3.4.21}$$

We can rephrase the (3.4.21) as (3.4.22) and solve it to get (3.4.26).

$$u(H_{[0,t)}) + \frac{T}{\alpha}\frac{d}{dt}u(H_{[0,t)}) \geq u(O) \tag{3.4.22}$$

$$\frac{d}{dt}\left(\frac{T}{\alpha}e^{\frac{\alpha}{T}t}u(H_{[0,t)})\right) \geq \frac{T}{\alpha}e^{\frac{\alpha}{T}t}u(O) \tag{3.4.23}$$

$$\int_0^x \frac{d}{dt}\left(\frac{T}{\alpha}e^{\frac{\alpha}{T}t}u(H_{[0,t)})\right)dt \geq \int_0^x e^{\frac{\alpha}{T}t}u(O)dt \tag{3.4.24}$$

$$\frac{T}{\alpha}e^{\frac{\alpha}{T}x}u(H_{[0,x)}) \geq \frac{T}{\alpha}(e^{\frac{\alpha}{T}x} - 1)u(O) \tag{3.4.25}$$

$$u(H_{[0,x)}) \geq (1 - \frac{1}{e^{\frac{\alpha}{T}x}})u(O) \tag{3.4.26}$$

$$u(H) \geq (1 - \frac{1}{e^{\alpha}})u(O) \tag{3.4.27}$$

Setting $x = T$ in (3.4.26) we get (3.4.27) which completes the proof. □

The condition of Theorem 3 is simply saying that H should be chosen such that at each point $t \in [0,T)$, the derivative of u is at least α times its optimal local maximum. Setting $\alpha = 1$ means at each $t \in [0,T)$ we can find the best $s \in S$ conditioned on $H_{[0,t)}$. Based on the previous intuition we present a generic greedy algorithm 2 to find H. This algorithm in general may not terminate, however if it terminates, for the resulting H, $u(H)$ will be at least $(1 - \frac{1}{e^{\alpha}})$ times the optimal. In general there can be other ways for finding such an H for each specific problem as we will show one such example later in this paper.

```
$t \leftarrow 0$ ;
$i \leftarrow 1$ ;
$H^0 \leftarrow \emptyset$ ;
while $t < T$ do
    find $(s_i, \Delta t_i)$ such that
    $\forall s \in S \ \forall \delta \in [0, \Delta t_i) : \ \dot{u}_{s_i}(0 | H^{i-1} \perp (s_i, \delta)) \geq \alpha \ \dot{u}_s(0 | H^{i-1} \perp (s_i, \delta))$ ;
    $H^i \leftarrow H^{i-1} \perp (s_i, \Delta t_i)$ ;
    $t \leftarrow t + \Delta t_i$ ;
    $i \leftarrow i + 1$ ;
end
$H \leftarrow H^{i-1}$ ;
```

Algorithm 2: Greedy for continuous case

In algorithm 2 in the main loop we need an *Incremental Oracle* that is specific to each problem. As we mentioned before, there might be other ways for finding a sequence H that satisfies the condition of Theorem 3 and as long as it satisfies that condition we have the $1 - \frac{1}{e^\alpha}$ guarantee.

3.5 Online ad allocation problem

In this section, we define the greedy algorithm for *Online Ad Allocation* problem and show that greedy approach achieves $1 - \frac{1}{e}$ approximation for this problem. We define a *"configuration"* as a mapping of query types to ads such that each query type is mapped to at most d ads. Let S be the set of all possible configurations. We can now represent any allocation of ads to queries over time $[0, T)$ by a continuous sequence $H = ((s_1, \Delta t_1), \cdots, (s_k, \Delta t_k))$ where $s_i \in S$, $\Delta t_i \in \mathbb{R}^+$, $k \in \mathbb{N}$ and $|H| = T$ which means *"Use each configuration s_p (in order) for a duration of Δt_p for $p \in \{1, \cdots, k\}$"*. We call H an *"Allocation Strategy"*.

Let $u(H)$ be the expected utility of the search engine for using an allocation strategy H. Note that for any given sequence of queries we can say based on H exactly which ads are displayed for each incoming query and so we can directly compute the utility of the search engine. Next we show that u is a *Submodular Non-decreasing Sequence Function* and so using a greedy algorithm yields an allocation that is at least $1 - \frac{1}{e}$ of the optimal. First we explain how the greedy algorithm works.

At any point in time, the greedy method chooses the best configuration as follows: For each query type j map it to the d ads with highest p_{ij} among those that have not exhausted their budgets yet and denote them by $Q_j(s)$. Let $r(s)$ be the expected revenue rate of such a configuration s. We can write $r(s)$ as follows:

$$r(s) = \sum_{j \in N} q_j \sum_{i \in Q_j(s)} p_{ij} \tag{3.5.1}$$

Note that the revenue of the search engine for using configuration s for a short period of length Δt assuming that none of the ads exhaust their budget during that time is given by $r(s)\Delta t$.

The greedy algorithm works as follows: Choose the best configuration (the one with maximum $r(s)$) as explained above by assigning query type j to the d ads with highest p_{ij} among those that have not exhausted their budgets yet. Keep that configuration until at least one of the ads runs out of budget. Then recompute the best configuration and switch to it. It is easy to see that the derivative of $u(H_{[0,t)})$ with respect to t is $r(s)$ where $s = H(t)$ is the configuration that is active at time t in H. That is because $\frac{d}{dt}u(H_{[0,t)}) = \dot{u}_s(H_{[0,t)})$ is exactly the rate at which the search engine is accumulating profit at time t which is $r(s)$ and for all other $s' \in S$ we have $r(s') \leq r(s)$ at time t. That also means that our greedy algorithm satisfies the requirement of the incremental oracle in 2 as the current configuration always has a higher revenue rate than all the other configurations. Also note that we may need to change the configuration only when an ad runs out of budget which means the total number of configuration changes is no more than m. The only thing that remains to be shown is that the utility function u is a *Submodular Non-decreasing Sequence Functions* which we prove next.

Lemma 3.5.1. *The utility function of online ad allocation problem satisfies Condition 1. In particular, consider the allocation strategies $A, B \in \mathbb{H}$ and assume that $A \prec B$. The remaining budget of each ad at the end of using B is less than or equal to its remaining budget in A.*

Proof. Consider the allocation strategies $A, B \in \mathbb{H}$ and assume that $A \prec B$. We argue that the profit extracted from each ad in B is at least as much as the profit extracted from each ad in sequence A.

We partition the ads into two categories:

- Ads that have no budget left after running sequence B.
- Ads that still have budget after running sequence B.

In the former case, sequence B extracted the maximum possible budget from the ad. So for this set of ads, our claim holds.

For the ads that belong to the second category, we know that they still have budget available. Consider an ad i that belongs to this category. We will show that the profit extracted by B from this ad is at least as much as the profit extracted by A.

Consider the configuration $s \in S$ that is active in B for a total time of Δt. For all queries of type j that arrive during that time and any ad i that is allocated to them by configuration s, we know that the profit extracted from budget of ad i by those queries is $\Delta t q_j$ because ad i never ran out of budget. Since $A \prec B$, configuration s is either not present in A or was used in A for less total time than B and so the total profit extracted from ad i in A is no more than the profit extracted from ad i in B.

Since for both categories the expected profit extracted by B from each ad is higher than or equal to the profit extracted by A from that ad, we can conclude that the non-decreasing property holds. □

Next, we show that Condition 2 holds as well.

Lemma 3.5.2. *Online ad allocation problem satisfies Condition 2.*

Proof. Consider the allocation strategies $A, B, C \in \mathbb{H}$ and assume that $A \prec B$. First of all, based on Lemma 3.5.1, we know that the remaining budget of each ad after A is less than or equal to its remaining budget after B. It is also easy to see that the contribution of each ad to $u(C|B)$ or $u(C|A)$ is equal to the difference in its budget before and after using the C. Now, consider using the allocation strategy A first followed by C. Again we partition the ads into two categories:

- Ads that have exhausted all of their budget after running $A \perp C$.
- Ads that still have budget after running $A \perp C$.

The contribution of the ads in the first category to $u(C|B)$ is no more than their contribution to $u(C|A)$ because they had equal or more remaining budget after using A than after using B and they have contributed all of their remaining budget to $u(C|A)$.

Now consider the ads that belong to the second category. By the same reasoning as we did for the proof of Lemma 3.5.1 we conclude that C has extracted profit from those ads at full rate since they did not run out of budget. So their contribution to $u(C|A)$ and $u(C|B)$ is equal. □

Finally, we can show that Condition 3 is also met. Notice that the derivative of the utility function is a step function that changes its value only when either in the sequence there is change of configuration or when some ad runs out of budget. The utility function is therefore differentiable and its derivative is continuous except on the endpoints of each piece. The total number of pieces is bounded by the number of ads which is finite. Therefore we conclude that the utility function is differentiable and its derivative is continuous except at a finite number of points.

Using the above properties, we conclude that the approximation ratio of greedy algorithm that will select the best configuration at each point of time is $(1 - \frac{1}{e})$ and it completes the proof. As a result we have:

Theorem 1. *Greedy approach obtains* $(1 - \frac{1}{e})$ *approximation for online ad allocation problem assuming that the queries are coming from a fixed but unknown distribution.*

Chapter 4
Query Rewriting for Keyword-based Advertising

As described before, the main objective in search advertising, is to show relevant ads for a given query. However, even though 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, or the set might be even empty. The objective is to find the set of ads that are not in the set but are relevant and appropriate to be shown for the keyword. For instance, an ad bidding on the keyword "wedding band" may be appropriate for the query "engagement ring" even if that ad hasn't bid on the keyword "engagement ring". A common mechanism to improve the relevance of keyword advertising is query rewriting. Query rewriting outputs a list of queries $q_1, q_2, \ldots, q_n$ (referred to as rewrites) whose ads are also relevant to a given query q. Most work on this topic, however, has primarily focused on generating relevant rewrites with respect to the original query. In query rewriting for keyword advertising, there are some extra constraints that we should consider while trying to optimize our objective function. The constraints can be summarized as follows:

1. Constraints on the number of rewrites per query due to system considerations.
2. Constraints on the number of queries for which a query rewrite can be used due to undesirability of frequency for users (we also call this frequency capping).
3. Budget constraints of an advertiser.

4.1 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 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.

- The advertiser bidding landscape for a keyword is very dynamic and fast changing. Advertisers manually or automatically distribute their budget 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, respectively. 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 *a priori* computation of keyword-ad relevance).

- 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.

- 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 chapter is organized as follows. In Section section 4.2, we state our combinatorial formulation. Section section 4.3 contains the results for the cardinality version and Section section 4.4 contains the results for the weighted version. section 4.5 contains the experimental results.

4.2 Formulation

In this section we formalize the problem framework and provide the necessary notation and technical background. Consider the 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 (Figure 4.1). 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.

We define the following notations:

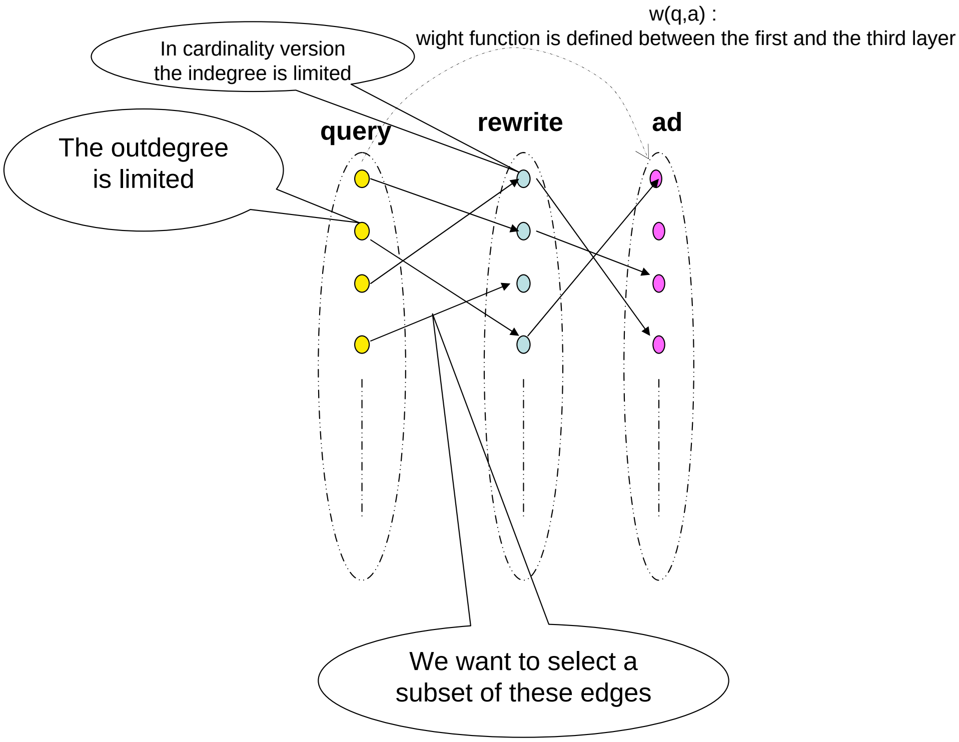


Figure 4.1: three layer graph containing queries, rewrites and ads

$\mathbf{\Gamma(Q')}$: For a subset $Q' \subseteq Q$ of queries, we use $\Gamma(Q') \subseteq W$ to denote all the rewrites that arise from some query in Q'.

$\mathbf{\Gamma(W')}$: For $W' \subseteq W$, we use $\Gamma(W') \subseteq A$ to denote all the ads that can be covered by some rewrite in W'.

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

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

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

$\mathbf{B_{q,d}(A')}$: is the set of top d ads in A' that are contributing in $\beta_{q,d}(A')$ function.

$\beta_{\mathbf{q,d}}(\mathbf{W'})$: is the benefit obtained from the set of top d ads belonging to $\Gamma(W')$ according to the benefit function $\beta_q(\cdot)$, where $W' \subseteq \Gamma(\{q\})$.

$\mathbf{B_{q,d}(W')}$: is the set of ads that are contributing in $\beta_{q,d}(W')$.

$\beta_{\mathbf{d}}(\mathbf{E'_Q})$: is $\sum_{q|(q,\cdot)\in E'_Q} \beta_{q,d}(\{w \mid (q,w) \in E'_Q\})$.

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:

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:

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

We now focus on the objective of this variant.

The objective then is to find E'_Q, subject to constraints given above, such that the d-benefit of $\beta_d(E'_Q)$ is maximized.

Weighted version. In the weighted version of the problem, the setup we assume is the following. Let $d > 0$ be given. Given a set of query rewrites, i.e., a subset of edges in E_Q, an *ad allocator* will optimally allocate 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:

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, $\sum_{i=1}^{\ell} T_i \leq L(a)$.

4.3 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 queries.

4.3.1 Warmup: Single query case

Note that since there is a single query, constraint (b) is vacuous, the query rewrites can 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 $Q = \{q\}$.

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 the result of Nemhauser et al. in [45].

Theorem 1. *The max d-benefit function is non-decreasing and submodular.*

```
$W' \leftarrow \emptyset$;
while $|W'| \leq K$ do
    Find $w \in W \setminus W'$ that maximizes $\beta_{q,d}(\Gamma(W' \cup \{w\}))$ ;
    $W' \leftarrow W' \cup \{w\}$ ;
end
Return $W'$ ;
```

Algorithm 3: Single-Query-Greedy

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'). □

Also it can be shown that $(1-1/e)$ is a tight bound for Algorithm Single-Query-Greedy4.3.4. It is easy to see that Algorithm Single-Query-Greedy runs in time $O((\sum_{w \in W} |\Gamma(W)|) \cdot K(\log d))$.

4.3.2 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, $\beta_d(E'_Q) = \sum_{q\in Q} \beta_{q,d}(R_q)$.

```
$E_g \leftarrow \emptyset$ ;
**while** $\exists e \in E_Q$ *that is unmarked* **do**
    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:
    **if** $|R_{q^*}| < K$ *and* $|\{q : (q, w) \in E_g\}| < D(w)$ **then**
        add $(q^*, w)$ to $E_g$ ;
    **end**
    Mark $(q*, w)$ ;
**end**
**return** $E_g$ ;
```

Algorithm 4: General-Greedy

The approximation algorithm is once again a greedy one. We can formulate our problem 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 [47]. 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 to the 1/3 approximation might be of independent interest.

Theorem 2. *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 ($|R_q| = K$) or w has been chosen as a rewrite $D(w)$ times ($|\{q : (q, w) \in E_g\}| = D(w)$).

Partition the set of edges in $E_o \setminus E_g$ into two groups, S_Q and S_W. Here, 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 make a one-to-one charging 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:

$$\begin{aligned}
s_1 &\geq \frac{OPT_q}{K} \\
s_2 &\geq \frac{OPT_q - s_1}{K} \\
&\vdots \\
s_K &\geq \frac{OPT_q - \sum_{i=1}^{K-1} s_i}{K}
\end{aligned}$$

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 $\beta_d(E_g) \geq (e-1)/(2e-1)OPT$. □

4.3.3 Hardness results for Cardinality Version

In this section, we show the hardness results for both variants of single query case as well as the general version.

Lemma 4.3.1. *Single query rewriting with cardinality constraint is NP-hard.(If d is an input parameter)*

Proof. To show the NP-hardness we reduce the maximum coverage to this problem. Consider an instance of maximum coverage where we want to select K of the subsets $S_1, \ldots, s_n \subseteq S$ and also we want to maximize the total number of coverage elements with those subsets. Now we reduce this instance to the following query rewriting problem. For the specified given query q,put one possible rewrite node w_i corresponding to each s_i. Also for each element $e_i \in S$ put add $a_i \in A$ and connect each W_i to a_j iff $e_j \in S_i$. Now if we set d large enough (for example any value $d > |S|$.) the best solution for the maximum coverage can be obtained by solving the corresponding single query rewriting instance with cardinality constraint. □

It can also be shown that the general case of the cardinality variant is NP-hard as well.

Lemma 4.3.2. *General query rewriting with cardinality constraint is NP-hard.*

Proof. We again reduce the maximum coverage problem to this problem. Consider the similar notations for the maximum coverage instance. Now use exactly the same mapping as the one used in the previous lemma and set the value of $L = 1$. (That means each rewrite is allowed to be suggested for at most query.) However since in this instance we only have one query this constraint does not add any more restrictions. So again to solve the maximum coverage it is enough to solve the general query rewriting with cardinality constraint. (More easily we can say that we can reduce the single query case to the general case easily.) □

4.3.4 Tight examples for cardinality version

In this section we will represent the examples that show the given analysis for the greedy algorithms, for both single query rewrite and general case are tight. First we focus on the single query case.

The suggested tight example for single query has the same structure as the tight example for maximum coverage.

Theorem 3. *The approximation factor* $1 - 1/e$ *given for Algorithm single-query-greedy is tight.*

Proof. suggested greedy algorithm For some given query q call the set of its possible rewrites $w_1, \ldots, w_{2k}$. Now suppose that $w_1, \ldots, w_k$ each covers ads of total weight $1/k$ and the ads they are covering are pairwise disjoint ads. We describe the coverage of $w_{k+1}, \ldots, w_{2k}$ as follows: 1) They cover pairwise disjoint set of ads. 2) w_{k+1} will cover $1/k$th of the ads that are covered by each w_j $j \leq k$ plus an extra ad of weight ϵ. So the total covered weight by w_{k+1} is $k.1/k^2 + \epsilon = 1/k + \epsilon$. 3) w_i will cover $1/k$th of the ads that are covered by each w_j, $j \leq k$ but not covered by w_p, $p < i$ plus some extra ad of weight ϵ.(e.g w_{k+2} will cover $(k-1)/k^2$ in total, so on). See Figure 4.2. The optimal solution will suggest $w_1, \ldots, w_k$ and the total covered weight is 1. However greedy solution will first suggest w_{k+1} and the $w_{k+2}, \ldots, w_{2k}$. So the total covered weight by the greedy is $1 - (1 - 1/k)^k + \epsilon = 1 - 1/e + \epsilon$. By making ϵ arbitrarily small, we can conclude that the analysis is tight. □

We can show that the given analysis for general case is tight as well.

Theorem 4. *The approximation factor* $(e-1)/(2e-1)$ *is tight for Algorithm* General-Greedy.

Proof. Consider the following example: Suppose $Q = \{q_0, q_1, \ldots, q_k\}$, $W = \{w_1, \ldots, w_{2k}\}$. Also assume that $\forall i \geq 1$ $(q_0, w_i) \in W$ and $(q_i, w_i) \in W$. Assume that $D(w_i) = 1$. and we are allowed to suggest k rewrites for each query. Now, suppose that $w_{k+1}, \ldots, w_{2k}$ are covering $1/k$ portion of the available ads each and their coverage is pairwise disjoint. For $i \leq k$ we define the coverage of w_i as follows: w_i will cover $1/k$ of the remaining ads that are covered by each w_j $j > k$ that are not covered by previous w_p $p < i$ plus some ϵ We assume that ads have the same weight for all the queries except the ϵ ads.(We assume that they have weight zero for all the queries except q_0) That means that w_1 will cover $1/k^2$ of the ads that are covered by each $w_j, j > k$ and in total it would cover $1/k + \epsilon$ of the ads. Optimal solution here is to select $w_{k+1}, \ldots, w_{2k}$ as the suggested rewrites for q_0 and for all $1 \leq i \leq k$ choose w_i as the suggested rewrite for q_i. Hence, the benefit of the optimal solution is $1 + 1 - (1 - 1/k)^k = 2 - 1/e$.

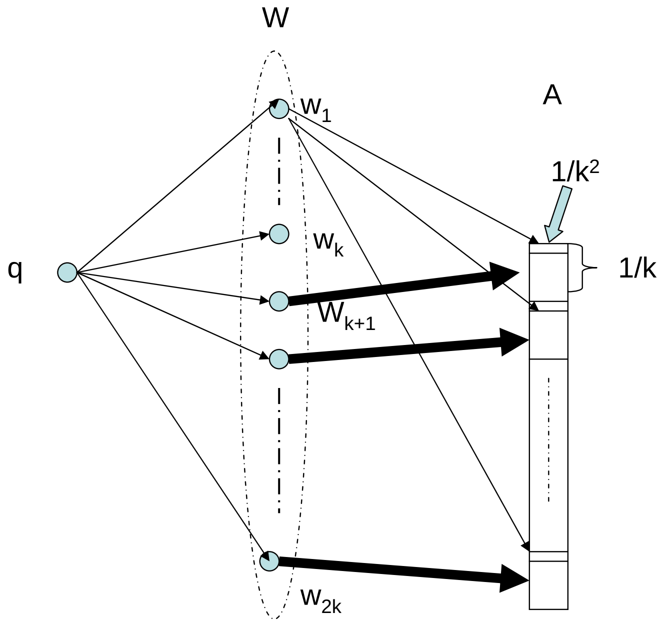


Figure 4.2: Tight example for the general cardinality case

However the greedy will select $w_1, \ldots, w_k$ as the rewrite for q_0 first and since the degree constraint for rewrites is at most 1 no other rewrite can be suggested anymore. So the total benefit by the greedy solution is $1 - (1 - 1/k)^k \approx 1 - 1/e$. Now ratio of the greedy to the optimal is $\frac{e-1}{2e-1}$ which shows that the analysis is tight. □

4.4 Algorithm for the weighted version

We are given a set M of ads and a set N of query types (keywords). Furthermore, for each ad i and query j define p_{ij}, B_i and q_j as in section 3.5. We are also given a set R of rewrites. Each rewrite $r \in R$ is associated with a small subset of the ads which we denote by W_r. The goal is to associate each query type with at most k rewrites so that later the ad-allocator only considers the ads that are associated with the rewrites of each query type in order to find the best ads to show for incoming queries of that type. Suppose that Y_j is the set of rewrites associated with query j. Formally, the ad-allocator will then only consider ad i such that $i \in \bigcup_{r \in Y_j} W_r$ to find the best d ads to show for queries of type j. The problem is how to find the sets Y_j so as to maximize the maximum profit that can be extracted by the ad allocator.

Next we give a greedy algorithm that gives a $1 - \frac{1}{e}^{1-\frac{1}{e}} \approx 0.47$ approximation which improves the previous 0.25 approximation given by Malekian et al. in [40].

We define a *"partial allocation"* as a tuple of the form $(j, Y_j, \vec{B}^j)$ where $\vec{B}^j = (B_{1j}, \cdots, B_{mj})$ is a vector of budgets in which B_{ij} is the maximum budget that we allow the ad allocator to extract from ad i for displaying the ad i for query type j. Y_j is the set of rewrites for query type j. Note that any solution of the query rewriting problem and the corresponding allocation problem can be written as a sequence of the following form:

$$H = (j_1, Y_{j_1}, \vec{B}^{j_1}), \cdots, (j_n, Y_{j_n}, \vec{B}^{j_n}) \tag{4.4.1}$$

We now define a utility function $u(H)$ on the sequences of the above form as follows:

Definition 1 ($u(H)$)**.** *Initialize u to 0 and set each of the B_i to the total budget of ad i. For each of the partial allocation tuples in order do the following. Suppose the current tuple is $(j, Y_j, \vec{B}^j)$. Set the current budget limit of each ad i to the minimum of B_i and B_{ij}. Suppose that $p_{ij} = 0$ for all ads that are not associated with any of rewrites in Y_j. Ignore all the queries of type other than j. Use the greedy algorithm of section 3.5 to solve the assignment problem for only the queries of type j considering the current budget limits. Notice that the greedy algorithm is optimal when we have* *only one query type. After computing the allocation, update each B_i to reflect how much of its budget has been used by j and then proceed to the next tuple in the sequence. Let $\vec{B}(H)$ denote the vector of remaining budgets at the end of this process (we will use this notation later). Suppose $OPT = (1, Y_1, \vec{B}^1), \cdots, (n, Y_n, \vec{B}^n)$ is the sequence in which Y_j's are the optimal rewrites and $\vec{B}^j$ is the vector of budgets used by ad j in the corresponding optimal allocation. Clearly $u(OPT)$ is equal to the total utility of the search engine for the optimal solution.*

Lemma 4.4.1. *$u(H)$ as defined in Definition 1 is a non-decreasing submodular sequence function.*

Proof Sketch: We only give the sketch of the proof as it is very similar to the proof of Lemma 3.5.1 and Lemma 3.5.2. Again we separate the ads to two groups. Those who have exhausted their budget at the end of computing u using the Definition 1 and those who still have budget left. We can then verify that for any two sequence of partial allocations A and B such that $A \prec B$, all of the ads that are in the first group after computing the $u(B)$ using Definition 1 have made their maximum contribution to $u(B)$ and so cannot contribute more to $u(A)$. For all the other ads we can show that their contributions to $u(A)$ and $u(B)$ are equal. The submodularity property follows in the same way as for Lemma 3.5.2. □

In order to be able to use the greedy algorithm 1, given in section 3.4, to approximate OPT we need an oracle that can find the best partial allocation $(j, Y_j, \vec{B}^j)$ to be appended to the current sequence. The marginal utility of adding a partial allocation $(j, Y_j, \vec{B}^j)$ is a non-decreasing submodular function in terms of Y_j with the constrain that $|Y_j| = k$. Therefore for each j we can get a $1 - \frac{1}{e}$ approximation by using a greedy algorithm. Start from an empty Y_j and add the rewrite that increases the marginal utility the most until k rewrites have been added. We then select among all possible query types j the one for which $(j, Y_j, \vec{B}^j)$ has the highest marginal utility and append that to the current sequence of partial allocation. Since we are approximating the best $(j, Y_j, \vec{B}^j)$ within a factor of $1 - \frac{1}{e}$, based on Theorem 2 the approximation ratio of the overall algorithm is $1 - \frac{1}{e^{1-\frac{1}{e}}} \approx 0.47$. The complete algorithm is described in Algorithm 5.

4.5 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 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. [12, 30, 7]). 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, Algorithm Single-Query-Greedy significantly outperforms Algorithm Baseline when comparing the d-benefit of the selected rewrites. We describe our results in detail in the next two sections.

4.5.1 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 [35, 7]:

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

- **Approximate match (2)**: A probable, but inexact match with user intent.
- **Marginal match (3)**: A distant, but plausible match to a related topic.
- **Clear mismatch (4)**: A clear mismatch.

The results of the experiment are summarized in Table 4.1.

Algorithm BASELINE does place a larger fraction of the rewrites it selects into the Precise match category; otherwise, the relevance of the two rewriting systems is equivalent.

4.5.2 *d*-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.

4.5.2.1 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 4.2. 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 4.3. In this table, we break down the percentage gain in Table 4.2 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.

	range of $\lvert W\rvert$			
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%

	range of $|W|$			
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 4.3: d-benefit: Percentage gain of Algorithm Single-Query-Greedy over Algorithm Baseline broken down by $|W|$.

```
$H \leftarrow \emptyset$ ;
for $i \in M$ do
    $B_i \leftarrow$ budget of ad $i$ ;
end
$\vec{B} \leftarrow (B_1, \cdots, B_m)$ ;
while $N \neq \emptyset$ do
    $\Delta' \leftarrow 0$ ;
    // find the best partial allocation to append
    for $j \in N$ do
        $Y_j \leftarrow \emptyset$ ;
        // find the best $k$ rewrites greedily
        for $w = 1, \cdots, k$ do
            $\delta' \leftarrow 0$ ;
            for $r \in R \backslash Y_j$ do
                $\delta \leftarrow u((j, Y_j \cup \{r\}, \vec{B})|H)$ ;
                if $\delta' < \delta$ then
                    $\delta' \leftarrow \delta$ ;
                    $r' \leftarrow r$ ;
                end
            end
            $Y_j \leftarrow Y_j \cup \{r'\}$ ;
        end
        // compute the marginal utility of adding $(j, Y_j)$
        $\Delta' \leftarrow u((j, Y_j, \vec{B})|H)$ ;
        if $\Delta' < \Delta$ then
            $\Delta' \leftarrow \Delta$ ;
            $j' \leftarrow j$ ;
        end
    end
    Define $\vec{B}^{j'}$ to be exactly equal to how much of the budgets are used by
    $(j, Y_j \cup \{r\}, \vec{B})$ when appended to $H$ ;
    $H \leftarrow H \bot (j', Y_{j'}, \vec{B}^{j'})$ ;
    $N \leftarrow N \backslash \{j'\}$ ;
    $\vec{B} \leftarrow \vec{B} - \vec{B}^{j'}$ ;
end
```

Algorithm 5: for Query Rewriting

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

Table 4.1: Relevance comparison: Algorithm BASELINE vs. Algorithm SINGLE-QUERY GREEDY

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

Chapter 5
Guaranteed Delivery Advertisement: Fair Allocation With a Compact Plan

Display advertising is a kind of advertising that typically contains text, photographs or other images and similar items.

As opposed to textual ads on search pages, in traditional way of display advertisement, impressions are sold via a contract between the advertiser and the desired publisher for the advertiser. Advertiser specifies the amount that he will pay per impression and the publisher guarantees to show a certain number of impressions for a certain period of time.

Recently there were some attempts on automating this process by introducing a third party that synchronizes between advertisers and publishers. In this scenario advertisers can specify the type of hits (for example women's shoe) they prefer without exactly specifying the publisher and then ask for the required number of impressions they want to obtain from the third party. Based on the number of available impressions via the publishers in contact with the third party, some of the advertisers will be accepted and a contract with them will be signed and the rest of the advertisers will be denied. In this context, we consider two problems. The problem considered in this chapter is related to the case where the third party has already decided which advertisers to accept. The question is how to do the allocation of impressions to advertisers so that it is close to a representative allocation and also it is feasible to do in real time (considering the enormous size of advertisement). We describe representativeness by an example: Suppose a person wants to buy 2 pounds of nuts and he asks for a mixture of almonds and peanuts. Although 2 pounds of peanuts is also a 2 pound mixture, he would be probably happier if he receives 1 pound of peanuts and 1 pound of almonds or in other words, this mixture is more representative. The detailed problem and solution are described in the rest of this chapter.

5.1 Fair Allocation with a Compact Plan

Suppose we are given a set I of different kind of *impressions*, a set U of *users*, and a set J of *contracts*. Each impression $i \in I$ has a supply $s_i > 0$ and is labeled with exactly one user $u \in U$. Each contract $j \in J$ has (1) a *weight* $w^j > 0$ per impression that captures its importance, (2) a desired set $\Gamma^j \subseteq I$ of impressions, and a *demand* $d^j > 0$, denoting the number of impressions that need to be allocated to this contract

j. For an impression i, let $\Gamma_i \subset J$ denote the set of contracts who interested in impression i.

As stated earlier, the goal is to find the most fair allocation of impressions to contracts. Let y_i^j be the number of impressions i that are assigned to contract j in a given allocation and let $\delta_i^j = d^j \cdot \frac{s_i}{\sum_{i' \in \Gamma^j} s_{i'}}$ be quantity that captures the ideal fair allocation. The goal is to minimize $\sum_j w^j \sum_{i \in \Gamma^j} \Delta(y_i^j, \delta_i^j)$, where $\Delta(\cdot, \cdot)$ is the *penalty function* that penalizes deviation from fairness, subject to supply and demand constraints. Different norm/distance functions can be used for $\Delta(\cdot, \cdot)$; In the rest of this chapter, we mainly use $\Delta = L_1$. L_1 distance function is simply computed by summing the distances of the allocation from the ideal fair allocation over all pair of (contact,impression)'s.

For the remainder of this section, a superscript j will always denote a contract and a subscript i will always denote an impression.

5.1.1 Fair allocation with L_1 penalty

In this section we consider the fair allocation problem with the L_1 penalty function. This problem can be formulated as a linear programming problem. Our main result is that we can solve this LP by solving a min-cost flow formulation, i.e., there is a combinatorial solution to the fair allocation problem with L_1 penalty. We also show that we can reconstruct the min-cost flow solution using $O(|I| + |J|)$ space for storing the preprocessed information and the flow can be recomputed in $O(1)$ using this information. We postpone the description of the preprocessing and reconstruction for min-cost flow to Section 5.1.2.

To understand the fair allocation problem with L_1 penalty function, we consider linear programming (LP) formulation of the problem, described below.

$$\min \sum_j w^j \sum_i \left| y_i^j - \delta_i^j \right|, \tag{5.1.1}$$

subject to

$$\begin{aligned} \forall j, \sum_i y_i^j &= d^j \qquad \text{(demand)} \\ \forall i, \sum_j y_i^j &\leq s_i \qquad \text{(supply)} \end{aligned}$$

Next, we describe the combinatorial ways to approach this problem. To simplify the description, from now on, for a given allocation A, we define $\mathsf{unfair}(A) = \sum_j w^j \sum_i \left| A_i^j - \delta_i^j \right|$.

Our approach to combinatorially solve (5.1.1) is based on min-cost flow. First, we show that (5.1.1) can be modeled exactly as a min-cost flow problem. Later in Section 5.1.2, we show that in fact by preprocessing the network flow solution, we can find a compact plan that only stores $O(|J| + |I|)$ values and can reconstruct the asymptotically optimal solution in $O(1)$ time. Also later in Section 5.1.5, we obtain an approximate solution, along with efficient reconstruction, for the L_1 penalty function case, when the demand constraints are "soft."

The flow network, with capacity and cost on each edge, is constructed as a four layer graph G (Figure 5.1). The first and the last layer are the source s and sink t respectively. The second layer represents the set J of contracts, and the third layer stands for the set I of impressions. Source s has an edge to each contract $j \in J$ in the second layer, with capacity d^j and zero cost. Contract $j \in J$ in the second layer is connected to impression $i \in I$ in the third layer iff $i \in \Gamma^j$. In this case, there are two edges between j and i. The first edge has capacity δ_i^j and cost zero and the second edge has capacity $s_i - \delta_i^j$ and cost $2w^j$. Finally each impression $i \in I$ in the third layer is connected to the sink t by an edge with capacity s_i and cost zero.

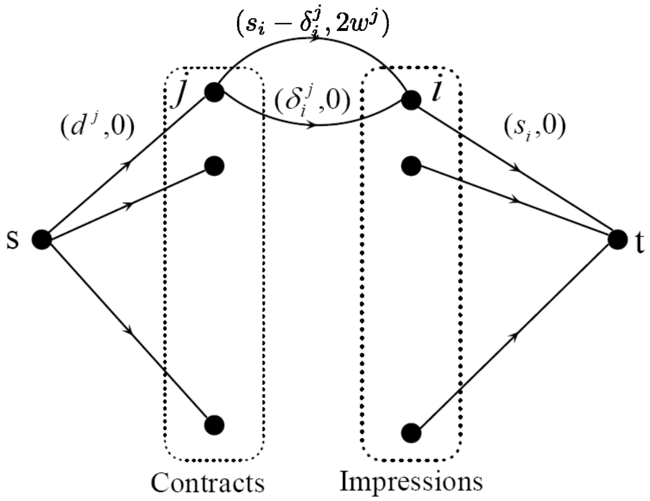


Figure 5.1: The network construction, with (capacity, cost) on the edges.

Theorem 1. *The min-cost s-t flow on G is the solution to (5.1.1).*

Proof. To prove the theorem, we need to show the following. (1) A maximum s-t flow in G is a feasible solution to 5.1.1, provided (5.1.1) has a feasible solution. (2) Any feasible solution to (5.1.1) is a feasible s-t flow in G. (3) Minimizing the cost of the maximum s-t flow in G is equivalent to minimizing the objective in (5.1.1).

First, we show that assuming that (5.1.1) has a feasible solution, a maximum s-t flow in G is a feasible solution to (5.1.1). Note that the capacity constraints on the edges from the third layer to t will guarantee the supply constraint. Also we know that the maximum flow in our network cannot exceed the total demand. Thus all constraints are satisfied. Since we look for the maximum flow, if there is any feasible solution to the LP that can satisfy all the demands and the supply constraint, then that solution would be selected as the maximum flow as well. This means if (5.1.1) has a feasible solution, then any feasible maximum flow in our network is a feasible solution to (5.1.1). Second, it is easy to see that any feasible solution to (5.1.1) is also a feasible s-t flow in G. And last, we argue that the cost of the optimal solution to (5.1.1) is the same as the minimum cost of the maximum flow in G. In other words, we need to show that the costs of the flow are computed correctly in G. Note that the total contribution to the cost of the pairs of impressions and contracts that are over-assigned, i.e., $y_i^j > \delta_i^j$, is half of the total cost. So it is enough to compute twice this cost. Now consider an impression i and a contract j. If $y_i^j > \delta_i^j$, then we can route at most δ_i^j flow through the edge with cost 0 and route $y_i^j - \delta_i^j$ through the edge with cost $2w^j$, which is equal to $2w^j(y_i^j - \delta_i^j)$, which is exactly twice the cost of this pair. □

So we showed that our min-cost flow solution has exactly the same value as (5.1.1). Next, we show that in fact, we can preprocess the min-cost flow solution so that by keeping track of only $O(|J| + |I|)$ values, we can reconstruct the complete flow.

5.1.2 Reconstruction for L_1

Even though the above formulation obtains an optimal solution, as we mentioned earlier, it is not feasible to store the entire allocation information. Storing the representation of the solution is expensive since it uses $O(|I| \cdot |J|)$ space. Ideally, we wish to store just $O(|I| + |J|)$ information (i.e., amortized $O(1)$ per node) that will let us reconstruct the flow along every edge. For practical reasons, we require the reconstruction to be time-efficient.

We consider the L1 formulation and show approximate reconstruction via dual variables for the nodes. First, we write the LP corresponding to the min-cost flow. As we showed before, this LP is equivalent to (5.1.1).

$$\min \sum_j \sum_{i \in B^j} 2w^j x_i^j \tag{5.1.2}$$

subject to

$$\begin{aligned} \forall j, \quad & \sum_i (x_i^j + \overline{x}_i^j) = d^j \\ \forall i, j \in E, \quad & \overline{x}_i^j \leq \delta_i^j \\ \forall i, \quad & \sum_{j \in C_i} (x_i^j + \overline{x}_i^j) \leq s_i \end{aligned}$$

Here, x_i^j denotes the flow along the top edge and $\overline{x}_i^j$ denotes the flow along the bottom edge from j to i in Figure 5.1. The dual for the above LP looks as follows.

$$\max \sum Y^j d^j - \sum Z_i s_i - \sum A_i^j \delta_i^j \tag{5.1.3}$$

subject to

$$\begin{aligned} \forall i, j, Y^j - Z_i - A_i^j &\leq 0 \\ \forall i, j, Y^j - Z_i &\leq 2w^j \end{aligned}$$

Here, x denotes the allocation, where x_i^j is the number of allocated impressions from impression set i to contract j. Since we want to maximize the dual function and the coefficient of A_i^j in the objective function is always negative, we would like to assign the value of A_i^j as small as possible. Considering the existing constraint in the dual that means that $A_i^j = \max(0, Z_i - Y^j)$. So we don't need to keep track of A_i^j variables in the dual. Next, we show how to reconstruct x_i^j variables by only keeping Y^j and Z_i variables and then we show that in fact it is enough just to keep track of Y^j variables.

Since we are considering an optimal solution and because of the complementary slackness, we have three cases for each edge between i and j. First we consider the bottom edges.

1. $Y^j - Z_i < 0$: in this case, we have $\overline{x}_i^j = 0$.

2. $Y^j - Z_i > 0$: in this case, we conclude that $\overline{x}_i^j$ is fully saturated.

3. $Y^j - Z_i = 0$: in this case, we have $Y^j = Z_i$. This is the only case that we can not compute the x_i^j directly from the dual variables. Instead, we make another network consisting only of these edges. The only constraint is the edge capacity constraint. Demand and supplies are updated based on the last two steps.

We have the same scenario for the top edges as well. In the third case, the dual variables do not give us any information on the value of the primal. We call the edges belonging to the third class *slack edges*. Next, we argue that any feasible assignment of flow to slack edges would be a solution. To see this, first notice that the cost of any slack edge (i, j) is exactly to $Y^j - Z_i$. Further, any path from i to j consisting of only slack edges has the same cost. Furthermore, any cycle consisting of slack edges has a cost of 0. Therefore, any feasible maximum flow on the slack edges would constitute a solution. However, this means we have to be able to reconstruct a maximum flow on the slack edges. Thus, in the worst case, reconstructing an arbitrary maximum flow using the dual is no easier than finding a maximum flow from scratch!

The way around this problem is to place additional constraints on the maximum flow to make it unique in a way the we can reconstruct it efficiently.

One way to impose extra constraints is by assigning secondary convex costs to slack edges and then finding a minimum convex cost flow on them. We can then store the Lagrangian multipliers of the convex program and use them to reconstruct the flow; this solution was proposed by Vassilvitskii, Vee, and Shanmugasundaram [50]. We can always do that because the Karush–Kuhn–Tucker (KKT) conditions of the convex program involve both the primal and dual variables and therefore they can always be used to reconstruct the primal. Note that complementary slackness conditions only contain dual variables and do not give any information if an edge is only partially full in the optimal primal.

We provide a completely different solution that does not involve placing secondary costs on slack edges and does not require solving a convex program. We show how to compute a plan for approximate reconstruction (Section 5.1.2.1) and how to use this plan to do the actual reconstruction (Section 5.1.2.2). In Section 5.1.3 we discuss the generic effect of supply and/or demand scaling, which is of interest when supply forecasts, for instance, are not available precisely.

5.1.2.1 Computing the plan

As described earlier, by computing the dual variables of the min-cost flow, we can decide which edges are completely saturated and which ones are completely empty. For the rest of the edges (partially full, a.k.a. slack) the problem was reduced to computing a maximum flow in the new subgraph. Here, we present a way to find a *specific* maximum flow solution that is *easy* to reconstruct. We start by developing some notation and definitions.

For a given node $v \in V$ and a given flow function flow : $E \rightarrow R^+$ on the edges, let

$$\begin{aligned}\text{In}(v) &= \sum_{u:(u,v)\in E} \text{flow}(u,v)\\ \text{Out}(v) &= \sum_{u:(v,u)\in E} \text{flow}(v,u),\\ \text{excess}(v) &= \text{In}(v) - \text{Out}(v)\end{aligned}$$

Definition 2 (ϵ-feasible flow)**.** *For a given flow function* flow$(x,y) : E \rightarrow R^+$*, we call it* ϵ-feasible *iff for any node j in the contract layer,* $0 \leq \text{excess}(j) \leq \epsilon \cdot \text{Out}(j)$ *or equivalently* $\text{Out}(j) \leq \text{In}(j) \leq (1+\epsilon)\,\text{Out}(j)$.

We now define a height function that is crucial to building the plan. The height function height : $V \rightarrow R^+$ simply assigns heights to the nodes. For a given height$(\cdot)$, the flow in each edge $e = (i,j)$ is defined as $\text{flow}(i,j) = \min(1, \max(\text{height}(i) - \text{height}(j), 0)) \cdot c_{i,j}$, where $c_{i,j}$ is the capacity of (i,j) in our network flow instance. For a given height : $V \rightarrow R^+$, $\hat{E}(v) = \delta$ if by changing height(v) to height$(v) + \delta$, we have excess$(v) = 0$. (Note that the value of height$(\cdot)$ for the rest of the nodes is unchanged.)

Next we show that there exists a height function height(v) for which the corresponding flow is ϵ-feasible and it is greater or equal to maximum feasible flow on the slack edges. Note that for a given height function height(v), computing the corresponding flow is trivial just from its definition. The method we use for computing the height function is as follows.

Algorithm 3 (Height)**.**

1. **Initialize:** Initialize height$(s) = 2|J| + 2$ for source s and height$(v) = 0$ for all other nodes v.
2. **Relabel:** For the given function height, find the node v with largest $\hat{E}(v)$. Set height$(v) = \text{height}(v) + \hat{E}(v)$. Update the excess flow values and $\hat{E}$ for the rest of the nodes.
3. **Loop:** Repeat step 2 until $\hat{E}(v) \leq \varepsilon = \frac{\epsilon}{2|I|(1+\epsilon)}$ for all $v \in V$.

We show that the corresponding flow for height(x) after termination of the algorithm is ϵ-feasible and also it is at least as large as the maximum feasible flow. We also show that the algorithm converges in time $O((|I|+|J|)\frac{|I||J|}{\epsilon})$. First we show some properties of the algorithm.

Lemma 5.1.1. *After the termination of the algorithm, the set of edges between s and contract nodes are all fully saturated or in other words for any contract node j,* $\text{height}(j) \leq \text{height}(s) - 1$.

Proof. First we partition the nodes into two sets X and Y based on whether they are reachable from s on the residual flow graph. We first claim that $Y \neq \emptyset$ and $t \in Y$. This follows since there can be no simple path from s to t in the residual graph. Any simple path from s to t is of length at most $2|J| + 1$ and since $\text{height}(s) = 2|J| + 2$, it should be the case that for some edge (u, v) on the path, $\text{height}(u) > \text{height}(v) + 1$ and therefore $\text{flow}(u, v) = c(u, v)$, which means (u, v) is not in the residual graph. Thus, s and t are disconnected.

Since X and Y are non-empty and disjoint, (X, Y) is a cut. Notice that the size of this cut is less than or equal to the cut which consists of edges between s and the contract nodes, because we may have some excess flow on nodes in X. We can conclude that if the cut (X, Y) is not the same as the cut between s and the contract nodes then the demands of the contracts were not satisfiable to begin with which contradicts out assumption. Therefore, $X = \{s\}$ and $Y = \bar{X}$, so for any contract node j, $\text{height}(j) \leq \text{height}(s) - 1$. □

Lemma 5.1.2. *For all $v \in V$,* $\text{excess}(v)$ *is always non-negative during running the algorithm.*

Proof. First, we can see that at the initialization step, $\text{excess}(v) \geq 0$. We show that after running each step, the property still holds. Suppose the algorithm selected v at a round and relabeled it so after the relabel step $\text{excess}(v) = 0$. For all nodes that has an outgoing edge to v, their outgoing flow is only decreased so their excess will increase. And for all nodes with incoming edges from v, their incoming flows only increase so their excess will increase as well. For the rest of the nodes the excess will not change. □

Theorem 4. *The flow computed by Algorithm 3 is ϵ-feasible.*

Proof. Consider the time when the algorithm has terminated. We know that for each $v \in V$, if we increase its heights by $\hat{E}(v) < \varepsilon$ then $\text{excess}(v)$ becomes 0. Consider a node j from the contract layer at the end of the algorithm. Since we know that all the contracts are satisfiable, the set of edges from s to the contract nodes is the minimum cut and $\text{height}(j) \leq \text{height}(s) - 2$. Therefore, $\text{flow}(s, j) = c(s, j)$ and raising j by ε will not change its incoming flow. Suppose $\text{Out}^{\max}(j)$ is the total capacity of edges going out of j. We know that after termination of the algorithm, $\hat{E}(j) \leq \varepsilon$. So, $\text{excess}(j) \leq \varepsilon \, \text{Out}^{\max}(j)$. Since $\text{excess}(j) = \text{In}(j) - \text{Out}(j)$, we can conclude that $\mathrm{In}(j) - \mathrm{Out}(j) \leq \varepsilon\, \mathrm{Out}^{\max}(j)$. Also note that there can be at most $2|I|$ edges going out of j, each one with a capacity of less than the incoming capacity of j which was $c(s, j)$ which means $\mathrm{Out}^{\max}(j) \leq 2|I|c(s, j)$ and therefore $\mathrm{In}(j) - \mathrm{Out}(j) \leq \varepsilon 2|I|c(s, j)$. On the other hand since $\mathrm{In}(j) = \mathrm{flow}(s, j) = c(s, j)$, we can conclude that $\mathrm{In}(j) - \mathrm{Out}(j) \leq \varepsilon 2|I|\,\mathrm{In}(j)$. By rearranging the terms and replacing ε by $\frac{\epsilon}{2|I|(1+\epsilon)}$ we get $\mathrm{In}(j) \leq \epsilon\, \mathrm{Out}(j)$, which completes the proof. □

Now we now bound the running time of the algorithm that computes the plan.

Lemma 5.1.3. *The algorithm terminates after $O((|I| + |J|)|I| \cdot |J|/\epsilon)$ iterations.*

Proof. First of all, at each iteration, we increase the $\mathrm{height}(v)$ for some node v by at lease ε. Notice that no node will ever go higher than the source s so each node can be relabeled at most $(2|J| + 2)/\varepsilon$ times so the total number of iterations (for all nodes) in the worst case is $O((|I| + |J|)|J|/\varepsilon)$ which is $O((|I| + |J|)|I| \cdot |J|/\epsilon)$. □

5.1.2.2 Reconstruction using the plan

We now describe how to reconstruct the flow using just $\mathrm{height}(v)$. For now we assume that after the termination of the algorithm there is no excess flow on any node (i.e., $\forall v : \mathrm{excess}(v) = 0$). Obviously, because of the way we constructed the height function, the flow of every edge (u, v) is $\mathrm{flow}(u, v) = \min(1, \max(\mathrm{height}(u) - \mathrm{height}(v), 0)) \cdot c_{u,v}$. This would work perfectly well if there was no excess flow on any node. However because of the excess flows, the solution may not be feasible. To fix that we first tweak the demands before computing the height function as we explain next. First we define the notion of ϵ-good input.

Definition 5 (ϵ-good input)**.** *We say an input instance to our problem is* ϵ-good *if there is a feasible assignment of impressions to contracts when we scale up all the demands by a factor of* $1 + \epsilon$.

Theorem 6. *Assuming that we are given any input instance which is* $(2\epsilon + \epsilon^2)$*-good. Consider the following modification of our method.*

1. *Scale up all the demands by a factor of* $(1 + \epsilon)^2$*, compute the height function* $\mathrm{height}(\cdot)$ *as explained in Algorithm 3, and then set the demands back to their original values.*

2. *At reconstruction time, for each contract* j *and impression* i *reconstruct the flow on* (j, i) *using the height function* $\mathrm{height}(j)$ *as before but then scale it down by a factor of* $(1 + \epsilon)$.

Then, the reconstructed solution is feasible which means it strictly satisfies the supply constraints and the demand constraints. Further, it may assign extra impressions to contracts up to $(1+\epsilon)$ *times their demand.*

Proof. First, notice that since the input is $(2\epsilon+\epsilon^2)$-good, we can still satisfy all the demands which means the set of all demand edges is still a minimum cut. Suppose that we have used Algorithm 3 to compute the height functions. After the termination of the algorithm, because the solution is ϵ-feasible, for any contract j we have $\mathrm{Out}(j) \geq \mathrm{In}(j)/(1+\epsilon)$. But notice that we scaled up all the demands by $(1+\epsilon)^2$ at the beginning out height computation algorithm, so $\mathrm{In}(j) = (1+\epsilon)^2 d^j$ where d^j is the original demand of the contract j. Therefore, $\mathrm{Out}(j) \geq (1+\epsilon)d^j$ and clearly if we scale down the flow that we reconstruct for edges going out of j by $(1+\epsilon)$, still the outgoing flow of j is at least as much as d^j which means the demand constraints are satisfied. Using a similar argument for the supply side we can show that for any supply node i the incoming flow of i cannot exceed its supply s_i. □

In summary, the reconstruction method works as follows. The preprocessing stage consists of three main steps.

- Construct the min-cost flow and its dual solution. Represent the dual variable of contract nodes by Y_j and the dual variables of impression nodes by Z_i.
- Remove the edges that are forced to be completely full or completely empty by complementary slackness and update the flow graph accordingly (i.e., update the supplies and demands).
- Scale all the demands by $(1+\epsilon)^2$ and the run Algorithm 3 to compute the heights.

The reconstruction stage operates by examining each edge (i,j) and

- if $Y_j - Z_i < \mathit{cost}(i,j)$, then $\mathrm{flow}(j,i) = 0$;
- if $Y_j - Z_i > \mathit{cost}(i,j)$, then $\mathrm{flow}(j,i) = \mathit{capacity}(i,j)$;
- if $Y_j - Z_i = \mathit{cost}(i,j)$, compute the flow using the height function and scale it down by $1+\epsilon$.

$\mathit{cost}(i,j)$ and $\mathit{capacity}(i,j)$ are coming from the min cost flow network (Figure 5.1)

5.1.3 Effect of supply scaling

Notice in applying Theorem 6, we scale up the demands before running the height algorithm but we use the same supply. There might be also other reasons for scaling up the demands. For example, suppose the we do not know the exact supply of the supply nodes (i.e., the s_i values), but we may have an estimate of each supply node which we call s'_i. For example suppose we know that with probability $\delta \approx 1$, $s_i \geq \frac{s'_i}{1+\epsilon}$. Under, such a circumstance we may want to scale down all the supply estimate s'_i by some factor $1 + \alpha$ (or equivalently scale up all the demand constraints, compute the flow and then scale the flow down by the same factor) to make sure that with high probability we can always meet the supply constraints.

Scaling the demands (or supplies) may affect the value of our objective function. The next result gives an upper bound on the change of the objective function when we scale the demands by an arbitrary factor.

Theorem 7. *For a given input instance which is α-good, and with the optimal objective function value* $\textsc{Opt}$*, if we scale the demands by $1 + \alpha$, then the new optimal value of the objective function* $\textsc{Opt}'$ *is at most $|J|^2\alpha \cdot \max_{j \in J} d^j$ away from* $\textsc{Opt}$ *and that is tight.*

Proof. Consider the flow corresponding to the optimal allocation of the original input (before scaling up the demands). Now, for each contract j one by one, we scale the capacity of the edge (s, j) by $1 + \alpha$. Since the input instance is α-good, we should be able to augment the flow by αd^j which is the amount of increase in the capacity of (s, j). By applying the augmentation we may change the flow of each of the other contract nodes by at most αd^j which means the value of the objective function may increase at most by αd^j. Since there are $|J|$ augmentations and each augmentation may affect the flow of all the other contract nodes in the worst case, the total change in the objective function value is upper bounded by $|J|^2 \alpha \max_{j \in J} d^j$.

We prove the tightness by constructing an example shown in Figure 5.2. Suppose we have n contracts $c_1, \ldots, c_n$ and $n + 1$ types of impressions $b_1, \ldots, b_{n+1}$. Each c_i is interested in buckets b_i and b_{i+1}. Also suppose the original demands of all c_i are 2 and the supply of b_1 is 1 and supplies of $b_2, \ldots, b_{n+1}$ are 2. Now in the optimal solution each contract c_i will receive 1 unit of supply from b_i and 1 unit from b_{i+1}. Now, suppose we scale up the demand of each contract by $1 + \alpha$. Notice that for any $1 \leq t < n$, the total demand of the first t contracts minus the total supply of the first t impression sets is exactly equal to 1 which is equal to the amount of supply that c_k receives from b_{k+1}. After scaling the demand, that total is increased by 2α. It is easy to see that the total change in the objective function value in this case is roughly $\sum_{i=1}^{n} 2i\alpha = 2\frac{n(n+1)}{2}\alpha$ which completes the argument. □

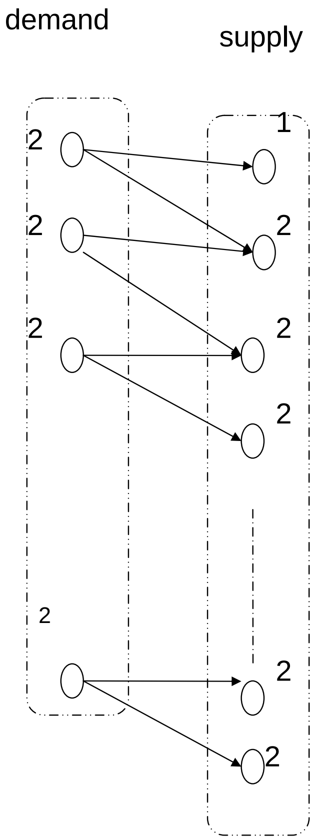


Figure 5.2: Tight example for Theorem 7.

5.1.4 Convex penalty functions

In this section we describe the combinatorial solution for a general convex penalty function. Our solution is based on finding a convex cost flow. For simplicity of exposition, we describe the method for the L_2^2 function; the solution, however, works for any convex distance function.

For L_2^2 penalty function, the quadratic programming formulation of the problem is as follows.

$$\min \sum_j w^j \sum_i (y_i^j - \delta_i^j)^2, \tag{5.1.4}$$

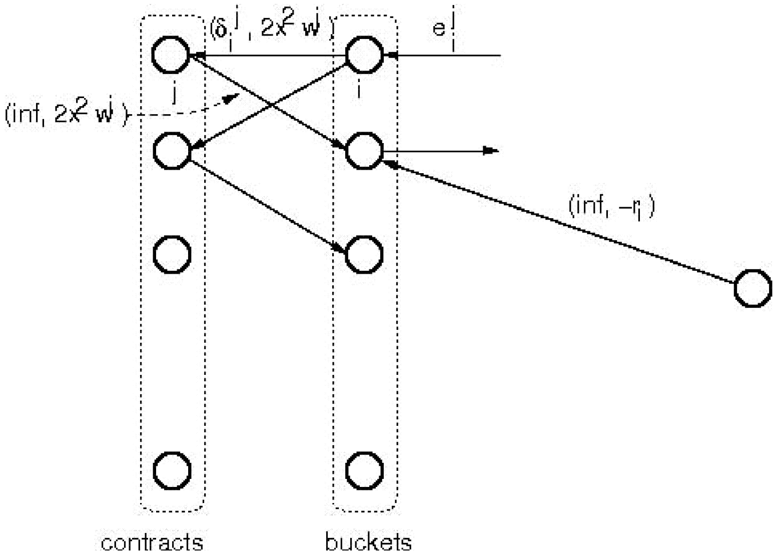


Figure 5.3: The network construction, with (capacity, cost) on the edges for L_2^2.

subject to

$$\begin{aligned}
\forall j, \sum_i y_i^j &= d^j \qquad \text{(demand)} \\
\forall i, \sum_j y_i^j &\leq s_i \qquad \text{(supply)} \\
\forall i, z_i &\leq s_i - \sum_{j \in C_i} y_i^j
\end{aligned}$$

We show how to solve (5.1.4) using a convex cost flow. The flow network, with capacity and cost on each edge, is constructed in the following manner. We create a node for each contract and a node for each impression set. Now compute the completely fair allocation for each contract. In this tentative allocation, some of the buckets could be overfull and some of them could have excess supply. The goal is to reallocate the contracts to buckets in the least expensive way (according to the L_2^2 penalty) such that none of the buckets is overfull. We represent the excess of bucket i by $e_i = s_i - \sum_{j \in C_i} \delta_i^j$. If $e_i \leq 0$, then we consider the bucket node as a supply node and set its supply equal to $-e_i$. If $e_i \geq 0$, then we set the impression set node as a demand node and set its demand equal to e_i. Also we connect each supply impression node to each contract that is interested in that impression, setting the capacity of the edge between the impression and the contract to δ_i^j and the cost of the edge to be $f(x) = 2x^2W^j$, where x is the flow on that edge. See Figure 5.3 for an example.

From our construction, it is easy to see the following.

Lemma 5.1.4. *The solution to network described above is equivalent to the solution to (5.1.4).*

The proof is very similar to the proof described for Theorem 1.

In [50], Vee et al showed that the solution to the L_2^2 can be reconstructed using the Lagrange multipliers of the QP. They showed that, assuming that the lagrange multiplier for supply constraint is p_i and the lagrange multiplier for demand constraint is α^j, the best allocation that minimizes L_2^2 norm and is a feasible solution satisfies the following:

$y_i^j = \max(0, g_i^j(\alpha^j - p_i))$, where $g_i^j(x) = d^j \cdot \frac{s_i}{\sum_{i \in \Gamma^j} |s_i|}(1 - \frac{x}{d^j})$.

That means if we would be able to compute the Lagrange multipliers and store the values, we can reconstruct the real allocation. They also showed that the solution to L_2^2 is unique. Which means that the returned solution by the convex cost function also fits in their argument. Earlier in this section, we showed how to compute y_i^j values. Here, we give a solution sketch on how we can compute lagrange multipliers combinatorially as well. For simplicity, we denote $\phi = d^j \cdot \frac{s_i}{\sum_{i \in \Gamma^j} |s_i|}$ and $\sigma = \frac{s_i}{\sum_{i \in \Gamma^j} |s_i|}$. Rewriting the above equation, we have $y_i^j = \phi - \sigma \cdot (\alpha^j - p_i)$. This means that if we compute y_i^j, we can compute $\alpha^j - p_i$. So first for all pairs (j, i) with $y_i^j > 0$, compute $d_i^j = \alpha^j - p_i$. We now construct a graph with one node for each Lagrange multiplier (in total $|I| + |J|$ nodes) and place an edge between two of them if there is a corresponding $y_i^j > 0$ for them; let the length of the edge be d_i^j. First of all it is easy to see that in each connected component, if we know the lagrange multiplier for one of the vertices (for example p_i), the lagrange multiplier for the rest of the vertices in the same component can be computed as well. So for each connected component, assign some variable p to one of the nodes and compute the distance of all the nodes from that node. This way, the value of the lagrange multipliers in each component can be represented by $p + d$ for which d is already computed and the only variable term is p. So the only remaining question is how to choose p values in each component. Looking back at the Lagrangian for the QP, and replacing y_i^j's and α^j and p_i by their computed values, the objective will be written as a linear function in terms of the selected representing values (p values) for each component. It is enough now to compute the best value for each of them assuming that we want to minimize the Lagrangian objective function.

5.1.5 A greedy solution for L_1 allocation with soft demand

Finally, we present a simple greedy approach for a slightly generalized version of the L_1 penalty function. In this version, we assume that the demand constraint is soft, meaning, it is possible to satisfy a contract partially. (The search engine, however, should pay extra amount per unsatisfied demand, similar to the model used in [22]; we will capture this by a parameter β.) We also show how to preprocess and then reconstruct the greedy solution using $O(|I| + |J|)$ space for storing the preprocessed information and $O(\max_{i \in I} |\Gamma_i|)$ time to recompute the allocation.

We assume each contract has its own weight $w^j = \frac{W^j}{d^j}$ and to implement the soft demand constraint, we assume an amount of $\beta \cdot w^j$ is paid for each impression that cannot be allocated to the contract j, where the factor $\beta \geq 1$. We now present a greedy algorithm and prove that the total cost of its solution is at most $(1 + \beta)/2$ that of the optimal solution. The greedy algorithm proceeds as follows.

Algorithm 8 (Greedy allocation)**.**

1. Start from the contract j with the largest w^j.

2. Give the most fair allocation possible to contract j.

3. Repeat until all contracts are considered.

We now show that this algorithm obtains an approximation to the optimum.

Lemma 5.1.5. *The greedy allocation is a $(1+\beta)/2$-approximation for the L_1 penalty function for the soft demand case with factor β.*

Proof. The proof is based on charging. We start by defining some notation. As usual, for a given allocation A, let A_i^j be the number of allocated impressions from bucket i to contract j. In an allocation A, we call a contract j on bucket i as *under-represented* if $A_i^j < \delta_i^j$; let $\mathsf{under}_i^j(A) = \max(0, \delta_i^j - A_i^j)$. Similarly, we call j *over-represented* on i if $A_i^j > \delta_i^j$ and let $\mathsf{over}_i^j(A) = \max(0, A_i^j - \delta_i^j)$. Let OPT denote the optimal allocation and GREEDY denote the greedy allocation.

Now we make the following claim: in any allocation A, we have $\mathsf{unfair}(A) \leq 2\sum_{i\in I, j\in J} \mathsf{under}_i^j(A)$. The claim holds because for any contract j, $\sum_{i\in I} \mathsf{over}_i^j(A) \leq \sum_{i\in I} \mathsf{under}_i^j(A)$, where the inequality changes to equality when d^j is completely satisfied in A. In addition, $\mathsf{unfair}(A) = \sum_{i\in I, j\in J}(\mathsf{under}_i^j(A) + \mathsf{over}_i^j(A))$.

This means that it is enough to consider the under-represented contracts and lose only a factor of two in OPT. Also, since the allocation is greedy, the amount of under-fairness on each impression in greedy is lower than any other allocation. I.e., $\sum_{i\in I,j\in J} \mathsf{under}_i^j(\textsc{Greedy}) \le \sum_{i\in I,j\in J} \mathsf{under}_i^j(A)$ for any allocation A including $\textsc{Opt}$.

Now, let us consider the total cost of under-fairness on bucket i in the greedy solution, $\mathsf{under}_i(\textsc{Greedy}) = \sum_{j\in\Gamma_i} \mathsf{under}_i^j(\textsc{Greedy})$. Even if we do not accommodate these impressions in any other bucket and pay the β factor instead of allocating them, the total amount would be at most $\beta \cdot \mathsf{under}_i(\textsc{Greedy})$. Now the total value of the objective function for the greedy solution is at most $(\beta+1)\sum_{i\in I} \mathsf{under}_i(\textsc{Greedy})$. From the earlier argument, we know that $\sum_{i\in I} \mathsf{under}_i(\textsc{Greedy}) \le \textsc{Opt}/2$. These imply that the greedy solution is a $(\beta+1)/2$ approximation for the L_1 penalty with soft demand. □

5.1.5.1 Reconstructing the greedy solution

Next, we show how we can reconstruct the greedy solution by storing only $O(|J|)$ preprocessed information. The running time for reconstructing the allocation based on stored information is $O(\max_{i\in I}|\Gamma_i|)$. In the preprocessing phase, we compute the greedy allocation as described in Algorithm 8. The stored information for each contract j is $\sum_{i\in\Gamma^j} \mathsf{over}_i^j(\textsc{Greedy})$. The reconstruction is as follows:

Algorithm 9 (Greedy reconstruction)**.** When a new impression from bucket i arrives,

1. For all contracts j, set $\mathsf{over}^j = \sum_{i\in\Gamma^j} \mathsf{over}_i^j(\textsc{Greedy})$.

2. Consider the set of all contracts $j \in \Gamma_i$ that are still under-represented. If this set is not empty, assign the new impression to the contract in this set with largest w^j.

3. Otherwise, consider all contracts $j \in \Gamma_i$ that their over^j is still positive. Assign the new impression to the contract with the largest w^j in this set. Update $\mathsf{over}^j = \mathsf{over}^j - 1$.

As described earlier, in this method we need to keep track of only one variable for each contract. Also at each impression arrival, in the worst case we have $O(\max_{i\in I}|\Gamma_i|)$ processing time. With similar argument given in Lemma 5.1.5, we can show that the computed solution here is also $\frac{1+\beta}{2}$ approximation for the fair allocation with soft demands.

Chapter 6
Guaranteed Delivery Advertisement: Online Ad Allocation

The second problem related to guaranteed delivery advertisement that we consider in this chapter, focuses on to the question of which advertisers to sign a contract with to maximize the total obtained revenue.

It turns out that maximizing revenue in this problem is NP-hard. A special simplified instance of our problem, called "Resource Allocation", can be approximated to within a constant factor $(1/2-\varepsilon)$ efficiently (see [14]). *Resource Allocation* problem can be defined as follows: Suppose we want to schedule a set of jobs so that the total profit of scheduled jobs is maximized. Jobs are arriving at different days and will be active for some specified duration. However the total number of available resources everyday is bounded. The constraint is that each job needs a specified number of resources everyday. we have R resources available everyday. The objective is to find a feasible schedule so that the obtained profit from the scheduled jobs is maximized.

The setting that is used for resource allocation is equivalent to restricting our problem to the following problem: advertisers are only interested in satisfying their branding constraints, meaning that they want their ad to be shown for a specific period of time and requiring a fixed amount of impressions per day. In this setting, if we are interested in maximizing social welfare (or revenue), our problem becomes identical to the resource allocation problem. However, the above results on resource allocation hold only for the offline setting (see [14]). Nevertheless, in a practical setting, the advertisers arrive online. In this chapter, we consider the online variant of this problem.

6.1 Online Setting

In *Guaranteed Delivery* advertisement as in the *Non Guaranteed Delivery*, advertisers arrive online, contracts are negotiated at different times, and the decision to accept or reject a proposed contract is done online. In this section, we try to address the problem of what set of advertisers to accept in the online setting for *Guaranteed Delivery*.

The structure of the rest of this chapter is as follows: In Subsection 6.1.1 we show that if contracts can be dropped with no penalty and no prior is known on the distribution of the arriving contracts (i.e. in an adversary setting), then even the case of allocating one impression per round cannot be approximated to within any constant ratio. In a more restricted setting where impressions are valued similarly by all advertisers, using results from [51] it can be shown that an approximation ratio of 4 is optimal.

Motivated by the setting and results by Woeginger [51], in Subsection 6.1.2 we consider a setting where n resources are available and contracts arrive online. We propose a greedy algorithm such that, for a class of valuations of the contracts similar to that studied by Woeginger [51], achieves a competitive ratio of at most $4n$, thus generalizing the result by Woeginger[51] (where $n = 1$ and the competitive ratio obtained was 4). We also present a matching lower bound for the competitive ratio of n, thus showing that our algorithm is almost tight.

In subsection 6.1.3 we simulate the performance of our algorithm against real data. Surprisingly, we find that our algorithm performs within only a few percentage points of the optimal omniscient offline algorithm.

6.1.1 Definitions and Settings

Here we present the setting used by Woeginger [51] and some of its results. Assume we are interested in the following problem. Advertisers arrive online and request a contract. Upon arrival, advertiser k reveals all its information (t_k, τ_k, q_k, b_k) and requests a contract in which:

- t_k**:** Stands for the start day of the period that the advertiser is interested in.
- τ_k**:** Stands for the duration of the period.
- q_k**:** Stands for the number of impression that the advertiser wants to receive during his desired period.

We assume that advertiser A_k arrives at time t_k. The publisher then decides on the spot to accept or reject the contract.

In the setting considered in [51] it was assumed that

- $n = 1$ (only one impression per round is available)
- $q_k = 1$ (all of the supply is exhausted by one advertiser at any given time)
- payment is received once the contract is fully completed.
- contracts can be dropped without penalty (i.e. the publisher can accept a contract and drop it later if a better one arrives)

The goal is to maximize social welfare (or revenue) under *adversarial arrival* of advertisers. It is clear that no online algorithm can have a bounded approximation ratio without further assumptions, thus there is an extra assumption that links the duration of the contract τ to its value b (which look a lot like convexity).

Under such assumptions the authors give a 4 approximation algorithm. It is interesting to note that the approximation ratio is tight, and that the algorithm is very simple (greedy algorithm on contract value).

6.1.2 Online Algorithm

In [51] the assumption is that $n_t = 1, 1 \leq t \leq T$. However it is not a valid assumption for our problem. The basic model is the same: Advertisers arrive online and request a contract. Upon arrival, advertiser k reveals all its information $(t_k, \tau_k, q_k, b_k, V_k)$ and requests a contract. We assume that advertiser A_k arrives at time t_k. The publisher then decides on the spot to accept or reject the contract.

However the constraints in our problem, should be relaxed as follows:

- We assume $n \geq 1$ we can have more than one resource per day but it is the same for all days.
- $q_k \leq n$ and it is possible to have more than one active advertiser per round however their total demand should not exceed n.
- Payment is received once the contract is fully completed.
- Contracts can be dropped without penalty (i.e, the publisher can accept a contract and drop it later if a better one arrives)

The goal is to maximize social welfare (or revenue) under an *adversarial arrival* of advertisers. We call this problem with the properties defined above *Online Resource Allocation Problem*. By an argument similar to that given in [51], it is clear that no online algorithm can have a bounded approximation ratio without further assumptions. In our argument, we use the following assumption which is very similar to the given constraint in [51] but defined over $q.\tau$. Assuming that $b_k = f(\tau_k.q_k)$ or in other words, the total profit from scheduling contract k, is a function f of the total size of the contract. the requirement is that function f is a C-benevolent function. We say f is C-benevolent if we have:

1 $f(0) = 0$ and $f(p) > 0$ for all $p > 0$.

2 for $0 < \epsilon \leq p_1 \leq p_2 \Rightarrow f(p_1) + f(p_2) \leq f(p_1 - \epsilon) + f(p_2 + \epsilon)$.

Under these assumptions, we will show that for arbitrary sequence of advertisers with C-benevolent profit functions, there is no online algorithm that can have a worst case ratio smaller than n. We then present an online greedy algorithm with a worst case analysis for C-benevolent profit functions, and we show that our algorithm is only a constant factor away from the best possible online algorithm.

6.1.2.1 Lower Bounds

In this section, we prove that there is no online algorithm with worst case ratio smaller than n for any C-benevolent profit function.

Theorem 1. *There is no deterministic online algorithm for* Online Resource Allocation problem *with a worst case ratio smaller than n.*

Proof. Before describing the strategy of the adversary, we define the two types of demands (contracts) that will be used by the sequence of advertisers submitted by the adversary. For the rest of this section, we use contracts and advertisers interchangeably.

Wide: An advertiser A_i is called *Wide* if $\tau_i = n$ and also $q_i = n$ meaning that they want n impressions per day for a duration of n days.

Long: An advertiser A_i is called *Long* if $\tau_i = n^2$ and $q_i = 1$ meaning that they want 1 impressions per day for a duration of n^2.

Next, we describe the strategy of the adversary against any given online algorithm and also give an analysis to prove the lower bound of n. The strategy of the adversary against any given Heuristic H is as follows:

- Start the sequence of contracts by sending a *Wide* contract at each day.
- If H selects a *Wide* contract, then the adversary starts sending one *Long* contracts per day during the active period of the *Wide* selected contract as long as it stays selected by H.
- If H selects a *Long* contract, the adversary starts sending *Wide* contracts, during the active period of the *Long* contract currently chosen by H as long as it stays active. *Wide* Contracts are submitted as soon as the previous *Wide* Contract sent by the adversary is finished.
- The adversary will stop sending any more contracts if he either sends n *Wide* contracts or n *Long* contracts or the worst case ratio is already n.

1. If the new coming contract does not have conflict with any currently scheduled contracts, schedule it.
2. Otherwise, find[a] the set of contracts with the minimum total profit that if we drop from the schedule then the new coming contract can be scheduled. If the profit of the new contract is more than twice the total profit of these contracts schedule it.
3. In the rest of cases, drop the new contract.

[a] This involves running an algorithm for a knapsack like problem on the accepted set of contracts.

Figure 6.1: ORA

If H keeps the first *Wide* contract, then at day n the worst case ratio is already n. Now suppose H switches to the *Long* contract. Again if H keeps this contract, it will lose at day n^2 so H should switch to a *Wide* contract at some point. We can see that at the end of each round, H still only keeps one contract and also after each switch, the adversary will send at least one contract. It is clear that if H holds on to one contract and stops switching it will lose a factor of n by the finishing time of that contract. Now look at the solution after $2n$ switches. Using the given facts above and the pigeon hole principle, we can conclude that we have n contracts of the same category. Also we know that *Wide* contracts don't have conflicts with each other at all and also any set of size less than or equal to n of *Long* contracts can be scheduled together without any conflict. That means that at this point, the optimal offline solution will exceed n^3 however the solution picked by H is n^2 which gives us the desired worst case ratio for any H. □

6.1.2.2 The online Algorithm

In this section, we present the algorithm:

Next, we show that in the worst case, the ratio of *Online Resource Allocation Algorithm* is $8n$.

Lemma 6.1.1. *The Algorithm* ORA *has the worst case ratio of at most* $8n + 2$ *on C-benevolent Profit functions.*

Proof. To make the analysis more clear, we first categorize all the contracts to three groups simply based by how they are dealt with by algorithm ORA.

Final: are the contracts that are selected, and retained until the end by Algorithm *ORA*.

Dropped: are the contracts that are (initially) selected by the algorithm but dropped later on in favor of other contracts.

Ignored: are the ones that are not selected by Algorithm *ORA* at all.

We first show that the total profit of *Dropped* contracts is at most equal to the total profit of *Final* contracts. We also show that the total profit of the *Ignored* contracts belonging to the optimal solution is at most $4n$ times of the sum of *Dropped* and *Final* contracts.

Lemma 6.1.2. *The total profit of the* Dropped *contracts is at most equal to the total profit of the* Final *contracts.*

Proof. Consider all the *Dropped* and *Final* contracts. Create one node corresponding to each of these contracts. If a contract is dropped because of another contract, put a directed edge between them. It can be seen that the outdegree of each node corresponding to a *Dropped* contract is 1 and it is 0 for all the *Final* contracts. The structure of this graph is a directed forest rooted at *Final* contracts. We show that the total profit of all the internal nodes in each tree is at most equal to the profit of its root. The proof is by induction. Assume that for all trees of height less than h the total profit of the all the internal nodes in the subtree are less than the profit of the root. Now consider a tree of height h. We know by the way we put the edges that the profit of the root is at least twice the total profit of all the roots of the subtrees. But by induction, we also know that the total profit of the vertices of the subtrees are greater or equal to the profit of the nodes in the subtrees so that means that the profit of root of tree is greater or equal to the profit of all the nodes in the tree. Each *Dropped* contract belongs to exactly one tree since it should have exactly one outgoing edge which completes the proof. □

Next, we show that the total profit of *Ignored* contracts belonging to the optimal solution is at most $4n$ times of the total profit of *Final* and *Dropped* contracts.

Lemma 6.1.3. *The total profit of* Ignored *contracts in OPT are at most* $4n$ *times the total profit of* Final *and* Dropped *contracts assuming the profit of each contract is computed by a C-benevolent function of the size of the contract.*

Proof. Again, we use a charging method. We define the allocation for a contract and a heuristic as follows:

Alloc(A_i, H): Assuming that we named the items available at each day by $I = i_1, \ldots, i_n$, we define *Alloc*(A_i, H) as the exact set of items assigned to A_i by H at each day. Without loss of generality we assume that we have one copy available from i_t each day.

Without loss of generality in the rest of the proof we make the following assumptions:

- We assume that for a given algorithm H and a contract A_i, the allocations can be set in a way that as long as A_i is not dropped *Alloc*(A_i, H) does not change for the duration of each contract.
- The assigned items to each contract are staying the same during its active period. However the assigned resource might or might not be consecutive.

Consider an *Ignored* contract A_i that belongs to the optimal solution. We define the set $Conf(A_i)$ as follows:

$Conf(A_i)$: Consider *Alloc*(A_i, OPT). Also consider all A_j contracts that are either *Final* or *Dropped* and also $t_j \leq t_i$ and $Alloc(A_j, ORA) \bigcap Alloc(A_i, OPT) \neq \emptyset$. Call it $PConf(A_i)$. Now define $Conf(A_i) \subseteq PConf(A_i)$ as follows:

- Initialize $Conf(A_i) = PConf(A_i)$.
- Sort $A_j \in Conf(A_i)$ in the increasing order of b_j.
- If the $b_i < \sum_{A_j \in Conf(A_i)} b_j$ and $|Conf(A_i)| > 1$, remove the contract with minimum b_j from $Conf(A_i)$.
- Repeat until either $|Conf(A_i)| = 1$
  or $b_i \geq \sum_{A_j \in Conf(A_i)} b_j$.

Defining the $Conf(A_i)$ as above, we can guarantee that if $|Conf(A_i) > 1|$ then $\sum_{A_j \in Conf(A_i)} b_j \leq b_i \leq 2. \sum_{A_j \in Conf(A_i)} b_j$. Next, we replace A_i with $|Conf(A_i)|$ virtual copies $A^v_{i,j}$ corresponding to each $A_j \in Conf(A_i)$ and set virtual profit $b^v_{i,j} =\leq 2.b_j$ so that $\sum_{A_j \in Conf(A_i)} b^v_{i,j} = b_i$. So $A^v_{i,j} = (t_i, \tau_i, q_i, b^v_{i,j}, 0)$.It is easy to show that it is always possible to do this. Now we can show the following lemma:

Lemma 6.1.4. *Defining the price per unit of a contract(virtual contract) A_i by $ppu(A_i) = \frac{b_i}{\tau_i \cdot q_i}$, then $ppu(A^v_{i,j} \leq 2.ppu(A_j)$ for all $A_j \in Conf(A_i)$.*

Proof. If $|Conf(A_i)| = 1$ and $b_i \leq b_j$ where $A_j \in Conf(A_i)$, then since profit function is C-benevolent, we can conclude that $ppu(A^v_{i,j}) = ppu(A_i) \leq ppu(A_j)$. In the rest of the cases we can argue as follows:

Since the profit function is monotonically non decreasing, with the way we defined $Conf(A_i)$ we can conclude that $\tau_i.q_i \geq \tau_j.q_j \ \forall A_j \in Conf(A_i)$. Now considering $A^v_{i,j}$ we know that $b^v_{i,j} \leq 2.b_j$. So we have:

$$\begin{aligned} ppu(A^v_{i,j}) &= \frac{b^v_{i,j}}{\tau_i.q_i} && (6.1.1)\\ &\leq \frac{2.b_j}{\tau_i.q_i} && (6.1.2)\\ &\leq \frac{2.b_j}{\tau_j.q_j} && (6.1.3)\\ &= 2.ppu(A_j) && (6.1.4)\\ & && (6.1.5) \end{aligned}$$

□

Next, we partition $Conf(A_i)$ into two subsets:

$Tail(A_i)$: $Tail(A_i) \subseteq Conf(A_i)$ is the set of all $A_j \in Conf(A_i)$ that has overlap with *Alloc*(A_i, OPT) at their finishing time in *Alloc*(A_j, ORA).

$Mid(A_i)$: $Mid(A_i) \subseteq Conf(A_i)$ contains all the rest of contracts in $Conf(A_i)$.

Now the method we are using for charging is as follows:

- For each $A_j \in Conf(A_i)$ do the following:
  - If $A_j \in Tail(A_i)$ then charge A_j twice the profit of $A^v_{i,j}$ (6.1.6)
  - Otherwise, for each day t that A_j and A_i have overlap in *Alloc*(A_i, OPT) and *Alloc*(A_j, ORA), charge one of the common items in A_j, $2n$ times the *ppu*$(A^v_{i,j})$. (6.1.7)

We show that the profit of each $A^v_{i,j}$ is charged completely at least once to some $A_j \in Conf(A_i)$. Also, we will show that each A_j is not charged more than $4.n$ times considering all the *Ignored* contracts in the optimal solution.

We first show that:

Lemma 6.1.5. *$\forall A_j \in Conf(A_i)$, By the charging method described above, the total profit of $A^v_{i,j}$ is charged to A_j.*

Proof. With the way we defined $A^v_{i,j}$, we know that $b^v_{i,j} \leq 2.b_j$. Now if $A_j \in Tail(A_i)$, we charge its profit twice which will directly cover $b^v_{i,j}$. Now consider the situation where $A_j \in Mid(A_i)$. Since $A_j \in Mid(A_i)$ we know that $t_i \geq t_j$ and $t_i + \tau_i \leq t_j + \tau_j$. We conclude that if $Alloc(A_i, OPT)$ and $Alloc(A_j, ORA)$ are overlapping in at least one day, they should have an overlap on every day t where $t_i \leq t \leq t_i + \tau_i$. That means in 6.1.7, we charge the profit of one unit of A_j $2n$ times for t satisfying $t_i \leq t \leq t_i + \tau_i$.Also we know that $b^v_{i,j} = ppu(A^v_{i,j}).\tau_i.q_i \leq 2.ppu(A_j).\tau_i.n$. So we can conclude that the total profit of $A^v_{i,j}$ is completely covered in our charging scheme. □

Next, we show that:

Lemma 6.1.6. *Each contract that belongs to* Final *or* Dropped *set, is charged at most $4n$ times.*

Proof. Consider a contract A_j that belongs to *Final* or *dropped*. It can be shown that the total number of *ignored* contracts in the optimal solution that contain A_j in their $Tail(A_i)$ are at most n. The reason is that A_j can occupy at most n resources on its finishing day and since we are considering a feasible fixed allocation of the optimal solution, each of these resource can belong to at most one contract in the optimal solution and in total at most n contracts in OPT have overlap with A_j at its finishing time. Also, each resource of A_j will be charged once and for an amount equal to $2n$ times the price per unit of that resource because the charged resource should belong to the overlap of A_j and A_i and A_i belongs to the optimal solution and we consider a fixed feasible allocation of OPT, so no other A_k where $k \neq i$, can have overlap with A_j at the same resource. Therefore in 6.1.7, each unit will be charged at most $2n$ times. Putting all these together, we can conclude that each A_j in either *Final* or *Dropped* is charged at most $4n$ times. □

The final goal is to compare the profit of *Final* contracts with the total profit of optimal. Partition the total profit of OPT to $OPT_f + OPT_d + OPT_i$ based on the category that they belong to in ORA. Also call the profit of *Final* contracts P_f and profit of *dropped* contracts, P_d. We know that $OPT_f \leq P_f$, $OPT_d \leq P_d \leq P_f$ and finally $P_i \leq 4n.(P_f + P_d) \leq 8n.P_f$. So $OPT_f + OPT_d + OPT_i \leq (8n + 2)P_f$ which completes the proof. □

□

6.1.3 Simulation

In this subsection we test the performance of our algorithm on real data derived from the Yahoo! display advertisement business. To do so, we selected four different types of impressions and considered all contracts that could be satisfied using those type of impressions. The types of impressions represent different sets of *properties* (e.g. mail, Finance, etc.) and *positions* (e.g. top, bottom, side). We then partitioned the business period into rounds. This partition determined the value of T for each dataset. Data is derived from proprietary but real contract data from 2007.

The simulation methodology is the following. We first assumed that all advertisers were interested *only* in being delivered the minimum number of impressions. Next, we parameterized the simulation by n, the number of impressions available per round, and ran the simulation for 5 distinct values of n. In order to calculate the optimal revenue, we attempted to solve the following integer program:

$$\max_x \sum_{k=1}^{K} b_k x_k \text{ subject to} \tag{6.1.8}$$

$$\sum_{k=1}^{K} q_k a_{t,k} x_k \leq n, \text{ for all } 1 \leq t \leq T \tag{6.1.9}$$

$$x \in \{0,1\}^K \tag{6.1.10}$$

where $a_{t,k} = 1$ if advertiser k is interested in impressions during round t, and $a_{t,k} = 0$ otherwise. The variable $x_k = 1$ indicates that a contract will be sold to advertised A_k. Note that the number of variables is equal to K, and that the number of constraints is given by T, the number of periods.

Since the number of contracts may be large, it is important to note that the ability to solve the previous integer program depends both on the number of constraints (as given by Inequalities 6.1.9) active at optimality, and on the total number of contracts selected at an optimal allocation. We capture both effects by varying the number of impressions available per round. If n is large enough, all advertisers can book a contract. As n decreases, the number of rounds where a potential contention exists increases, and the solution becomes “lumpier”.

In the course of solving the integer program, we also obtained the solution to the LP relaxation, where the Inequalities 6.1.10 are replaced with

$$0 \leq x_k \leq 1, \text{ for all } 1 \leq k \leq K \tag{6.1.11}$$

Since for some of the datasets the integer program produced a *worse solution* than that of our algorithm, we report the results of our algorithm as compared to the solution obtained by the LP relaxation. We note that we used default settings for the IP solver, which might account for its inability to handle some of the datasets. The results of our simulations are summarized in Table1 6.1-6.4.

Table 6.1: Results for the first dataset

n	Number Booked	Performance loss
n_1	2450	0
n_2	1709	3.71
n_3	803	14.2
n_4	88	10.5
n_5	33	8.36

Table 6.2: Results for the second dataset

n	Number Booked	Performance loss
n_1	1581	0
n_2	1070	10.4
n_3	551	12.6
n_4	169	14.4
n_5	111	37.9

Table 6.3: Results for the third dataset

n	Number Booked	Performance loss
n_1	9169	0
n_2	9124	0.05
n_3	8323	0.52
n_4	6615	0.84
n_5	1285	4.99

The parameters for each dataset are summarized in Table 6.5.

The performance loss is given in percentage points with respect to the LP solution. Note that our algorithm performs orders of magnitude better than the guaranteed performance (note that even in the case on $n = 1$, the performance guarantee would be of 25%, or a performance loss of 75%!). This indicates that the instances typically

Table 6.4: Results for the fourth dataset

n	Number Booked	Performance loss
n_1	8826	0
n_2	6355	20.0
n_3	4603	23.6
n_4	850	8.09
n_5	436	6.72

Table 6.5: Parameters of the datasets

Parameters	Dataset1	Dataset2	Dataset3	Dataset4
n_1	5M	5M	1B	500M
n_2	1M	1M	500M	100M
n_3	500k	500k	100M	50M
n_4	100k	100k	50M	10M
n_5	50k	50k	10M	5M
# contracts	2450	1581	9169	8826
T	12	90	90	90

encountered in real setting are far from those necessary to make our algorithm perform poorly.

In Table 6.6 we compare the performance of the IP solution to that of the LP solution. Note that for datasets two to four, the performance of the integer programming solution was sometimes below that of our solution, and in some instances was not even available. We used the open source branch and cut code Cbc from the COIN-OR repository [41], with default parameter settings, but with the attempted addition of the probing, Gomory, knapsack, redsplit and clique cuts. In general, the number of cuts obtained was small. The branch and bound algorithm was run until either the 10th integer solution was obtained, or twenty minutes have elapsed.

Table 6.6: Performance of the Integer Program solution

n	Dataset1	Dataset2	Dataset3	Dataset4
n_1	0	0	0	0
n_2	0.04	5.01	0.02	N/A
n_3	0.10	8.53	17.7	N/A
n_4	1.77	36.7	8.27	91.1
n_5	2.80	32.1	9.54	N/A

Chapter 7
Conclusion and Future Work

Online advertising is an emerging field in computer science and it is also the main revenue resource for large search engine companies. In this thesis, we focused on online advertising and described different ways of online advertising. We considered some problems related to this way of advertising and suggested some solutions. We conclude this thesis by describing the summary of our results and some of the future directions to pursue research in this area. We looked at four main problems related to online advertising and presented the solution to each problem. In these problems, we considered constraints that are forced by the real system requirements such as space(memory) constraint, speed and simplicity, and the same time we tried to maximize objective functions defined for each problem. In *Non Guaranteed Delivery* we looked at two problems.

- Online Ad Allocation Problem
- Query Rewriting Problem

We start by generalizing the notion of *Submodularity* over set functions to a more broad class of functions (i.e., *Sequence Functions*) and showed that the result of Nemhauser et al [46] still holds for *Sequence Submodular* functions. As we saw in this thesis, this result is useful in analyzing the performance of a greedy approach for online problems. We used this property as a tool to analyze *Online Ad Allocation* problem as well as the weighted version of *Query Rewriting* problem. We showed that if the queries are coming from a fixed but unknown distribution during time, the greedy approach obtains at least $(1 - \frac{1}{e})$ of the profit in expectation. This is a good news since greedy algorithms are simple and fast and this result shows that their profit is also good. The second problem that we considered was the *query rewriting problem*. *Query rewriting* was used in Yahoo! as a tool to help retrieving relevant ads. However, the original design was only based on the relevance of the original query to the rewrites (similar to the ones for information retrieval) and new requirements for advertisement was not included. We considered this problem from the advertising point of view, defined a new set of constraints and objectives and then developed a solution based on this new model. Also, for the first time, we modeled a global variant of the problem by considering all queries together and defined the appropriate constraints and requirements for the global variant. We considered two models:

Cardinality Version: In this variant, 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 model, we proposed 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.

Weighted Version: In the second variant (called the *weighted version*), we modeled the constraints of an ad's budget. We assumed that the traffic for a query for a fixed period of time is known. The goal here was 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 was that an ad can only contribute benefit for traffic up to its budget. For this version, we proposed another greedy algorithm with an approximation ratio of $1 - \frac{1}{e^{1-\frac{1}{e}}}$. In the analysis, we used the sequence submodularity property.

Research on *Guaranteed Delivery Advertisement* is getting more popular. In this category as we mentioned before, advertisers are more interested in branding. As a result, they need their ad to be shown to a good set of users that are related to advertisers desired set of keywords. As a result, they want that the search engine *guarantees* to allocate their required number of impressions to them for their requested time period. Two main questions in this area are: (a) to decide which set of advertisers to accept and sign a contract with. (b) after accepting a set of advertisers as contractors, which allocation is the most favorable one for contractors. We looked at the following two problems:

Online Ad Allocation for Display Advertisement The model we considered is as follows: Advertisers arrive online and request a contract. Upon arrival, advertiser k reveals all her information including the number of impressions she is interested per day and the start and the end of the time period that she wants her ads to be displayed. We assumed that the payment is according to the total number of impressions that will be allocated to an advertiser (if and only if her total demand will be satisfied). The publisher then decides on the spot to accept or reject the contract. We also assume that the total number of available impressions per day is fixed. Contracts can be dropped without penalty (i.e., the publisher can accept a contract and drop it later if a better one arrives). However in this case publisher cannot charge the advertiser anything if the contract is dropped. The goal is to maximize social welfare (or revenue) under *adversarial arrival* of advertisers. Under these assumptions, we showed that for an arbitrary sequence of advertisers, there is no online algorithm that can have a worst case ratio smaller than n where n is the total number of available impressions per day. We then presented an online greedy algorithm with a worst case analysis and we showed that the obtained profit from our algorithm is only a constant factor worse than the best possible online algorithm. Since the lower bound on the best possible algorithm in the adversary setting is quite pessimistic, we simulated the performance of our greedy algorithm on actual data from Yahoo!'s display advertisement business. We found that our algorithm performs very well in practice as the observed performance was quite close to the optimal, as opposed to the worst case lower bound.

Fair Contract Allocation Given a set of impressions (i.e., the supply) and the number of available impressions and contracts with their desired number of impressions, how to find a feasible assignment of impressions to contracts that is as *fair* as possible? Answering this question involves formulating what fairness precisely means in this context. And, given the large number of advertisers (typically, in the hundreds of thousands) and the astronomical number of impressions (typically, in the hundreds of millions) in an online setting, we desire an allocation algorithm that is practical and combinatorial and whose allocation can be stored *succinctly*, ideally, using space linear in the number of impressions and contracts as opposed to the naive quadratic storage. Of course, this succinct representation should let us reconstruct the allocation along every pair of contract and impression bucket in a time-efficient manner. The price of each impression is computed based on the historical data. We considered the general problem of fair allocation in a bipartite supply-demand setting. Our formulation is combinatorial and captures the notion of deviation from fairness by a natural and general form of a penalty function. For the case of L_1 penalty functions we obtained a simple combinatorial algorithm for the fair allocation problem. By L_1 penalty function, we mean that we want to find a feasible allocation of impressions to contracts while minimizing the absolute distance of this allocation from the most representative allocation for each contract. Our solution was based on solving a min-cost flow problem on a bipartite graph, which has been done very efficiently. We also showed how to precompute and store a *linear* amount of information such that the allocation along *any* edge in the bipartite graph can be approximately answered in *constant* time, under mild assumptions on the input instances. This space-efficient reconstruction method might be of independent interest in contexts beyond fair allocation.

We also proved two additional properties of our formulation. First is robustness, where we show how to upper bound the performance loss when the supply estimates are only approximately known. Second is extensibility, where we showed an even simpler greedy approximation algorithm when some of the demand constraints are relaxed.

Finally, we extended our combinatorial solution to any convex function.

7.1 Future Work

In the context of online algorithms, one approach is to design algorithms with a good worst case performance. Another approach is to design an algorithm with good expected performance. As we saw, in the context of online ad allocation in *Non-Guaranteed Delivery*, by adding the assumption that the queries are coming from an unknown distribution, we can get a better approximation factor. Computer scientists have traditionally adopted the former approach since it does not depend on the probability distribution of the input and worst case analysis is generally much simpler than analysis of the expected performance. Nevertheless, many algorithms with theoretically not so good worst case performance do much better in practice since they have a better performance in expectation.

The offline version of many online algorithms like online resource allocation [14], use linear programming to find an intermediate fractional solution. A similar LP approach usually cannot be used for the online version since solving the LP requires all the information to be known in advance. Still it might be possible to find certain forms of LP's for which an approximation to the optimal solution can be constructed gradually. Consider for example the following class of linear programs:

Definition 1 (Online Packing LP:). *Consider the linear program of the following form:*

$$\textit{maximize:} \quad \mathbf{c}.\mathbf{x} \tag{7.1.1}$$

$$\mathbf{A}\mathbf{x} \leq \mathbf{b} \tag{7.1.2}$$

$$x_i \in [0, 1] \tag{7.1.3}$$

such that the matrix **A** *and the vectors* **b** *and* **c** *are non-negative. We say each variable* x_i*, has the status of being* active *or* inactive*. If inactive then* $x_i = 0$ *otherwise* x_i *could take any value in* $[0, 1]$*. We learn the status (i.e., active/inactive) of variable*

x_i at time t_i and as soon as we find out its status we are required to choose its value (if inactive it should be set to 0*). Once initialized, the value of a variable cannot be changed at a later time. As for distribution information, we know that each variable x_i will be active with probability p_i independent of other variables. The goal is to design an online algorithm that would maximize the expected value of the objective function.*

The online resource allocation problem [14] can be easily modeled as an online packing LP. If we could efficiently compute the expected value of the objective function for the optimal offline solver then we could use the following online algorithm. At each time t_i that the status of variable x_i is revealed, if x_i is active then compute the expected value of the objective for the rest of the LP once for $x_i = 0$ and once for $x_i = 1$ and the choose then one with the highest total expected value. Therefore, it is interesting to see if there is an efficient way to compute the expected value of the objective function. It is worth mentioning that we can efficiently compute an upper bound on the expected value of the objective by adding the constraints $x_i \leq p_i$ and solve the offline linear program. It might also be possible to compute an approximation to the optimal online algorithm if we could compute a constant factor lower bound for the expected value of the objective function and then use that instead of the exact expected value in the online algorithm. This problem will cover a lot of other problems assuming that the inputs are coming from a distribution (e.g., *Online Ad Allocation* in *Guaranteed Delivery*).

Another open problem is related to the online ad allocation problem in *Non-Guaranteed Delivery* from the game theoretic perspective. In the current setting we assumed that we *know* the payments of each ad for each keyword. However in reality, advertisers are bidding on each keyword and the payment is computed based on the bids of the participants in the auction. Assuming that bidders are strategic it is not the case that they will bid their true valuation for each keyword. The interesting question arises when we consider the budget constraint as well. The new problem can be summarized as follows:

- Assuming that the advertisers are strategic and rational and assuming that they have a budget constraint that is publicly known, design a mechanism that will maximize the revenue of the search engine.

We can consider *Fair Allocation Problem* from game theoretic aspect as well. The question can be defined as follows:

- Suppose again that advertisers are rational and strategic. Design a pricing strategy so that assuming that we use the allocation described in chapter 5, truth telling about the demand is dominant strategy for advertisers.

To conclude, I believe that the problems in this thesis address a few of the current challenges in e-Commerce and Internet Advertising. There are many interesting new problems arising in this field and the area is rapidly growing. I believe that this is one of the areas that will receive higher attention for research in the near future due to its potential for generating revenue at large scales for IT companies.

Bibliography

[1] A. Agrawal, P. N. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized Steiner problem on networks. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 134–144, 1991.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, 1993.

[3] S. Alaei, E. Arcaute, S. Khuller, W. Ma, A. Malekian, and J. Tomlin. Online allocation of display advertisements subject to advanced sales contracts. In *KDD Workshop on Data Mining and Audience Intelligence for Advertising*, pages 69–77, 2009.

[4] S. Alaei, R. Kumar, A. Malekian, and E. Vee. Fair allocation with succint representation, 2009.

[5] S. Alaei and A. Malekian. Maximizing sequence-submodular functions and its application to online advertising, 2008.

[6] N. Andelman and Y. Mansour. Auctions with budget constraints. In *9th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 26–38, 2004.

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

[8] Y. Azar, B. Birnbaum, A. R. Karlin, C. Mathieu, and C. T. Nguyen. Improved approximation algorithms for budgeted allocations. In *ICALP '08: Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*, pages 186–197, 2008.

[9] M. Babaioff, J. Hartline, and R. Kleinberg. Selling banner ads: Online algorithms with buyback. In *Proc. 4th Workshop on Ad Auctions*, 2008.

[10] R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *J. Algorithms*, 2(2):198–203, 1981.

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

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

[13] A. L. Buchsbaum, H. J. Karloff, C. Kenyon, N. Reingold, and M. Thorup. Opt versus load in dynamic storage allocation. *SIAM J. Comput.*, 33(3):632–646, 2004.

[14] G. Calinescu, A. Chakrabarti, H. J. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation. In *Proceedings of the 9th International Integer Programming and Combinatorial Optimization Conference, IPCO 2002*, pages 401–414, 2002.

[15] D. Chakrabarty and G. Goel. On the approximability of budgeted allocations and improved lower bounds for submodular welfare maximization and gap. In *FOCS '08: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 687–696, Washington, DC, USA, 2008.

[16] F. Constantin, J. Feldman, S. Muthukrishnan, and M. Pl. An online mechanism for ad slot reservations with cancellations. In *SODA*, 2009.

[17] S. C. Deerwester, S. T. Dumais, T. K. Lancaster, G. W. Furnas, and R. Harshman. Indexing by latent semantic analysis. In *JASIS*, pages 391–407, 1990.

[18] N. R. Devanur and T. P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *EC '09: Proceedings of the tenth ACM conference on Electronic commerce*, pages 71–78, New York, NY, USA, 2009. ACM.

[19] N. R. Devanur, C. H. Papadimitriou, A. Saberi, and V. V. Vazirani. Market equilibrium via a primal-dual algorithm for a convex program. *J. ACM*, 55(5), 2008.

[20] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 97(1):242–259, 2007.

[21] L. Epstein and A. Levin. Improved randomized results for that interval selection problem. In *ESA*, pages 381–392, 2008.

[22] U. Feige, N. Immorlica, V. S. Mirrokni, and H. Nazerzadeh. A combinatorial allocation mechanism with penalties for banner advertising. In *WWW*, pages 169–178, 2008.

[23] U. Feige, V. S. Mirrokni, and J. Vondrak. Maximizing non-monotone submodular functions. In *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 461–471, Washington, DC, USA, 2007. IEEE Computer Society.

[24] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko. Tight approximation algorithms for maximum general assignment problems. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 611–620, New York, NY, USA, 2006. ACM.

[25] R. Garg, V. Pandit, and V. Kumar. Approximation algorithms for budget-constrained auctions, 2001.

[26] A. Ghosh, P. McAfee, K. Papineni, and S. Vassilvitskii. Bidding for representative allocations for display advertising, 2009.

[27] G. Goel and A. Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 982–991, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[28] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.

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

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

[31] S. Iwata, L. Fleischer, and S. Fujishige. A combinatorial, strongly polynomial-time algorithm for minimizing submodular functions. *Journal of the ACM*, 48:761–777, 2001.

[32] S. IWATA, L. FLEISCHER, I. Lisa, and S. FUJISHIGE. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *Journal of the ACM*, 48:761–777, 2000.

[33] K. Jain and V. V. Vazirani. Eisenberg-Gale markets: Algorithms and structural properties. In *Proc. 39th Annual ACM Symposium on Theory of Computing*, pages 364–373, 2007.

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

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

[36] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for online bipartite matching. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, New York, NY, USA, 1990. ACM.

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

[38] S. Khuller, A. Moss, and J. S. Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999.

[39] B. Lehmann, D. Lehmann, and N. Nisan. Combinatorial auctions with decreasing marginal utilities. *Games and Economic Behavior*, 55(2):270–296, May 2006.

[40] A. Malekian, C.-C. Chang, R. Kumar, and G. Wang. Optimizing query rewrites for keyword-based advertising. In *EC '08: Proceedings of the 9th ACM conference on Electronic commerce*, pages 10–19, New York, NY, USA, 2008. ACM.

[41] COIN-OR Foundation: `http://www.coin or.org`.

[42] P. McAfee and K. Papineni. Maximally representative allocation for guaranteed delivery advertising campaigns, 2008. Manuscript.

[43] A. Mehta, A. Saberi, U. Vazirani, and V. Vazirani. Adwords and generalized online matching. *J. ACM*, 54(5):22, 2007.

[44] H. Miyazawa and T. Erlebach. An improved randomized on-line algorithm for a weighted interval selection problem. *J. Scheduling*, 7(4):293–311, 2004.

[45] G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of Operations Research*, 3(3):177–188, 1978.

[46] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. volume 14, pages 265–294, Berlin, Heidelberg, 1978. Springer.

[47] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. In *Math. Prog.*, pages 265–294, 1978.

[48] S. S. Seiden. Randomized online interval scheduling. *Oper. Res. Lett.*, 22(4-5):171–177, 1998.

[49] A. Srinivasan. Budgeted allocations in the full-information setting. In *APPROX-RANDOM*, pages 247–253, 2008.

[50] E. Vee, S. Vassilvitskii, and J. Shanmugasundaram. Online allocation with a compact plan, 2009. yahoo research reoport, YL-2009-05.

[51] G. J. Woeinger. On-line scheduling of jobs with fixed start and end times. *Theoretical Computer Science*, 130(1):5–16, 1994.

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

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