

ABSTRACT

Title of dissertation: ALGORITHMS FOR ONLINE
 ADVERTISING PORTFOLIO
 OPTIMIZATION AND CAPACITATED
 MOBILE FACILITY LOCATION

Mustafa Sahin, Doctor of Philosophy, 2017

Dissertation directed by: Professor S. Raghavan
 The Robert H. Smith School of Business

In this dissertation, we apply large-scale optimization techniques including column generation and heuristic approaches to problems in the domains of online advertising and mobile facility location.

First, we study the online advertising portfolio optimization problem (OAPOP) of an advertiser. In the OAPOP, the advertiser has a set of targeting items of interest (in the order of tens of millions for large enterprises) and a daily budget. The objective is to determine how much to bid on each targeting item to maximize the return on investment. We show the OAPOP can be represented by the Multiple Choice Knapsack Problem (MCKP). We propose an efficient column generation (CG) algorithm for the linear programming relaxation of the problem. The computations demonstrate that our CG algorithm significantly outperforms the state-of-the-art linear time algorithm used to solve the MCKP relaxation for the OAPOP.

Second, we study the problem faced by the advertiser in online advertising in the presence of bid adjustments. In addition to bids, the advertisers are able to

submit bid adjustments for ad query features such as geographical location, time of day, device, and audience. We introduce the Bid Adjustments Problem in Online Advertising (BAPOA) where an advertiser determines base bids and bid adjustments to maximize the return on investment. We develop an efficient algorithm to solve the BAPOA. We perform computational experiments and demonstrate, in the presence of high revenue-per-click variation across features, the revenue benefit of using bid adjustments can exceed 20%.

Third, we study the capacitated mobile facility location problem (CMFLP), which is a generalization of the well-known capacitated facility location problem that has applications in supply chain and humanitarian logistics. We provide two integer programming formulations for the CMFLP. The first is on a layered graph, while the second is a set partitioning formulation. We develop a branch-and-price algorithm on the set partitioning formulation. We find that the branch-and-price procedure is particularly effective, when the ratio of the number of clients to the number of facilities is small and the facility capacities are tight. We also develop a local search heuristic and a rounding heuristic for the CMFLP.

ALGORITHMS FOR ONLINE
ADVERTISING PORTFOLIO
OPTIMIZATION AND CAPACITATED
MOBILE FACILITY LOCATION

by

Mustafa Sahin

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
2017

Advisory Committee:

Professor S. Raghavan, Chair/Advisor

Professor Bruce Golden

Professor MohammadTaghi HajiAghayi (Dean's Representative)

Dr. Abhishek Pani

Professor Courtney Paulson

© Copyright by
Mustafa Sahin
2017

to clare, chubbs, and lily...

Acknowledgments

I don't think I can thank my advisor, Prof. Raghavan, enough for his guidance and support, for challenging me and keeping me focused, and for inspiring me to strive for the better. This degree would not be possible without him. I am hoping to keep learning from him for many years to come.

I would like to thank Prof. Golden for his invaluable feedback on this dissertation, and advice and support for my career. I am thankful for the indispensable contributions of my co-authors, Dr. Halper, Dr. Pani, and Prof. Salman, to my research, and to this dissertation. I wish to thank Prof. HajiAghayi and Prof. Paulson for being in my dissertation committee, and their feedback and insights.

I would like to thank all my friends and family here and back home who, despite the sizable distance and time difference, were there for me whenever I needed. Don, Jordan, and Mary, aka the Marlborough Family. I cannot believe I was fortunate enough to have them as my roommates. I will always cherish the years we spent together. No doctoral degree would be possible in the Smith School without Justina Blanco. However, in this instance, my life wouldn't be possible without her and Talha Aziz, aka my US parents. They have guided and helped me every step of the way, but perhaps most importantly, introduced me to my wife, Clare.

Last but certainly not least, I would like to thank my wife for her unyielding support and understanding. She gave me the fuel and the motivation to complete this dissertation, which I dedicate to her and the girls.

Table of Contents

List of Tables	vi
List of Figures	vii
1 Introduction	1
2 Large-Scale Advertising Portfolio Optimization in Online Marketing	7
2.1 Related Work	14
2.2 Modeling the OAPOP as an MCKP	20
2.2.1 Structural Properties of the Solution to the MCKP-LP	22
2.2.2 Sorting Algorithm for the MCKP-LP	26
2.2.3 Linear Algorithm for the MCKP-LP	30
2.3 Column Generation Algorithm	32
2.3.1 Initial Solution Procedure	33
2.3.2 Column Generation Procedure	34
2.3.2.1 Generating Columns	36
2.3.2.2 Removing Columns	39
2.3.2.3 Overall Implementation and Running Time Analysis	40
2.4 Branch-and-price Algorithm	45
2.5 Computational Results	48
2.5.1 Generating Online Advertising Instances	49
2.5.2 Results for Online Advertising Instances	53
2.5.3 Results for Literature Instances	60
3 Targeted Online Advertising with Bid Adjustments	69
3.1 Related Work	76
3.2 The Bid Adjustment Problem in Online Advertising	78
3.3 Iterative Adjustment Algorithm	82
3.3.1 Base Bid Subproblem	83
3.3.2 Feature Adjustment Subproblem	85
3.3.3 The Multiple Choice Knapsack Problem	90
3.3.4 Summary of the Steps in the Iterative Adjustment Algorithm	93

3.4	Formulating the BAPOA as an MIP	95
3.5	Computational Results	99
3.5.1	Data Generation	101
3.5.2	MIP Instances	108
3.5.3	Online Advertising Instances	113
3.5.4	Creating Advertising Campaigns	121
4	The Capacitated Mobile Facility Location Problem	129
4.1	Related Work	134
4.2	Integer Programming Formulations	137
4.2.1	Layered Graph Formulation	137
4.2.2	Set Partitioning Formulation	142
4.3	Column Generation and Branch-and-Price Procedure	146
4.3.1	Column Generation Procedure for LP2	147
4.3.2	Branching Scheme	149
4.3.2.1	Binary Branching	149
4.3.2.2	Partition Branching	150
4.3.2.3	Hybrid Branching	151
4.3.3	Columns Management	151
4.3.3.1	Setting initial columns	152
4.3.3.2	Adding columns through pricing	155
4.3.3.3	Managing active columns	156
4.4	Heuristics	156
4.4.1	LP Rounding Heuristic	156
4.4.2	Local Search Heuristic	158
4.5	Computational Results	160
4.5.1	Test Instances	160
4.5.2	Computational Settings	162
4.5.3	Homogeneous Facilities Case	163
4.5.3.1	Comparison of the LP Relaxations	164
4.5.3.2	Comparison of the Lower Bounds	166
4.5.3.3	Comparison of the Upper Bounds	167
4.5.4	Heterogeneous Facilities Case	173
5	Concluding Remarks	178

List of Tables

2.1	CPU times (in seconds) for Constant OA instances	54
2.2	CPU times (in seconds) for Random OA instances	55
2.3	CPU times (in seconds) for CPC OA instances	56
2.4	CPU times (in seconds) for UC instances	60
2.5	CPU times (in seconds) for WC instances	62
2.6	CPU times (in seconds) for SC instances	65
3.1	Bids and corresponding number of clicks, cost, and revenue data	74
3.2	Distribution of clicks and cost, and revenue-per-click based on feature combinations.	74
3.3	The Flat Bid revenue as a percentage of the IAA revenue for 1,000, 5,000, 10,000 and 25,000 keyword instances	115
4.1	Comparison of the quality of LP1 and LP2 when $ V / F \leq 10$ and $ T = 1$	170
4.2	Comparison of the quality of lower bounds obtained from IP1, IP1* and IP2 when $ V / F \leq 10$ and $ T = 1$	171
4.3	Comparison of the quality of upper bounds obtained from IP1, IP1*, IP2, LP2RH, and LSH when $ V / F \leq 10$ and $ T = 1$	172
4.4	Comparison of the quality of LP1 and LP2 when $ V / F \leq 10$ and $ T = 2$	175
4.5	Comparison of the quality of lower bounds obtained from IP1, IP1* and IP2 when $ V / F \leq 10$ and $ T = 2$	176
4.6	Comparison of the quality of upper bounds obtained from IP1, IP1*, IP2, and LP2RH when $ V / F \leq 10$ and $ T = 2$	177

List of Figures

2.1	Hierarchical structure of an advertising portfolio	11
2.2	Graphical representation of the upper convex hull of levels	23
2.3	Upper convex hull of keywords	25
2.4	Sorted list of marginal revenue-to-cost ratio	25
2.5	Illustration of the sorting algorithm for the MCKP-LP	25
2.6	Left and right branches for the fractional keyword.	47
2.7	Sample chart of the estimated cost vs the estimated number of clicks	51
2.8	Average CPU time per 1 million keywords vs number of keywords for Constant, Random, and CPC OA instances	57
2.9	Average CPU time per 1 million keywords vs number of levels for Constant, Random, and CPC OA instances	59
2.10	Average CPU time per 1 million keywords vs budget percentage for Constant, Random, and CPC OA instances	61
2.11	Average CPU time per 1 million keywords vs number of keywords for UC, WC, and SC instances	64
2.12	Average CPU time per 1 million keywords vs number of levels for UC, WC, and SC instances	66
2.13	Average CPU time per 1 million keywords vs budget percentage for UC, WC, and SC instances	67
3.1	Hierarchical structure of an advertising portfolio featuring bid adjust- ments	72
3.2	A sample chart of the bid vs the estimated number of clicks	102
3.3	A sample chart of the bid vs the estimated cost	104
3.4	IAA and Flat Bid revenues as percentages of the CPLEX upper bound for 5% budget	109
3.5	IAA and Flat Bid revenues as percentages of the CPLEX upper bound for 10% budget	110
3.6	IAA and Flat Bid revenues as percentages of the CPLEX upper bound for 15% budget	111
3.7	IAA and Flat Bid revenues as percentages of the CPLEX upper bound for 20% budget	112

3.8	IAA and Flat Bid revenues as percentages of the CPLEX upper bound for $\{5\%, 10\%, 15\%, 20\%\}$ budget averaged over $\varepsilon = \{0\%, 10\%, 20\%\}$	113
3.9	The Flat Bid revenue as a percentage of the IAA revenue for 1000 keywords for $\{5\%, 10\%, 15\%, 20\%\}$ budget	116
3.10	The Flat Bid revenue as a percentage of the IAA revenue for 1000 keywords and $\{3, 6, 12, 24\}$ number of times of day.	117
3.11	The running time of the IAA per 1000 keywords vs $ K $	118
3.12	The running time of the IAA per 1000 keywords vs ν	119
3.13	The running time of the IAA per 1M feature combinations vs the number of feature combinations.	120
3.14	The total running time of the subproblems in IAA per 1M feature items vs the number of feature items.	121
3.15	The IAA-Agglo and IAA-Omni revenue as a percentage of the IAA-Individual revenue for $\varepsilon = 0\%$	125
3.16	The IAA-Agglo and IAA-Omni revenue as a percentage of the IAA-Individual revenue for $\varepsilon = 10\%$	126
3.17	The IAA-Agglo and IAA-Omni revenue as a percentage of the IAA-Individual revenue for $\varepsilon = 20\%$	127
4.1	Original and layered graph representations of a solution to an instance of the CMFLP	138
4.2	An example demonstrating LP2 provides a tighter linear programming relaxation bound than that of LP1	146
4.3	An example demonstrating LP1 is feasible for an infeasible problem whereas LP2 is infeasible	146

Chapter 1: Introduction

Business analytics stands atop of three pillars; descriptive, predictive, and prescriptive analytics. They co-exist in a feedback loop as equally important components. On one hand, the ubiquity of and the ability to quickly process data enable firms perform descriptive and predictive analytics at a very high level gaining invaluable insights and looking into the future with unprecedented accuracy and granularity. On the other hand, performing prescriptive analytics in this environment often means dealing with decision problems with massive inputs that need to be solved very quickly. In today's fast paced operational landscape, problems are only getting larger while feedback loops are getting shorter and shorter. Firms now have less time to perform effective prescriptive analytics, which often entails efficiently solving complex and large-scale problems. In this dissertation, we develop solution methods for large-scale decision problems, focusing on both the quality and the efficiency of the methods, in the context of online advertising and logistics.

In Chapter 2, we study the online advertising portfolio optimization problem (OAPOP) faced by an advertiser. Online advertising spending has soared in the US (and worldwide), growing from \$7.26 billion in 2003 to \$72.5 billion in 2016 [PwC Report, 2017]. Advertisers have an overwhelming task in managing online

advertising portfolios with millions of targeting items (e.g., keyword, cookies, website, etc.) across many platforms (e.g., Google, Facebook, etc.) and formats (e.g., search, display, etc.). The OAPOP allows the advertiser to consolidate advertising campaigns across many platforms and formats into a single large-scale portfolio and operate under a single advertising budget. In a typical online advertising auction, the advertising platform collects bids from advertisers for a given targeting item and determines the order of ads that are displayed. The advertiser, on the other hand, has to decide which targeting items to bid on and how much, which platforms and formats to advertise on, all within an advertising budget. Fortunately, advertisers have ample resources to build predictive models for relevant advertising metrics (e.g., number of clicks, number of impressions, cost-per-click, etc.) at a targeting item level for different platforms and formats. In addition to data collected by the advertiser, advertising platforms provide detailed data of past performance and forecast for relevant advertising metrics.

We show when the advertiser has available predictive models that would provide expected cost and revenue data for a given bid amount, the OAPOP can be modeled as a Multiple Choice Knapsack Problem (MCKP). However, given the size of advertising portfolios in industry, the number of targeting items and bid amounts considered in the OAPOP can be in the order of billions. We therefore explore an efficient solution approach that would adequately address the operational requirements (e.g., the problem is solved multiple times throughout the day). We first elaborate on some structural properties of the optimal solution for the linear programming relaxation of the problem. We discuss existing methods, their strengths

and shortcomings, and offer an efficient column generation algorithm. We assess the efficiency of the column generation algorithm in simulated data sets based on a sample collected from “Google Keyword Planner”, with up to 2.5 billion decision variables, which reflects the size of the advertising portfolios currently encountered in industry. Our findings indicate the column generation algorithm significantly outperforms (10 times faster in some cases) the state-of-the-art linear time algorithm under practical settings.

In 2013, Google and Bing have introduced “Enhanced Campaigns” that allow advertisers to more effectively target desired user characteristics. In enhanced campaigns, the advertisers can adjust their bids based on ad query features. Such features include geographical location, time of day, device, audience, etc. Every time an ad query is received, these adjustments are multiplied by a base bid to reach a final bid, which is then submitted to the auction. These adjustments allow advertisers to favor some features over others based on their cost and revenue implications. Bid adjustments are a relatively recent development in online advertising. Even though it is of extreme importance to advertisers, the problem has not been adequately addressed in the literature from a budget optimization and revenue management standpoint.

Enhanced campaigns provide a more sophisticated bidding language, providing an opportunity to build upon our approach in Chapter 2 to take advantage of bid adjustments. The model and solution approach discussed in Chapter 2 provides a high level budget allocation to campaigns operating under enhanced campaigns bidding language (presently, Google and Bing campaigns). However, we can further

increase the advertiser’s return on investment by using bid adjustments for these campaigns. In Chapter 3, we address the need for an efficient and high quality algorithm that can take advantage of bid adjustments and introduce the Bid Adjustments Problem in Online Advertising (BAPOA). We model the BAPOA as a mathematical program. However the way bid adjustments interact in the bidding language leads to computational challenges. We show that the mathematical programming formulation can be decomposed into two subproblems. One to determine bids, one to determine bid adjustments. We develop an efficient algorithm to solve the BAPOA by iteratively creating and solving the two subproblems. We evaluate the quality of the algorithm in comparison to upper bounds obtained from a mixed integer program (the BAPOA can be modeled as a mixed integer program for a discrete approximation) on small instances, and show that the algorithm provides near optimal solutions on these instances. On experiments performed in industry scale data sets (generated based on a sample collected from Google Keyword Planner), we observe the benefit of using bid adjustments increase as the revenue variation increases across features. Our algorithm for the BAPOA operates for a given set of campaigns and ad groups. However, the formation of campaigns and ad groups is a crucial part of the advertiser’s task since different formations will yield different results. We therefore explain how clustering can be used on the adjustments obtained by our algorithm to form campaigns and ad groups to maximize the effectiveness of bid adjustments.

In Chapter 4, we discuss the capacitated mobile facility location problem (CMFLP). In the CMFLP, the objective is to move mobile facilities from their existing

locations to new locations and assign clients in such a way that the total weighted distance traveled by facilities and clients is minimized. In addition to various supply chain applications, the problem finds an application in humanitarian relief logistics where mobile clinics (cancer screening units, blood banks, eye clinics, vaccination booths, etc.) serve people who would otherwise have limited access to healthcare. This application typically results in large-scale problems with many healthcare facilities need to be located to serve large number of patients. We propose a set partitioning formulation and an efficient column generation procedure to solve the linear programming relaxation. We find that the column generation procedure is particularly effective both in terms of solution time and quality, when applied to settings encountered in the mobile healthcare application. We develop a branch-and-price algorithm to solve the CMFLP. Through computational experiments, we demonstrate the quality of the branch-and-price algorithm in comparison to a state-of-the-art commercial solver. In large-scale instances, the branch-and-price algorithm significantly outperforms the state-of-the-art solver, finding better upper and lower bounds for the problem.

The rest of this dissertation is organized as follows. In Chapter 2, we discuss the Online Advertising Portfolio Optimization Problem. In Chapter 3, we introduce the Bid Adjustment Problem in Online Advertising that is suited for platforms and formats operating with bid adjustments. In Chapter 4, we study the capacitated mobile facility location problem. We deliver concluding remarks in Chapter 5. Note that Chapters 2-4 are self contained in terms of motivating their respective problems, discussing related work and contributions, developing models and solution

approaches, and discussing computational experience.

Chapter 2: Large-Scale Advertising Portfolio Optimization in Online Marketing

Online advertising revenues in the United States have grown from \$7.26 billion in 2003 to \$72,5 billion in 2016, the growth from 2015 to 2016 alone was 21.8% [PwC Report, 2017]. Social media and mobile based advertising made great strides in the last few years and are responsible for 22.5% and 50.5% of total revenue in 2015, respectively. From an advertiser's perspective, maximizing return from online advertising efforts became an increasingly difficult endeavor. This has led to the development of software like Adobe Marketing Cloud (<http://www.adobe.com/marketing-cloud.html>), DoubleClick by Google (www.doubleclickbygoogle.com), Kenshoo Infinity (<http://kenshoo.com>), and Marin Software (<http://www.marinsoftware.com/>) that help advertisers manage their online advertising campaigns.

In this chapter, we study the Online Advertising Portfolio Optimization Problem (OAPOP) faced by an advertiser who wishes to maximize the return on online advertising investment subject to an advertising budget. A constant flow of data and shifting consumer patterns require fast solution methods for the OAPOP. The problem needs to be solved and resolved multiple times throughout the day. The portfolios advertisers are interested in can contain tens of thousands of targeting

items (e.g., keywords, cookies, websites, demographic dimensions etc.) for small businesses and millions for large enterprises. In addition, the same targeting items may appear across multiple advertising platforms (e.g., Google, Bing, Facebook etc.) and formats (e.g., search, display, etc.). Therefore, the size of the portfolio considered may easily reach tens of millions since a targeting item under different advertising platforms and formats counts as different targeting items for the purposes of bidding and allocating budget. Further, with new ad delivery channels on the rise, such as Snapchat and programmatic TV ads, the size of these problems will only increase with time

The two main bidding options used in online advertising are; performance based bidding, and impression based bidding. In performance based bidding, also known as cost-per-click (CPC) bidding, the advertiser is only charged for an ad if the ad gets clicked whereas in impression based bidding, also known as cost-per-mille (CPM) bidding, the advertiser is charged for impressions regardless of whether or not the ad gets clicked. In 2016, 64% of online advertising revenue resulted from CPC bidding whereas CPM bidding was responsible for 35% of the revenue [[PwC Report, 2017](#)]. The search advertising format, which roughly made up half of online advertising revenue in 2016, typically operates under CPC bidding. In contrast, the display advertising format operates under both CPC and CPM bidding, and accounted for roughly 45% of online advertising revenue in 2016. In both search and display advertising, the advertisers provide portfolios of targeting items to various advertising platforms along with bidding options (i.e., CPM or CPC), bid amounts, and budgets. Every time a query is received, the platform matches the query to

advertisers' portfolios and determines which ads are displayed as well as how much to charge each advertiser based on bid amounts and bidding options. A query can range from a keyword search in search advertising to a website visit in display advertising. There are billions of queries generated everyday resulting in hundreds of millions of dollars in advertising spending.

Every advertising platform has its own rules governing the payment and the allocation of the ads. However, every major platform (e.g., Google,¹ Bing,² etc.) holds a generalized second price auction for each query. For example, in Google search advertising, the page position of an ad is determined based on the so called 'Ad Rank' of the ad. The Ad Rank is calculated using various features including bid amount, click probability, landing page experience, ad relevance, and ad formats. When a keyword is searched, Google calculates the Ad Rank of each advertiser who placed a bid on the keyword. Then the advertiser with the highest Ad Rank gets the top position on the page and if clicked, the advertiser pays the minimum bid amount that would maintain the position of the advertiser. The subsequent page positions are awarded in decreasing order of Ad Rank and the advertisers are charged for clicks in a similar fashion. In display advertising, Google's auction mechanism is similar to that of search advertising save for some minor differences. For instance, since both CPC and CPM bidding options are allowed, different options may compete in the same auction. Google converts CPC bids into an equivalent CPM bid by estimating the expected clicks an ad would get for a thousand impressions. More information

¹Explained by Google's Chief Economist Hal Varian, <https://youtu.be/5ZnWq0XMClc>

²<https://advertise.bingads.microsoft.com/en-us/blog/post/september-2013/bing-ads-auction-explained-how-bid,-cost-per-click-and-quality-score-work-together>

on Google's auction mechanisms for search³ and display⁴ advertising formats are available online.

There are two ways online advertising impacts advertiser revenue. First, clicks that directly result in sales. Second, clicks and impressions that do not directly result in sales but build brand awareness, which over time, results in sales. For each query of a targeting item, the bid amount determines the page position and the respective click outcome resulting in a payment based on the bidding option. Over a time horizon (e.g., an hour, a day, etc.), a query for a targeting item may be received many times. The average number of clicks, number of impressions, and amount paid can all be viewed as functions of the bid amount. Therefore, a bid amount can be linked to an expected cost over a given time horizon. All the data required to estimate the expected cost are provided by advertising platforms in detailed historical performance reports and forecasts. Moreover, by using methods such as Pixel Tracking and URL redirects the advertiser can track direct sales outcomes of each click, and indirect sales outcomes of impressions and clicks. *In essence, regardless of bidding option, advertising platform, or advertising format, a bid amount can be linked to estimates of expected cost and expected revenue for every targeting item.* Using these estimates, an advertiser can consolidate all online advertising campaigns under one portfolio and operate under a common budget over a specified time horizon.

An advertiser has multiple ad campaigns each with a budget (e.g., daily) that needs to be specified to the advertising platform. Each campaign has a set

³Adwords Auctions: <https://support.google.com/adwords/answer/142918?hl=en>

⁴Display Network Auctions: <https://support.google.com/adwords/answer/2996564?hl=en>

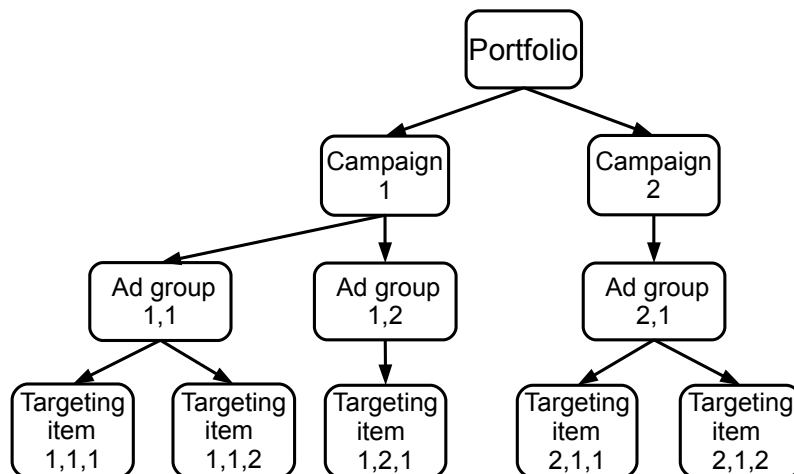


Figure 2.1: The hierarchical structure of a portfolio with 2 campaigns, 3 ad groups, and 5 targeting items. In this portfolio, Campaign 1 represents a search format campaign in Google Adwords, whereas Campaign 2 represents a display format campaign in Bing Ads. The OAPOP is solved for a specified portfolio budget, however, ad platforms require the advertisers to submit a separate budget for each campaign. After determining optimal bids for every targeting item in the portfolio by solving the OAPOP, the advertiser can aggregate the expected cost for each targeting item in a given campaign and submit it as the campaign budget.

of ad groups which in turn have a set of targeting items. The advertiser needs to determine a bid for each targeting item in an ad group. Figure 2.1 schematically represents the structure of the advertiser’s problem. Note that the total expected cost of a campaign based on the determined bids would be set as the budget of that campaign, ensuring the sum of campaign budgets across all the campaigns add up to the total advertising budget. This allows the campaigns to be combined into a single advertising portfolio optimization problem, which we refer to as the OAPOP. In the OAPOP, given campaigns across various bidding options, advertising platforms, and advertising formats, the advertiser needs to decide how much to bid on each targeting item to maximize the total expected revenue while ensuring the total expected cost

does not exceed the specified budget. This problem can be modeled as a multiple choice knapsack problem (MCKP). In the MCKP, the objective is to select at most one item (bid amount) from each class (targeting item) such that the sum of the weights (expected cost) of selected items does not exceed the capacity (budget) while the total reward (expected revenue) is maximized.

Much of the previous research (discussed in Section 2.1) has focused on other aspects of the problem (e.g., game-theoretic results or simplified/smaller versions of the problem faced by advertisers in practice). As the scale and volume of online advertising has increased so has the need for an efficient solution to the OAPOP. Our research is motivated by the daily operational problem faced by an advertiser with a large advertising portfolio. In this setting it is necessary to solve huge OAPOPs rapidly (e.g., solving the OAPOP in seconds multiple times a day for portfolios with tens of millions of targeting items). In addition, advertisers need to assess the revenue-cost trade-off of using different budget values. Therefore, it is important to develop a solution method where this trade-off can be assessed efficiently. In settings such as Real Time Bidding Display and Facebook FBX, the advertiser is provided information on a cookie in real-time and has to submit a bid in less than 20 milliseconds. To address these strict latency requirements, typically an offline problem is solved (where a set of cookies is treated as a targeting item) to determine the average optimal bid and adjustments to this offline bid are made in real time based on the ad request features. The OAPOP solution can be used to determine the offline optimal bid for the target.

Our Contributions: In this chapter, we model the advertiser’s Online Adver-

tising Portfolio Optimization Problem as a MCKP (an integer program) by linking bid amounts for targeting items to expected cost and revenue. Due to the structural properties of the linear programming relaxation of the MCKP (MCKP-LP) its solution has at most one targeting item that is fractional. This solution can easily be rounded to obtain a feasible solution to MCKP; which we find for all practical purposes can be considered as “optimal” (since it is well-within the optimality tolerance limit of most commercial optimization solvers) for the large-scale instances considered in this chapter. We propose a scalable column generation algorithm to rapidly solve enterprise size MCKP-LP instances. Through computational experiments conducted on problem sets based on instances in online advertising and literature, we demonstrate the column generation algorithm significantly outperforms the state-of-the-art linear time algorithm proposed by [Dyer \[1984\]](#) and [Zemel \[1984\]](#). In addition, the column generation algorithm can provide solutions for different budget values (for revenue-cost trade-off analysis) in a single run of the algorithm, a feature [Dyer’s](#) and [Zemel’s](#) algorithm does not possess. We show the column generation algorithm solves problems with 2.55 billion variables (corresponding to 50 million targeting items) in the order of a few minutes on a personal computer.

The rest of the chapter is organized as follows. Section [2.1](#) provides a literature review. In Section [2.2](#), we describe the advertiser’s problem and elaborate on the structural properties of the MCKP along with two solution approaches for the MCKP-LP. We provide the sorting algorithm for the MCKP-LP originally proposed by [Sinha and Zoltners \[1979\]](#), which we use as a building block for the column generation (CG) algorithm described in Section [2.3](#). We also provide a high level overview

of the [Dyer](#)'s and [Zemel](#)'s (DZ) algorithm. In Section [2.4](#), we explain how to embed the CG algorithm within a branch-and-bound framework to solve MCKP (although this was never necessary for OAPOP instances we discuss the branch-and-bound framework for completeness). In Section [2.5](#), we discuss our computational experience on large-scale simulated online advertising instances with up to 50 million targeting items and 2.55 billion bid levels. Although we perform online advertising computations in the context of Google Adwords, the computational setting and results are valid for other online advertising platforms (e.g., Bing Ads) and formats (e.g., display format, social ads).

2.1 Related Work

The rapid online advertising revenue growth in the last fifteen years resulted in increased research activity in the Operations Research, Computer Science, Information Systems, and Marketing communities. There is a significant body of research on mechanism design problems faced by advertising platforms. To list a few, [Feng et al. \[2007\]](#) model and compare various ad allocation mechanisms. The authors report that allocating purely based on bid is as efficient as allocating based on bid and relevance when bids and relevance are positively correlated. [Varian \[2007\]](#) explicitly calculates the Nash equilibria of the Google Adwords auctions and shows that the generalized second price auction is not incentive compatible and is not equivalent to the Vickrey-Clarke-Groves (VCG) mechanism. [Edelman et al. \[2005\]](#) and [Aggarwal et al. \[2006\]](#) independently prove that truth telling is not a dominant strategy for

generalized second price auctions and [Aggarwal et al. \[2006\]](#) develop a mechanism called the ladder auction where bidding true valuations is an optimal strategy.

The problems faced by the advertisers started to gain some attention in recent years. [Rusmevichientong and Williamson \[2006\]](#) formulate the advertisers' keyword selection problem in three settings. First, they describe a static setting where the click-through rates and expected profits are fixed and known. In the second setting, the expected profits are known but the click through rates are unknown while in the third setting, both the click through rates and the expected profits are unknown. The static setting is modeled as a variation of the stochastic knapsack problem and used as a baseline for an adaptive algorithm that solves the other settings. The authors do not model multiple bid levels and therefore are unable to capture the effect of different bid levels on cost and revenue. [Ghose and Yang \[2009\]](#) use a six-month panel data to model the relationship between the click through rates, conversion rates, cost-per-click, and the ad position. The authors show the revenue generated from a click is not uniform across all positions. Though not addressed in the paper, the study makes a compelling case for using optimization methods that take into account different bid levels resulting in different expected costs and revenues. [Feldman et al. \[2007\]](#) consider a problem where the advertiser has a set of keywords and search queries for each keyword. Then the authors try to maximize the total number of clicks such that the total cost does not exceed the capacity. The authors offer two approximation algorithms, the strategy in the first algorithm is a two-bid uniform strategy where the advertiser randomizes between two bid amounts for all keywords, which results in a $(1 - 1/e)$ approximation guarantee. The strategy

in the second algorithm is a uniform bid strategy that uses one bid amount for all keywords, which results in a $(1/2)$ approximation guarantee. Note, however, that in practice advertisers are highly unlikely to use uniform bidding strategies (i.e., the same bid amount for all keywords).

[Borgs et al. \[2007\]](#) consider the bidding problem among multiple bidders with limited budgets for multiple keywords with multiple ad slots. The authors introduce a bidding heuristic based on marginal return-on-investment across all keywords. When all bidders use the same heuristic, it triggers a cycling behavior. The authors show that the cycling can be eliminated by introducing random perturbations to bids. [Muthukrishnan et al. \[2007\]](#) study the stochastic budget optimization problem for multiple keywords and single slot auctions where the budget and the set of keywords are known whereas the number of searches and clicks for each keyword are assumed to be probabilistic. The authors consider three types of randomness, the number of searches and clicks vary but proportions of clicks for different keywords stay the same, each keyword has its own probability distribution for searches and clicks, and the number of searches and clicks come from scenarios each of which specifies the exact number of searches and clicks for every keyword. The authors provide structural and approximation insights for each type of randomness. [Abhishek and Hosanagar \[2013\]](#) consider the advertiser's bid determination problem with multiple keywords in multi-slot auctions. The authors propose a stochastic model where expected value (e.g., revenue) is maximized subject to a budget. Under mild technical conditions and the assumption that the advertiser has detailed information on the number of competing bids for each keyword and their distributions, the authors are

able to derive a closed form expression for the optimal bid. This approach works for small problems (like the 247 keyword problem considered in their paper) but does not scale up to large problems for two reasons. First, they dualize the budget constraint and solve the problem using Lagrangian relaxation. This requires repetitively solving the problem using a subgradient approach (i.e. changing the Lagrange multiplier and resolving the problem) which can be notoriously slow to converge. Second, the burden of collecting information on competitors' bids on each keyword is quite significant, and in fact not practically feasible due to the opaque nature of all ad platforms.

As discussed earlier, we model the OAPOP as a multiple choice knapsack problem. The MCKP is a well-studied variant of the knapsack problem where the choice of selecting an item is replaced by the requirement to select exactly one item out of each class of items. [Lin \[1998\]](#) provides a survey and [Kellerer et al. \[2004\]](#) discusses solution methods for the MCKP. [Sinha and Zoltners \[1979\]](#) highlight and prove structural properties of the linear programming relaxation of the MCKP (MCKP-LP). They use these properties to develop a sorting algorithm for the MCKP-LP. They embed this approach into a branch-and-bound algorithm for the MCKP. [Dyer \[1984\]](#) and [Zemel \[1984\]](#) independently provide a linear time algorithm for the dual of the MCKP-LP. [Pisinger \[1995\]](#) provides a primal version of the algorithm with a few improvements and an expanding core approach to solve the integer version of the MCKP. Unfortunately, the sorting algorithm proposed by [Sinha and Zoltners \[1979\]](#) is not viable for large-scale problems as it requires the entire data set containing tens of millions of targeting items and billions of bid

levels to be sorted. Although [Dyer's](#) and [Zemel's](#) linear time algorithm represents the state-of-the-art in terms of solving the MCKP-LP, there are significant benefits (where rapid solutions are of paramount importance) in online advertising if one is able to solve the MCKP-LP and thus, the OAPOP faster. Our column generation algorithm for the MCKP-LP (applied to the OAPOP) addresses this shortcoming.

Some versions of the advertiser's problem have been modeled as variants of the multiple choice knapsack problem previously in the literature. [Pani \[2010\]](#) models the OAPOP as an MCKP in the context of sponsored search and employs a column generation algorithm for the LP relaxation of the MCKP. In addition, the author provides a proof of correctness for the sorting algorithm introduced by [Sinha and Zoltners \[1979\]](#) using the complementary slackness property of linear programming. We also provide this proof for completeness. We use this work as a building block for our column generation algorithm. [Zhou et al. \[2008\]](#) consider the problem of bidding on a single keyword and model it as an online multiple choice knapsack problem (O-MCKP). The classes in the O-MCKP correspond to time intervals (e.g., 24 hourly time intervals in a day) and items in a class correspond to ad slots. The advertiser needs to determine a bid amount for the keyword and is only allowed to change it between time intervals. In the beginning of each time interval before determining the bid amount, the authors assume the advertiser knows the cost and revenue of each ad slot for that time interval. While cost and revenue for the succeeding time intervals are not known by the advertiser, cost-to-revenue ratio of each ad slot is assumed to be in the range $[L, U]$. The authors offer an online bidding strategy for each time interval with a competitive ratio (against an omniscient bidder who

knows cost and revenue of every ad slot of every time interval a priori) of $\ln(U/L)+2$. Note that the algorithm is of little practical importance due to its large competitive ratio. More importantly, the model setting fails to account for the budget allocation trade-off between keywords present in a multiple keyword portfolio. [Zhou and Naroditskiy \[2008\]](#) go one step further and consider multiple keywords in the online problem. In each time interval, keywords are considered sequentially in an online fashion. They assume the revenue-to-cost distribution for all keywords in subsequent time intervals is independently and identically distributed. They call this the stochastic multiple choice knapsack problem (S-MCKP). The authors propose a heuristic algorithm for the S-MCKP, using Lueker’s algorithm ([\[Lueker, 1998\]](#)) for the online knapsack problem. Again, considering the problem in an online fashion does not allow for trade-offs between keywords to be considered as effectively as in the OAPOP. Further, the assumption of an independent and identical distribution of revenue-to-cost for all time intervals and all keywords is not supported by data in practice. [Berg et al. \[2010\]](#) consider the advertiser’s problem with a soft budget constraint where overages are penalized. They model it as a penalized multiple choice knapsack problem (P-MCKP). Under a very restrictive assumption that the overall penalty function (which is a function of the amount the budget is exceeded by) can be separated out by keyword and that this separable penalty function is convex for each keyword, it is very easy to show that the original sorting algorithm for the MCKP can be applied to the problem. However, this restrictive assumption has no connection to problems faced in practice.

We note that when the budget has flexibility, the typical approach in practice

is to solve the OAPOP with different budget values and assess the trade-off between the increased revenue and any penalties. As we will see in Sections 2.2 and 2.3 this can actually be done in one run of the [Sinha and Zoltners \[1979\]](#) sorting algorithm as well as our column generation algorithm, but not in the Dyer’s and Zemel’s algorithm.

2.2 Modeling the OAPOP as an MCKP

In the OAPOP, the advertiser has under consideration a set of targeting items and an advertising budget. For each targeting item, the advertiser needs to determine a bid amount from a set of possible bids (i.e., set of bid levels) that will be submitted to the ad platform for a specified time horizon. Since the largest fraction of targeting items are “keywords”, for brevity, we use “keyword” instead of “targeting item” henceforth. However, it is important to note that the model, the solution approach, and the computational insights directly apply to all types of targeting items (e.g., cookies, websites, demographic dimensions, etc.) concurrently present in the portfolio. We choose a time horizon of ‘one day’ as is common in practice; although none of our methods or conclusions are affected by a different time horizon (in fact, if different bid amounts are desired through the 24 hours of a day, a keyword may be represented as 24 different keywords, one for each hour of the day). We assume the advertiser has available daily expected cost and revenue estimates for each bid level. In addition to building predictive models based on historical data, there are many sophisticated tracking tools available that help advertisers de-

termine these estimates, e.g., Google Adwords Keyword Planner. The objective of the advertiser is to maximize the daily expected revenue across all keywords and all possible bid levels given the daily budget. Let K be the set of keywords and $P_i = \{0, 1, 2, \dots\}$ the set of bid levels for keyword i . The advertiser has a total budget of B . Every keyword $i \in K$ and level $j \in P_i$ has a nonnegative expected cost-revenue pair (c_{ij}, r_{ij}) where $j = 0$ such that $(c_{i0}, r_{i0}) = (0, 0)$ is designated as the zero bid level. If the selected bid level for keyword i is j then the expected daily cost is c_{ij} and the expected daily revenue is r_{ij} . Let x_{ij} be a binary variable taking the value 1 if the advertiser places a bid on keyword i at level j , and 0 otherwise. Then we have the following integer program (IP).

$$\begin{aligned} & \text{Maximize } \sum_{i \in K} \sum_{j \in P_i} r_{ij} x_{ij} \\ & \text{Subject to } \sum_{j \in P_i} x_{ij} = 1 \qquad \qquad \qquad i \in K \end{aligned} \tag{2.1}$$

$$\sum_{i \in K} \sum_{j \in P_i} c_{ij} x_{ij} \leq B \tag{2.2}$$

$$x_{ij} \in \{0, 1\} \qquad \qquad \qquad i \in K, j \in P_i$$

The objective function maximizes the total expected revenue across all keywords and bid levels. Constraint set (2.1) states exactly one bid level is chosen for each keyword. Constraint (2.2) ensures the total budget is not exceeded. The advertisers problem as formulated above is a Multiple Choice Knapsack Problem (MCKP), which is NP-Hard. The advertiser typically has the option to *pause* a keyword in the portfolio, which corresponds to not placing a bid for the keyword. In some cases,

the advertiser may wish to keep the keyword active on the portfolio at the *minimum allowable bid* level. The amount of the minimum allowable bid varies based on the advertising platform. For instance, the minimum allowable bid is \$0.01 in Google Adwords. In the case of minimum allowable bids, the zero bid level can be adjusted to correspond to the minimum allowable bid level by setting $B = B - \sum_{i \in K} c_{i0}$, $R' = \sum_{i \in K} r_{i0}$, $c_{ij} = c_{ij} - c_{i0}$, and $r_{ij} = r_{ij} - r_{i0}$ for all $j \in P_i$. After the problem is solved, R' would be added to the objective value to calculate the actual revenue.

2.2.1 Structural Properties of the Solution to the MCKP-LP

We now discuss some well known properties of the solution to the MCKP-LP which we will use in our column generation algorithm. For ease of exposition, we will present these results in the context of the OAPOP.

Proposition 2.2.1 LP Dominance. *If some levels j , j' and j'' for keyword i such that $c_{ij''} > c_{ij'} > c_{ij}$ and $r_{ij''} > r_{ij'} > r_{ij}$ satisfy the following*

$$\frac{r_{ij''} - r_{ij'}}{c_{ij''} - c_{ij'}} \geq \frac{r_{ij'} - r_{ij}}{c_{ij'} - c_{ij}}$$

then level j' is said to be LP-dominated by levels j and j'' . There exists an optimal solution to the MCKP-LP where $x_{ij'} = 0$.

Proof See [Sinha and Zoltners \[1979\]](#).

The proposition essentially states that a convex combination of levels j and j'' outperforms the selection of level j' for keyword i . Using this LP Dominance

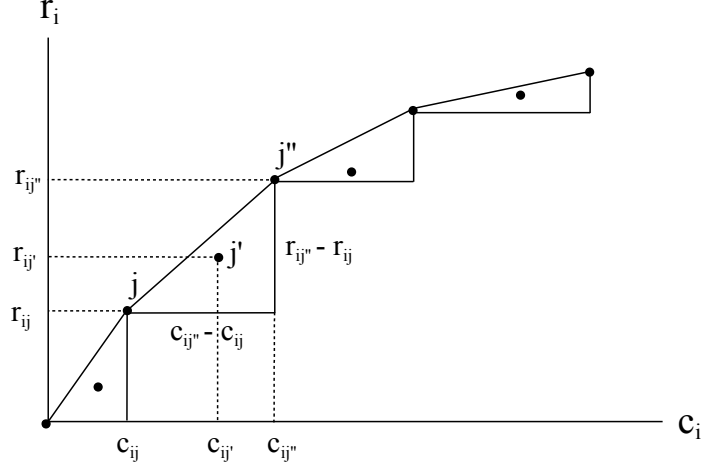


Figure 2.2: The upper convex hull of levels for keyword i .

property, it immediately follows that the LP-Dominating levels for keyword i form the upper convex hull of the revenue-to-cost function as shown in Figure 2.2. In other words, the only levels with nonzero values in the solution to the MCKP-LP lie on the upper convex hull of the revenue-to-cost function for keyword i . Let Ω_i denote the set of levels that form the upper convex hull of keyword i . Without loss of generality, assume levels in Ω_i are in nondecreasing order of c_{ij} . Throughout the chapter, we use (\cdot) to map the order of a level in Ω_i to a level in P_i . For example, if $P_i = \{0, 1, 2, 3, 4, 5, 6\}$ and $\Omega_i = \{0, 2, 6, 4\}$, then $(0) = 0$, $(1) = 2$, $(2) = 6$, and $(3) = 4$. For any keyword $i \in K$, we have $(0) = \arg \min_{j \in P_i} \{c_{ij}\}$ and $(|\Omega_i| - 1) = \arg \max_{j \in P_i} \{r_{ij}\}$ for Ω_i . Let $\bar{r}_{i(j)} = r_{i(j)} - r_{i(j-1)}$ be the *marginal revenue*, $\bar{c}_{i(j)} = c_{i(j)} - c_{i(j-1)}$ be the *marginal cost* and $\delta_{i(j)} = \bar{r}_{i(j)} / \bar{c}_{i(j)}$ be the *marginal efficiency* of keyword i and bid level (j) . We now review a well known sorting algorithm [due to [Sinha and Zoltners, 1979](#)] for the MCKP-LP. We prove the correctness of the algorithm [differently than [Sinha and Zoltners, 1979](#)] in a

manner that will be useful in the development of the column generation algorithm for the OAPOP. First, we describe a couple of useful properties that follow from LP-Dominance.

Proposition 2.2.2 *For Ω_i , $(0) = 0$, $(1) = \arg \max_{j \in P_i \setminus \{0\}} \{r_{ij}/c_{ij}\}$, and $\delta_{i(1)} = r_{i(1)}/c_{i(1)}$.*

Proposition 2.2.2 states that of all the levels for keyword i , the second level on the upper convex hull (i.e., level (1)) has the highest revenue/cost ratio.

Proposition 2.2.3 *$\delta_{i(j'+1)}(c_{i(j)} - c_{i(j')}) \geq r_{i(j)} - r_{i(j')}$ for all $j \in \{0, \dots, |\Omega_i| - 1\}$, $j' \in \{0, \dots, |\Omega_i| - 2\}$.*

Proposition 2.2.3 states that given a level (j') on the upper convex hull of keyword i , the incremental revenue/cost ratio for any pair of levels (j), (j') such that $j > j'$ is less than or equal to the marginal efficiency of level ($j' + 1$). Conversely, the incremental revenue/cost ratio for any pair of levels (j), (j') such that $j < j'$ is larger than or equal to the marginal efficiency of level ($j' + 1$).

Let $\Omega = \{(i, j) \mid i \in K, j \in \Omega_i\}$ and let MCKP-LP(Ω) denote the LP relaxation of MCKP where all LP-dominated levels are removed from the problem. The MCKP-LP(Ω) can be formulated as,

$$\begin{aligned}
 & \text{Maximize} && \sum_{i \in K} \sum_{j \in \Omega_i} r_{ij} x_{ij} \\
 & \text{Subject to} && \sum_{j \in \Omega_i} x_{ij} = 1 && i \in K \quad (2.3)
 \end{aligned}$$

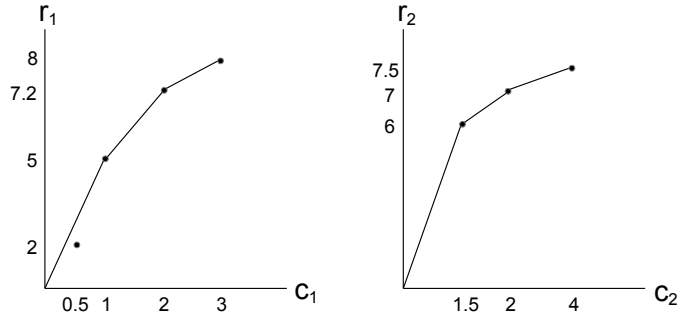


Figure 2.3: Upper convex hull of keywords 1 and 2

$\bar{\Delta}$	$\bar{r}_{i(j)}$	$\bar{c}_{i(j)}$	$\delta_{i(j)}$
(1,(1))	5	1	5
(2,(1))	6	1.5	4
(1,(2))	2.2	1	2.2
(2,(2))	1	0.5	2
(1,(3))	0.8	1	0.8
(2,(3))	0.5	2	0.25

Figure 2.4: Sorted $\bar{\Delta}$

Figure 2.5: Illustration of the sorting algorithm for the MCKP-LP

$$\sum_{i \in K} \sum_{j \in \Omega_i} c_{ij} x_{ij} \leq B \tag{2.4}$$

$$x_{ij} \geq 0 \quad i \in K, j \in \Omega_i$$

By Proposition 2.2.1, solving the MCKP-LP(Ω) is equivalent to solving the MCKP-LP.

The structural properties of the MCKP-LP allow for the development of efficient algorithms. We review two combinatorial algorithms for the MCKP-LP that takes advantage of these structural properties.

2.2.2 Sorting Algorithm for the MCKP-LP

To easily follow the sorting algorithm consider the problem with a single keyword. Then for a given budget B , the optimal revenue is given by the point on the convex hull corresponding to a cost of B . This corresponds to a solution that either selects a single level or two levels - one with a cost larger than B and one with a cost smaller than B . The sorting algorithm concatenates all the levels on the convex hull of every keyword into a single sorted list $\bar{\Delta}$ (sorting is based on marginal efficiency δ_{ij}). It selects the levels in the sorted list $\bar{\Delta}$ in order until the budget is exhausted. Let $n = \sum_{i \in K} |P_i|$, then the running time of the sorting algorithm is $O(\sum_{i \in K} |P_i| \log |P_i| + n \log n)$ where the first term is for deriving the convex hull (assuming c_{ij} can be in arbitrary order) of every keyword $i \in K$ and the second term is for sorting $\bar{\Delta}$.

Algorithm 1 describes the procedure which we illustrate with a small example in Figure 2.5. In the example, the keyword set is $K = \{1, 2\}$, where $(c_1, r_1) = \{(0, 0), (0.5, 2), (1, 5), (2, 7.2), (3, 8)\}$, $(c_2, r_2) = \{(0, 0), (1.5, 6), (2, 7), (4, 7.5)\}$, and $B = 3$. We derive the upper convex hull for each keyword as depicted in Figure 2.3, i.e., $\Omega_1 = \{0, 2, 3, 4\}$, $\Omega_2 = \{0, 1, 2, 3\}$. Initially, we set $C = B$, $R = 0$, $x_{1(0)} = 1$ and $x_{2(0)} = 1$. Then we calculate the marginal efficiency for each level in the convex hull of each keyword, e.g., $\delta_{1(2)} = \frac{7.2-5}{2-1} = 2.2$. We populate and sort $\bar{\Delta}$ in nonincreasing order of $\delta_{i(j)}$ as shown in Figure 2.4. As the first level, we select the level with the largest marginal efficiency in $\bar{\Delta}$, i.e., $(1, (1))$ and set $R = R + \bar{r}_{1(1)} = 0 + 5 = 5$, $C = C - \bar{c}_{1(1)} = 3 - 1 = 2$, $x_{1(0)} = 0$, $x_{1(1)} = 1$, $\bar{\Delta} = \bar{\Delta} \setminus \{(1, (1))\}$, and $\Delta =$

Algorithm 1 Sorting Algorithm for MCKP-LP

```

1: derive  $\Omega_i$  for all  $i \in K$ 
2: set  $R = 0$ ,  $C = B$ , set  $x_{i(0)} = 1$  for all keywords  $i \in K$ .
3: set  $\bar{\Delta} = \{(i, (j)) \mid i \in K, j = 1, \dots, |\Omega_i| - 1\}$  and sort  $\bar{\Delta}$  in nonincreasing order
   of  $\delta_{i(j)}$ 
4: set  $\Delta = \{(i, (0)) \mid i \in K\}$ 
5: while  $C > 0$  do
6:   select  $(i, (j)) = \arg \max_{(i, (j)) \in \bar{\Delta}} \delta_{i(j)}$ 
7:    $\Delta := \Delta \cup \{(i, (j))\}$ ,  $\bar{\Delta} := \bar{\Delta} \setminus \{(i, (j))\}$ 
8:   if  $C - \bar{c}_{i(j)} < 0$  then
9:     set  $x_{i(j)} := C/\bar{c}_{i(j)}$ ,  $x_{i(j-1)} := 1 - x_{i(j)}$ ,  $R := R + \bar{r}_{i(j)}x_{i(j)}$ ,  $C := 0$ 
10:  else
11:    set  $x_{i(j)} := 1$ ,  $x_{i(j-1)} := 0$ ,  $R := R + \bar{r}_{i(j)}$ ,  $C := C - \bar{c}_{i(j)}$ 
12:  end if
13: end while

```

$\Delta \cup \{(1, (1))\}$. Second, we select $(2, (1))$ and set $R = R + \bar{r}_{2(1)} = 5 + 6 = 11$, $C = C - \bar{c}_{2(1)} = 2 - 1.5 = 0.5$, $x_{2(0)} = 0$, $x_{2(1)} = 1$, $\bar{\Delta} = \bar{\Delta} \setminus \{(2, (1))\}$, and $\Delta = \Delta \cup \{(2, (1))\}$. Third, we select $(1, (2))$, however since $C - \bar{c}_{1(2)} < 0$, we set $x_{1(2)} = C/\bar{c}_{1(2)} = 0.5$, $x_{1(1)} = 1 - x_{1(2)} = 0.5$, $R = R + \bar{r}_{1(2)}x_{1(2)} = 11 + 2.2 \times 0.5 = 12.1$, $C = 0$, $\bar{\Delta} = \bar{\Delta} \setminus \{(1, (2))\}$, and $\Delta = \Delta \cup \{(1, (2))\}$. We terminate the algorithm since $C = 0$ with $R = 12.1$, $x_{1(1)} = 0.5$, $x_{1(2)} = 0.5$, and $x_{2(1)} = 1$.

Theorem 2.2.4 *The MCKP-LP can be solved by the sorting algorithm outlined in Algorithm 1.*

Proof Consider the dual of the MCKP-LP(Ω).

$$\text{Minimize } B\theta + \sum_{i \in K} \lambda_i \quad (2.5)$$

$$\text{Subject to } c_{ij}\theta + \lambda_i \geq r_{ij} \quad i \in K, j \in \Omega_i \quad (2.6)$$

$$\theta \geq 0 \quad (2.7)$$

At the termination of the sorting algorithm, define $A = \{i \mid i \in K, x_{i(0)} \neq 1\}$ and $\bar{A} = K \setminus A$ and construct a dual solution by setting $\theta = \delta_{s(\ell+1)}$ where s denotes the last keyword and $(\ell + 1)$ denotes the last level selected by the sorting algorithm, $\lambda_s = r_{s(\ell+1)} - \theta c_{s(\ell+1)}$, $\lambda_i = 0$ for $i \in \bar{A}$, and $\lambda_i = r_{ij'} - \theta c_{ij'}$ for $i \in A \setminus \{s\}$ where $j' = \{j \mid j \in \Omega_i, x_{ij} = 1\}$. We show that the solution obtained from the sorting algorithm and the constructed dual solution satisfy primal feasibility, dual feasibility, and the complementary slackness conditions.

1. **Primal Feasibility.** By construction.

2. **Dual Feasibility.** Constraint (2.7) is satisfied since $\theta = \delta_{s(\ell+1)} \geq 0$. Consider constraint set (2.6), for a keyword $i \in \bar{A}$, $\lambda_i = 0$ and the constraint trivially holds for $j = (0)$. For $j = \Omega_i \setminus \{(0)\}$, the constraint set becomes $\delta_{s(\ell+1)} \geq \frac{r_{ij}}{c_{ij}}$, which holds since the sorting algorithm selects level $(\ell + 1)$ for keyword s and does not select any level from keyword i . Therefore, $\delta_{s(\ell+1)} \geq \delta_{i(1)}$ holds for all $i \in \bar{A}$. Combining with Proposition 2.2.2, we have $\delta_{s(\ell+1)} \geq \delta_{i(1)} = \max_{k \in \Omega_i} \frac{r_{ik}}{c_{ik}} \geq \frac{r_{ij}}{c_{ij}}$ for every $j \in \Omega_i \setminus \{(0)\}$.

For a keyword $i \in A$, the constraint set becomes

$$c_{ij}\delta_{s(\ell+1)} + r_{ij'} - c_{ij'}\delta_{s(\ell+1)} \geq r_{ij} \quad j \in \Omega_i,$$

or,

$$\delta_{s(\ell+1)}(c_{ij} - c_{ij'}) \geq r_{ij} - r_{ij'} \quad j \in \Omega_i.$$

Let $(k) = j'$, then (k) is the last level selected for keyword i by the sorting algorithm, i.e., $(k + 1), \dots, (|\Omega_i| - 1)$ are not selected. Therefore, $\delta_{s(\ell+1)} \geq$

$\delta_{i(k+1)}$. Combining with Proposition 2.2.3, we get

$$\delta_{s(\ell+1)}(c_{ij} - c_{ij'}) \geq \delta_{i(k+1)}(c_{ij} - c_{ij'}) \geq r_{ij} - r_{ij'} \quad j \in \Omega_i,$$

or,
$$\delta_{s(\ell+1)}(c_{ij} - c_{ij'}) \geq r_{ij} - r_{ij'} \quad j \in \Omega_i.$$

3. **Complementary Slackness.** For constraint (2.4), $\theta \left(B - \sum_{i \in K} \sum_{j \in \Omega_i} c_{ij} x_{ij} \right) = 0$ is satisfied since the budget B is exhausted in the sorting algorithm. For constraint set (2.6), $x_{ij}(c_{ij}\theta + \lambda_i - r_{ij}) = 0$ for $i \in K, j \in \Omega_i$ is satisfied for $i \in \bar{A}$ since $x_{ij} = 0$. For $i \in A$ and $j \neq j' \in \Omega_i$, $x_{ij} = 0$, and for j' , $\lambda_i = r_{ij'} - c_{ij'}\theta$ by construction. ■

The following well known property of the solution to the MCKP-LP immediately follows Theorem 2.2.4.

Proposition 2.2.5 LP Fractional variables. *There exists an optimal solution $\mathbf{x}^* = \{x_{ij} \mid i \in K, j \in P_i\}$ for the MCKP-LP that has either zero or two fractional variables. When \mathbf{x}^* has no fractional variables it is optimal for the MCKP. When \mathbf{x}^* has two fractional variables, they correspond to two adjacent levels in Ω_i for a single keyword i .*

When an advertiser has a flexible budget and wishes to perform a trade-off analysis between revenue and cost, the usual practice is to solve the problem with different budget values and compare the solutions. The sorting algorithm can provide solutions for different budget values on a single run. Observe that the marginal efficiency of a bid level does not depend on the budget. Therefore if the problem

is solved for the maximum budget value in consideration, the amount spend by the algorithm can be tracked to obtain solutions for all the other budget values. In addition, although we focus on expected revenue maximization, the sorting algorithm can be modified to maximize expected profit (defined as revenue minus cost for a bid level) by updating the stopping criterion. To maximize expected profit, we simply terminate the algorithm when the marginal efficiency of a selected level is less than or equal to 1. We note that these desirable properties of the sorting algorithm (i.e., solving for different budget values on a single run and maximizing expected profit) are preserved in our column generation algorithm.

2.2.3 Linear Algorithm for the MCKP-LP

[Dyer \[1984\]](#) and [Zemel \[1984\]](#) independently provide an $O(n)$ time algorithm for the MCKP-LP, which we refer to as the DZ algorithm. [Dyer \[1984\]](#) and [Zemel \[1984\]](#) observed there are only two variables in each constraint in the dual of the MCKP-LP. Using this observation, they developed a linear time algorithm (that avoids sorting) to solve the dual. We describe the main ideas of the algorithm based on a primal variant of the DZ algorithm due to [Pisinger \[1995\]](#).

The algorithm is focused on finding the optimal dual value θ , and it does so by guessing a value of θ and with it an associated feasible dual solution. This dual solution is used to construct a primal solution using the complementary slackness conditions. If the primal solution is feasible the algorithm terminates. If the primal solution is infeasible (i.e., the guessed value of θ is not optimal) the algorithm ac-

cordingly adjusts the guess and continues until at termination it obtains the optimal solution.

In one iteration of the algorithm, each keyword is considered. For a given keyword, it pairs all of its bid levels (if there are an odd number of levels one level is left unpaired). Let j' and j'' be two bid levels for keyword i that are arbitrarily paired, and assume without loss of generality $c_{ij'} < c_{ij''}$ (if $c_{ij'} = c_{ij''}$, the level with the smaller cost can be deleted from the problem). The algorithm calculates their associated slope $\frac{r_{ij''} - r_{ij'}}{c_{ij''} - c_{ij'}}$. The algorithm then determines the median slope over all keywords and pairs. The median slope provides their “guess” for the value of θ .

With this guess the other dual variables are set as $\lambda_i = \max_{j \in P_i} \{r_{ij} - \theta c_{ij}\}$, and let $Q_i(\theta) = \arg \max_{j \in P_i} \{r_{ij} - \theta c_{ij}\}$ denote the set of bid levels of keyword i that achieve the value λ_i (based on the dual solution constructed and complementary slackness these are the only levels of keywords i that can be non-zero in the primal solution). Next, $s_i = \arg \min_{j \in Q_i(\theta)} \{c_{ij}\}$ identifies the bid level in $Q_i(\theta)$ with the smallest cost, and $l_i = \arg \max_{j \in Q_i(\theta)} \{c_{ij}\}$ identifies the bid level in $Q_i(\theta)$ with the largest cost (these may be the same or different items). If the budget B is less than $\sum_{i \in K} c_{is_i}$, or the budget B is greater than $\sum_{i \in K} c_{il_i}$ then we do not have the optimal solution (as any solution constructed will exceed, or not achieve the budget respectively). Otherwise, we have the optimal solution to the problem. In the case that $B < \sum_{i \in K} c_{is_i}$, for every pair (ij', ij'') with $\frac{r_{ij''} - r_{ij'}}{c_{ij''} - c_{ij'}} \leq \theta$ it deletes bid level j'' (as it is possible to show this will not be in the optimal solution. In the case that $B > \sum_{i \in K} c_{il_i}$ for every pair (ij', ij'') with $\frac{r_{ij''} - r_{ij'}}{c_{ij''} - c_{ij'}} \geq \theta$ it deletes bid level j' (as it is possible to show this will not be in the optimal solution). The procedure continues

iteratively until we find the optimal solution.

Each iteration can be done in linear time (as a function of the total number of levels n in the problem). By showing that at least a sixth of the levels remaining are deleted in each iteration; they show that the overall running time of the algorithm is linear. Note that though the algorithm has a linear theoretical run time, it needs to scan through all the remaining levels in every iteration to evaluate the removal criteria. Therefore, it has a large constant factor.

In essence, the algorithm works as a column “removal” algorithm. Considering only one bid level is selected for each keyword, in the worst case the algorithm needs to remove all levels except the optimal level from every keyword. In the advertiser’s problem, the budget is small relative to the maximum amount possible to spend, we therefore consider a column generation approach especially suited for the advertiser’s problem. This, as we demonstrate in our computational experiments, results in significant gains in computational time while solving the OAPOP.

2.3 Column Generation Algorithm

The advertiser’s problem can contain tens of millions of keywords and may need to be solved multiple times in a day. The daily budget is typically very small relative to the maximum amount possible to spend (the value is generally in the order of up to 5%, and never more than 20% of the maximum amount possible to spend, for the vast majority of portfolios). In this setting, the budget will likely be exhausted before the latter levels on the convex hull of a keyword can be se-

lected by the sorting algorithm. In fact, for a significant fraction of keywords, the optimal solution may be to select the zero (i.e. minimal) bid level. We therefore employ a column generation algorithm that generates levels of the convex hull only as needed. The column generation algorithm has two phases. The initialization phase and the selection/generation phase. In the initialization phase, an initial solution to the MCKP-LP is obtained by considering a subset of columns (levels) in the problem. The procedure is described in Section 2.3.1. In the selection/generation phase (described in Section 2.3.2) we use this initial solution along with delayed column generation to select levels in the same order as the sorting algorithm (i.e. without explicitly generating the entire upper convex hull and the associated sorting). In each iteration of the delayed column generation procedure, the bid level with the largest marginal efficiency is selected, and the bid level necessary to infer the selection of the bid level with the largest marginal efficiency in the next iteration is generated. The procedure terminates when the budget is exhausted and the optimal solution to the MCKP-LP is obtained.

2.3.1 Initial Solution Procedure

In the initial solution procedure, we set the first two levels in the convex hull for every keyword as the initial set of columns. Then we solve the MCKP-LP for the initial set of columns and calculate the value of the dual variable θ . The first two levels in the convex hull for keyword $i \in K$ are easily identified. By Proposition 2.2.2, the first level is always the zero bid level, i.e., $(0) = 0$, and the

second level is the level with the largest $\{r_{ij}/c_{ij}\}$ for $j \in P_i$. Let $\Omega' \subseteq \Omega$ such that $\Omega'_i \subseteq \Omega_i$ for every keyword $i \in K$. We set $\Omega'_i = \{(0), (1)\}$ for each keyword $i \in K$ and set $\Omega' = \{(i, j) \mid i \in K, j \in \Omega'_i\}$ as the initial set of levels. Let $\theta(\Omega')$ denote the value of the optimal dual variable θ for the MCKP-LP(Ω'). Define $\Upsilon = \{(i, j) \mid \delta_{ij} \geq \theta(\Omega'), (i, j) \in \Omega'\}$ where Υ represents the levels selected in the optimal solution for the initial set of columns.

To solve the MCKP-LP(Ω'), we first check whether there are enough levels in Ω' to exhaust the budget. If $\sum_{i \in K} \bar{c}_{i(1)} < B$, then the optimal solution to the MCKP-LP(Ω') can be obtained by setting $\Upsilon = \{(i, (1)) \mid i \in K\}$ and $\theta(\Omega') = 0$ by complementary slackness. Otherwise the MCKP-LP(Ω') can be solved optimally via the Split Procedure outlined in Algorithm 2 by setting $M = \Omega'$ and $C = B$. Essentially, we are trying to find via binary search the marginal efficiency of the last level selected from the portfolio (i.e., the optimal dual variable $\theta(\Omega')$). Since there is only one value for marginal efficiency for each keyword, this can be accomplished by binary search. Note that $X = [X_1 \mid X_2 \mid X_3 \mid \dots]_\delta$ indicates X is partitioned such that $\min_{(i,j) \in X_k} \{\delta_{ij}\} \geq \max_{(i,j) \in X_{k+1}} \{\delta_{ij}\}$. Since the median of M can be found in $O(|M|)$ time, the split procedure runs in $O(|K|)$ time.

2.3.2 Column Generation Procedure

The delayed column generation procedure takes the solution obtained from the initial solution procedure as input. It adds the column with the largest reduced cost to the problem, then resolves the problem and repeats this procedure until we

Algorithm 2 Split Procedure

```

1: Input:  $M, C$ 
2: find the median  $\delta_{i'(1)}$  in  $M$ 
3: reorder  $M$  such that  $M = [M^+ \mid (i', (1)) \mid M^-]_\delta$ 
4: if  $\sum_{(i,(1)) \in M^+} c_{i(1)} > C$  then
5:   Split Procedure( $M^+, C$ )
6: else if  $\sum_{(i,(1)) \in M^+ \cup \{i'\}} c_{i(1)} \leq C$  then
7:    $C := C - \sum_{(i,(1)) \in M^+ \cup \{i'\}} c_{i(1)}$ 
8:   Split Procedure( $M^-, C$ )
9: else
10:   $\theta(\Omega') := \delta_{i'(1)}$ 
11: end if

```

obtain the optimal solution to the MCKP-LP(Ω). While this is conceptually the idea, this approach is computationally efficient in terms of implementation only if we minimize the burden associated with finding the column with the largest reduced cost.

We first observe that the column that is not in the current set of columns (i.e., $\Omega \setminus \Omega'$) with the largest reduced cost (for constraint set (2.6)) corresponds to the keyword and level with the largest marginal efficiency. For the current set of columns Ω' , the reduced cost for keyword i and level j is $r_{ij} - c_{ij}\theta(\Omega') - \lambda_i$. Recall that $\lambda_i = r_{i(j')} - \theta(\Omega')c_{i(j')}$ where $(j') \in \Omega'_i$ is the last level selected for keyword i . In a column generation procedure, we wish to select the level with the largest reduced cost. In other words, we wish to select $\arg \max_{(i,j) \in \Omega \setminus \Omega'} \left\{ \frac{r_{ij} - r_{i(j')}}{c_{ij} - c_{i(j')}} \mid \frac{r_{ij} - r_{i(j')}}{c_{ij} - c_{i(j')}} > \theta(\Omega') \right\}$. For keyword i , the level in $\Omega_i \setminus \Omega'_i$ with the largest reduced cost, by Proposition 2.2.3, corresponds to the level with the largest marginal efficiency, i.e., $\delta_{i(j'+1)} = \max_{j \in \Omega_i \setminus \Omega'_i} \left\{ \frac{r_{ij} - r_{i(j')}}{c_{ij} - c_{i(j')}} \right\}$. Therefore, the keyword and level in $\Omega \setminus \Omega'$ with the largest reduced cost is the keyword and level with the largest marginal efficiency, i.e., $\arg \max_{i \in K} \delta_{i(j'+1)}$.

Rather than update the solution after generating a column (which requires insertion into the list of columns in the problem), we will use the insight gained from the delayed column generation procedure to replicate the order in which keywords and levels are selected in the sorting algorithm (without the overhead of generating the entire convex hull and sorting).

2.3.2.1 Generating Columns

To replicate the sorting algorithm's order of selecting keywords and levels, we create and maintain a "candidate list" from which selections are performed. After the initial solution procedure, we designate Υ as the candidate list. Without loss of generality, suppose the marginal efficiency of levels in Υ after the initial solution procedure are ordered and labeled as $\delta_{1(1)} \geq \delta_{2(1)} \geq \dots$. The first level selected by the column generation procedure corresponds to $\delta_{1(1)}$, i.e., $(1, (1))$, since it has the largest marginal efficiency in the problem. Then the second level to be selected corresponds to $\max\{\delta_{1(2)}, \delta_{2(1)}\}$. However, level $(1, (2))$ is not in Υ and therefore needs to be generated. Note that generating level $(1, (2))$ does not require us to derive the entire upper convex hull for keyword 1. The next level, i.e., level $(1, (2))$, in the convex hull corresponds to the level that has larger cost than the current level, i.e., level $(1, (1))$, and has the largest revenue-cost ratio in relation to the current level. In other words, if the current level is $(i, (j))$, then the next level in the convex hull is $(j + 1) = \arg \max_{j' \in P_i | c_{ij'} > c_{i(j)}} \frac{r_{ij'} - r_{i(j)}}{c_{ij'} - c_{i(j)}}$. After generating level $(1, (2))$ and inserting it into Υ , we can determine the second level to be selected. Suppose

$\delta_{1(2)} > \delta_{2(1)}$, then the second level the procedure selects is $(1, (2))$ and it is inserted into Ω' . It is important to note that by asserting $\delta_{1(2)} > \delta_{2(1)}$, we can conclude level $(1, (2))$ has the largest reduced cost (i.e., largest marginal efficiency) in $\Omega \setminus \Omega'$. The third level to be selected corresponds to $\max\{\delta_{1(3)}, \delta_{2(1)}\}$. After generating level $(1, (3))$ and inserting it into Υ , we can now determine the third level to be selected. The column generation procedure selects levels in this fashion, and generates and inserts levels into Υ that are necessary to infer the subsequent selections. Each selected level is removed from Υ .

After the initial solution procedure, the levels in Υ are in no particular order of δ_{ij} . We can sort and maintain levels in Υ in nonincreasing order of δ_{ij} , then the level with the largest marginal efficiency can be accessed in $O(1)$ time. Considering some levels in Υ may never be selected by the procedure, maintaining the entire list as a sorted list may not be worthwhile. In addition, levels generated by the column generation procedure are also inserted into Υ , which further complicates the maintenance issue. To ensure the level with the largest marginal efficiency in the candidate list can be accessed in $O(1)$ time, we create two sublists Λ and Π such that $\Upsilon = \Lambda \cup \Pi$. We designate Π as the candidate sublist for levels generated by the column generation procedure and Λ as the candidate sublist for levels inherited from the initial solution. We set $\Lambda = \Upsilon$ and $\Pi = \emptyset$ after the initial solution procedure. The levels selected from Π are inserted into Ω' thus expanding the set of current levels. On the other hand, the levels selected from Λ are already in Ω' since they were inherited from the initial solution.

Maintaining Λ and Π warrants different approaches since only extractions

are performed on Λ whereas both insertions and extractions are performed on Π . Therefore, we maintain Π as a priority queue implemented as a binary heap. A binary heap provides $O(1)$ time access to its largest element and allows for $O(\log |\Pi|)$ time insertion and extraction. In contrast, since we only perform extractions on Λ , we maintain it as a partially sorted list. We partition and reorder Λ into κ buckets such that $\Lambda = [\Lambda_1 \mid \cdots \mid \Lambda_\kappa]_\delta$. The partitioning can be achieved by placing each $(i, (1)) \in \Lambda$ into one of κ buckets where each bucket corresponds to an interval of δ . We have picked uniform bucket intervals assuming no prior knowledge of the distribution of $\delta_{i(1)}$, $(i, (1)) \in \Lambda$. Note that placing all items in their respective buckets takes $O(|\Lambda|)$ regardless of the number of buckets or interval lengths. Let

$$\gamma = \frac{\max_{(i,(1)) \in \Lambda} \delta_{i(1)} - \min_{(i,(1)) \in \Lambda} \delta_{i(1)}}{\kappa}$$

then Λ_k is defined as

$$\Lambda_k = \{(i', (1)) \mid \delta_{i'(1)} \in \Lambda, \max_{(i,(1)) \in \Lambda} \delta_{i(1)} - (k-1)\gamma \geq \delta_{i'(1)} \geq \max_{(i,(1)) \in \Lambda} \delta_{i(1)} - k\gamma\}$$

In our column generation procedure, no level from Λ_k can be selected before every level in Λ_{k-1} is selected. The procedure only needs to search inside one bucket to find the level that can be selected. In the first iteration of the column generation procedure, the level with the largest marginal efficiency is in Λ_1 . If Λ_1 is fully sorted, then the largest element in Λ_1 can be accessed in $O(1)$ time. Therefore, we start the procedure by fully sorting Λ_1 , once every level is selected from Λ_1 , then we fully sort

Λ_2 , and so on. With this approach, the buckets are only sorted when needed. To identify the level with the largest marginal efficiency in Υ , we compare $\max_{(i,j) \in \Lambda} \delta_{ij}$ with $\max_{(i,j) \in \Pi} \delta_{ij}$, which can be done in $O(1) + O(1) = O(1)$ time.

2.3.2.2 Removing Columns

It is easy to observe that $\theta(\Omega') \leq \theta(\Omega'')$ if $\Omega' \subseteq \Omega''$. So as columns are added to the problem $\theta(\Omega')$ will only increase. Thus, columns with negative reduced cost need not be considered for selection. Notice this implies all keywords for which $x_{i(0)} = 1$ in the initial solution will stay at the zero bid level in the optimal solution to the problem. Every time a column with positive reduced cost is added to the problem, we need to update the solution and determine $\theta(\Omega')$. Rather than doing so, we calculate a dual lower bound $\underline{\theta}$ for $\theta(\Omega')$ such that $\underline{\theta} \leq \theta(\Omega')$ and update $\underline{\theta}$ using the bucket structure of Λ . This effectively allows us to remove columns in batches. Recall that $\theta(\Omega')$ cannot decrease as Ω' expands, therefore $\underline{\theta}$ is a valid lower bound for any $\theta(\Omega'')$ where $\Omega' \subseteq \Omega''$. In the column generation procedure, after level (i, j) is generated, we compare the marginal efficiency of the level with $\underline{\theta}$. If $\delta_{ij} \leq \underline{\theta}$, then there is no need to insert the level into the candidate sublist Π . We now elaborate on updating the dual lower bound $\underline{\theta}$.

Let ω denote the set of levels selected from Π and define $\bar{c}(X) = \sum_{(i,j) \in X} \bar{c}_{ij}$ where X is an arbitrary set of levels. After the initial solution procedure, $B - \bar{c}(\Lambda)$ denotes the leftover budget. Initially, $\omega = \emptyset$ and we insert levels selected from Π into ω . Suppose $\bar{c}(\omega) \geq B - \bar{c}(\Lambda)$ is satisfied in some iteration of the column generation

procedure, then there are now enough levels in Ω' added from Π to exhaust the budget. Therefore we set $\underline{\theta} = \min_{(i,j) \in \Lambda_\kappa} \delta_{ij}$. Note that if $B - \bar{c}(\Lambda) = 0$, i.e., the budget is exhausted in the initial solution, then $\bar{c}(\omega) \geq B - \bar{c}(\Lambda)$ is satisfied after the initial solution. Suppose $\bar{c}(\omega) \geq B - \bar{c}(\Lambda) + \bar{c}(\Lambda_\kappa)$ is satisfied in some iteration. Then the levels in ω have large enough total marginal cost that if we were to solve the MCKP-LP(Ω'), the budget would be exhausted before any level from Λ_κ can be selected. Therefore, we have $\theta(\Omega') \geq \delta_{ij}$ for every $(i, j) \in \Lambda_\kappa$. Thus, the entire bucket Λ_κ can be removed from consideration since no level from Λ_κ will be selected. However, calculating $\theta(\Omega')$ exactly would require us to solve the MCKP-LP(Ω'). Instead we set the dual lower bound $\underline{\theta} = \min_{(i,j) \in \Lambda_{\kappa-1}} \delta_{ij}$ since $\theta(\Omega') \geq \min_{(i,j) \in \Lambda_{\kappa-1}} \delta_{ij}$, that is, the marginal efficiency of the last level selected when solving the MCKP-LP(Ω') is at least $\min_{(i,j) \in \Lambda_{\kappa-1}} \delta_{ij}$. After updating $\underline{\theta}$, we repeat the process for bucket $\Lambda_{\kappa-1}$ by checking whether $\bar{c}(\omega) \geq B - \bar{c}(\Lambda) + \bar{c}(\Lambda_\kappa) + \bar{c}(\Lambda_{\kappa-1})$ is satisfied in some iteration. In general, to check if we can remove bucket Λ_k from consideration, we check whether $\bar{c}(\omega) \geq B - \bar{c}(\Lambda) + \sum_{t=k}^\kappa \bar{c}(\Lambda_t)$ is satisfied. The bucket structure of Λ allows us to update the dual lower bound in this fashion. Note that $\bar{c}(\Lambda_k)$ and $\min_{(i,j) \in \Lambda_k} \delta_{ij}$ can be determined for every $k = 1, \dots, \kappa$ as the buckets are created, thus require no additional computations.

2.3.2.3 Overall Implementation and Running Time Analysis

The column generation procedure is outlined in Algorithm 3. The procedure takes the initial set of levels Ω' , the values for variables \mathbf{x} , the set of selected levels

$\Delta(\Omega')$, the set of levels removed from consideration $\bar{\Delta}(\Omega')$, the remaining budget C , the total revenue R , the dual lower bound $\underline{\theta}$, the candidate sublists Λ and Π , ω , ρ , and k as inputs. Before executing the procedure, we set $\mathbf{x} = \{x_{ij} \mid x_{i(0)} = 1, x_{ij} = 0, i \in K, j \in P_i \setminus \{(0)\}\}$, $\Delta(\Omega') = \{(i, (0)) \mid i \in K\}$, $\bar{\Delta}(\Omega') = \emptyset$, $C = B$, and $R = 0$, $\underline{\theta} = \theta(\Omega')$, $\Lambda = \Upsilon$, $\Pi = \emptyset$, $\omega = \emptyset$. If $B - \bar{c}(\Lambda) > 0$ after the initial solution procedure, then we set $k = \kappa + 1$ and $\rho = B - \bar{c}(\Lambda)$, otherwise we set $k = \kappa$ and $\rho = \bar{c}(\Lambda_\kappa)$. In line 3 of Algorithm 3, the level with the largest marginal efficiency is selected from the candidate list $\Lambda \cup \Pi$. When a level is selected, the remaining budget and total revenue are updated as in the sorting algorithm. Between lines 15 and 18, the dual lower bound is updated and levels that cannot be selected are removed from Λ . In line 24 of Algorithm 3, the level $(i, (j + 1))$ is generated by calling `Generate Level((j + 1))` outlined in Algorithm 4. In line 25, the marginal efficiency of the generated level is compared to the dual lower bound. However, the level is only added to Π if its marginal efficiency is larger than the dual lower bound. The column generation procedure terminates when the budget is exhausted.

The column generation procedure outlined in Algorithm 3 takes the initial solution and provides an optimal solution to MCKP-LP(Ω). Recall that Δ is the set of levels selected by the sorting algorithm and $\bar{\Delta}$ is the set of remaining levels (i.e., levels that are not selected by the sorting algorithm). By definition, $\Delta \cup \bar{\Delta} = \Omega$. When the column generation algorithm terminates, $\Delta(\Omega') = \Delta$ and is in sorted order of δ_{ij} . On the other hand, $\bar{\Delta}(\Omega') \subseteq \bar{\Delta}$ since the column generation algorithm only generates the levels in the convex hull as needed. Note that the column generation algorithm can be modified in the same manner as the sorting algorithm to maximize

expected profit. Furthermore, like the sorting algorithm, the column generation algorithm only needs to be executed once to solve for different budget values as described in Section 2.2. This is possible since the column generation algorithm performs identical selections to that of the sorting algorithm. However, the DZ algorithm cannot easily be modified to maximize profit and can only provide a solution for one budget value.

Algorithm 3 Column Generation Procedure

```

1: Input:  $\Omega', \mathbf{x}, \Delta(\Omega'), \bar{\Delta}(\Omega'), R, C, \Lambda, \Pi, \underline{\theta}, \omega, \rho, k,$ 
2: while  $C > 0$  do
3:    $(i, (j)) = \arg \max_{(i, (j)) \in \Lambda \cup \Pi} \delta_{i(j)}$ 
4:    $\Delta(\Omega') := \Delta(\Omega') \cup \{(i, (j))\}$ 
5:   if  $C - \bar{c}_{i(j)} < 0$  then
6:     set  $x_{i(j)} := C/\bar{c}_{i(j)}, x_{i(j-1)} := 1 - x_{i(j)}, R := R + \bar{r}_{i(j)}x_{i(j)}, C := 0$ 
7:   else
8:     set  $x_{i(j)} := 1, x_{i(j-1)} := 0, R := R + \bar{r}_{i(j)}, C := C - \bar{c}_{i(j)}$ 
9:   end if
10:  if  $(i, (j)) \in \Lambda$  then
11:     $\Lambda := \Lambda \setminus \{(i, (j))\}$ 
12:  else
13:     $\Pi := \Pi \setminus \{(i, (j))\}, \omega := \omega \cup \{(i, (j))\}$ 
14:     $\Omega'_i := \Omega'_i \cup \{(i, (j))\}$ 
15:    if  $\bar{c}(\omega) > \rho$  and  $k > 0$  then
16:       $\bar{\Delta}(\Omega') := \bar{\Delta}(\Omega') \cup \Lambda_k, \Lambda := \Lambda \setminus \Lambda_k, k := k - 1$ 
17:       $\underline{\theta} := \min_{(i, j) \in \Lambda_k} \delta_{ij}, \rho := \rho + \bar{c}(\Lambda_k)$ 
18:    end if
19:  end if
20:  if  $C = 0$  then
21:     $\theta(\Omega) := \delta_{i(j)}$ 
22:     $\bar{\Delta}(\Omega') := \bar{\Delta}(\Omega') \cup \Lambda \cup \Pi$ 
23:  else
24:    Generate Level( $\Omega'_i, (j + 1)$ )
25:    if  $\delta_{i, (j+1)} > \underline{\theta}$  then
26:       $\Pi := \Pi \cup \{(i, (j + 1))\}$ 
27:    else
28:       $\bar{\Delta}(\Omega') := \bar{\Delta}(\Omega') \cup \{(i, (j + 1))\}$ 
29:    end if
30:  end if
31: end while

```

Algorithm 4 Generate Level

- 1: **Input:** $(j + 1)$
 - 2: reorder P_i such that $P_i = [P_i^+ \mid c_{i(j)} \mid P_i^-]_c$
 - 3: $P_i := P_i^+$
 - 4: $(j + 1) = \arg \max_{j' \in P_i} \frac{r_{ij'} - r_{i(j)}}{c_{ij'} - c_{i(j)}}$.
-

When the column generation algorithm terminates, a feasible integer solution $\mathbf{x}^{\{0,1\}}$ can be obtained by rounding the fractional MCKP-LP solution, i.e., setting $\mathbf{x}^{\{0,1\}} = \mathbf{x}$, $x_{s(\ell)}^{\{0,1\}} = 1$, and $x_{s(\ell+1)}^{\{0,1\}} = 0$, where $(s, (\ell+1))$ denotes the last level selected by the column generation procedure. The objective value corresponding to $\mathbf{x}^{\{0,1\}}$ is calculated by $R^{\{0,1\}} = R - \bar{r}_{s(\ell+1)}x_{s(\ell+1)}$. Therefore, $R^{\{0,1\}}$ is at most $\bar{r}_{s(\ell+1)}x_{s(\ell+1)}$ away from the optimal integer solution. Commercial solvers typically decide integer optimality by comparing the best available bound and the best integer solution value. If the best integer solution value is within a certain percentage (also called the optimality tolerance) of the best available bound, then the best integer solution is declared optimal. Regardless of the number of keywords, the column generation algorithm finds an optimal solution to the MCKP-LP with at most two fractional variables. Due to the large number of keywords in the online advertising application, we typically have $R \gg \max_{(i,j) \in \Omega} \bar{r}_{ij}$. Therefore the difference between the rounded integer revenue $R^{\{0,1\}}$ and LP relaxation revenue R is very small compared to R . In addition, as the number of keywords increase, the marginal revenue loss incurred by rounding one fractional variable decreases. In fact, in every instance we have experimented, the rounded integer solution was also optimal for the MCKP given the same optimality tolerance used by CPLEX. However, in other settings different from the OAPOP, the gap between the rounded integer solution and the LP relaxation

bound may not be within the optimality tolerance. We therefore provide a branch-and-price algorithm in Section 2.4 to optimally solve the MCKP using the column generation algorithm. The branch-and-price algorithm uses a partition branching rule on the last keyword and level selected by the algorithm. After branching, we update Λ and Π to maintain their structure, which in turn helps us rapidly solve the MCKP-LP for the child node.

In Algorithm 4, reordering P_i takes $O(|P_i|)$, identifying $(j+1)$ also takes $O(|P_i|)$ and the levels cannot be in Ω_i are deleted from P_i . In the worst case, $P_i = \Omega_i$ and all levels are generated, which takes $O(|P_i|) + O(|P_i| - 1) + \dots + O(1) = O(|P_i|^2)$ for keyword i . However, in the advertiser's problem, it is very likely most levels will not be generated. The running time of the column generation procedure depends on the number of levels selected. Suppose given budget B , the number of levels selected by the procedure can be represented by some function $f(B) = \sum_{i \in K} f_i(B)$. The running time of the column generation algorithm, i.e., the total running time of the initial solution procedure and the column generation procedure, can be expressed as $O(n + |K| + |K| \log |K| + f(B) \log f(B) + \sum_{i \in K} f_i(B) P_i)$. $O(n)$ time is spent identifying $(i, (1))$ for all $i \in K$. Split Procedure takes $O(K)$ time. $O(|K| \log |K| + f(B) \log f(B))$ time is spent maintaining Λ and Π , though maintaining Λ may be much faster depending on the interval selection, κ , and the number of levels removed due to dual lower bound. Finally, it takes $O(\sum_{i \in K} f_i(B) P_i)$ to generate levels. At first glance, the running time of the column generation algorithm may seem larger than the running time of the DZ algorithm, however the column generation algorithm can potentially run very fast where B is small relative to the maximum

amount possible to spend. In fact, we demonstrate the relationship between B and the running time of the column generation algorithm in Section 2.5.

2.4 Branch-and-price Algorithm

The MCKP can be solved with a branch-and-price algorithm where upper bounds are obtained from the column generation algorithm and the lower bounds are obtained from rounded integer solutions. In an optimal solution to MCKP-LP, only two convex hull levels from the same keyword s can be non-zero, and if two are nonzero, they must be adjacent levels $(\ell), (\ell+1) \in \Omega_s$. We therefore use a branching strategy (originally proposed in Beale and Tomlin [1970]) that splits P_s into two sets $P_s^{(<)}$ and $P_s^{(>)}$ such that $P_s^{(<)} = \{j \mid c_{sj} \leq c_{s(\ell)}, j \in P_s\}$ and $P_s^{(>)} = P_s \setminus P_s^{(<)}$. By splitting P_s , we create two child nodes corresponding to $P_s^{(<)}$ (left branch) and $P_s^{(>)}$ (right branch) as depicted in Figure 2.6. We now describe how easily the CG algorithm can be applied to both children, taking advantage of the computational effort already incurred in computing the solution for the parent node. Note that this cannot be done in the DZ algorithm in an efficient way; making it unsuitable to use in a branch-and-bound approach.

For both child nodes, the corresponding MCKP-LP can be solved by the column generation procedure outlined in Algorithm 3 with two minor modifications. First, we should not execute line 22 during the branch-and-price algorithm since we would like to maintain the structures of Λ and Π for the child node. Second, in line 3, instead of selecting $(i, (j)) = \arg \max_{(i, (j)) \in \Lambda \cup \Pi} \delta_{i(j)}$, we need to select

$(i, (j)) = \arg \max_{(i, (j)) \in \Lambda \cup \Pi \cup \bar{\Delta}(\Omega')}$ $\delta_{i(j)}$ since the levels pushed into $\bar{\Delta}(\Omega')$ because of the dual lower bound may be eligible to be selected after branching. Feasible integer solutions can be obtained by rounding the LP relaxation solution as described in Section 2.3.

For the parent node of both branches, P_s^p corresponds to the set of levels, Ω_s^p corresponds to the upper convex hull, and (c_{sj}^p, r_{sj}^p) correspond to the cost-revenue pairs for keyword s . B^p denotes the budget and $\mathbf{x}^p, \Delta(\Omega')^p, \Lambda^p, \Pi^p, R^p$, and k^p denote the final values of the inputs after the column generation terminates for the parent node.

On the left branch, the computations for the child node are fairly straightforward. Branching on $P_s^{(<)}$ corresponds to adding the constraint $\sum_{j \in P_s^{(>)}} x_{sj} = 0$ to the problem. We remove the levels that cannot be in the solution by setting $P_s = P_s^{(<)p}$. There is no need to derive Ω'_s from scratch, it can be obtained from the parent node by setting $\Omega'_s = \{j \mid c_{sj} \leq c_{s(\ell)}, j \in \Omega_s^p\}$. However, we have to remove $(s, (\ell + 1))$ from $\Delta(\Omega')$ and $(s, (\ell + 2))$ from Π or $\bar{\Delta}(\Omega')$ since $(\ell + 1), (\ell + 2) \notin \Omega'_s$, we set $\Delta(\Omega') = \Delta(\Omega')^p \setminus \{(s, (\ell + 1))\}$, $\bar{\Delta}(\Omega') = \bar{\Delta}(\Omega')^p \setminus \{(s, (\ell + 2))\}$ and $\Pi = \Pi^p \setminus \{(s, (\ell + 2))\}$. The solution \mathbf{x} remains the same after branching except for x_{sj} where $c_{sj} \geq c_{s(\ell)}$, $j \in P_s^p$ so we set $\mathbf{x} = \mathbf{x}^p$, then set $x_{s(\ell)} = 1$ and remove x_{sj} from \mathbf{x} for $c_{sj} > c_{s(\ell)}$, $j \in P_s^p$. We reduce the objective value $R = R^p - \bar{r}_{s(\ell+1)} x_{s(\ell+1)}^p$ since R^p contains the marginal revenue generated by level $(\ell + 1)$. Similarly, we restore the remaining budget to reflect the removal of level $(\ell + 1)$ by setting $C = \bar{c}_{s(\ell+1)} x_{s(\ell+1)}^p$. Finally, we set $\Lambda = \Lambda^p$ $B = B^p$, $k = 0$, $\underline{\theta} = 0$ and execute Algorithm 3.

On the right branch, branching on $P_s^{(>)}$ corresponds to adding the constraint

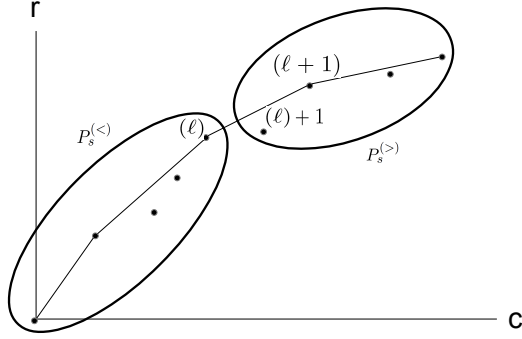


Figure 2.6: Left branch ($P_s^{(<)}$) and right branch ($P_s^{(>)}$) for keyword s and fractional levels $(\ell), (\ell + 1) \in \Omega_s$.

$\sum_{j \in P_s^{(<)}} x_{sj} = 0$ to the problem. We set $P_s = P_s^{(<)p}$, unlike the left branch, the levels in P_s no longer involve the zero bid level $(c_{s0}, r_{s0}) = (0, 0)$. We designate level $j' = \arg \min_{j \in P_s^p | c_{sj} > c_{s(\ell)}} c_{sj}$ as the zero bid level by setting $c_{sj} = c_{sj}^p - c_{sj'}^p$ and $r_{sj} = r_{sj}^p - r_{sj'}^p$ for $j \in P_s$ such that $c_{sj} > c_{s(\ell)}$. Then we update the budget to $B = B^p - c_{s0}$. With the change in the budget, the solution \mathbf{x}' can no longer be used for the child node. We reset the solution $\mathbf{x} = \{x_{ij} \mid x_{i(0)} = 1, x_{ij} = 0, i \in K, j \in \Omega'_i \setminus \{(0)\}\}$. However, the solution can be calculated by using the lists from the parent node. We remove the levels corresponding to keyword s from $\Delta(\Omega')$, $\bar{\Delta}(\Omega')$ and Π by setting $\Delta(\Omega') = \Delta(\Omega')^p \setminus \{(s, (j)) \mid 1 \leq j \leq \ell + 1\}$, $\bar{\Delta}(\Omega') = \bar{\Delta}(\Omega')^p \setminus \{(s, (\ell + 2))\}$ and $\Pi = \Pi^p \setminus \{(s, (\ell + 2))\}$. We find the new level (1) for keyword s by calculating $(1) = \arg \max_{j \in P_s} \{r_{sj}/c_{sj}\}$ and set $\Pi = \Pi \cup \{\delta_{s(1)}\}$. Then we set $\Lambda = \Lambda^p \cup \Delta(\Omega')$ and $\Delta(\Omega') = \emptyset$ respectively. Note that after the column generation algorithm terminates for the parent node, we have $\min_{(i,j) \in \Delta(\Omega')^p} \delta_{ij} \geq \max_{(i,j) \in \Lambda^p} \delta_{ij}$. Therefore setting $\Lambda = \Lambda^p \cup \Delta$ can be done in $O(1)$ time. Finally, we set $R = 0$, $C = B$, and $k = 0$, $\theta = 0$ and execute Algorithm 3.

2.5 Computational Results

To test the performance of the column generation (CG) algorithm for the OAPOP, we perform computational experiments on a large set of simulated online advertising instances. Section 2.5.1 describes our procedure to generate simulated online advertising (OA) instances; based on sample real world online advertising data collected from Google Keyword Planner. These instances contain anywhere between 1 million and 50 million keywords, and between 10 and 50 bid levels for each keyword; resulting in MCKP problem instances with as many as 2.55 billion variables and 50 million constraints. While our focus is to understand the performance of the CG algorithm on OA instances, we also wanted to study the behavior of the CG algorithm on other types of MCKP instances that have been considered previously in the literature. To this end, in addition to the OA instances we generated MCKP instances similar to those considered previously in the literature [see [Pisinger, 1995](#)]. Since literature instances are not of the massive scale that we are able to solve with the CG algorithm we used existing approaches in the literature to generate new MCKP instances (section 2.5.3 discusses these instances). For both sets of instances, we compare the performance of the CG algorithm with that of the DZ algorithm [our implementation follows [Pisinger, 1995](#)]. We discuss these results in Sections 2.5.2 and 2.5.3. Both the CG and DZ algorithm are implemented in C++ and all computational experiments are performed on a computer with Intel Xeon E5-1620 v3 CPU @ 3.50 GHz and 32GB RAM running 64-bit Windows 7.

2.5.1 Generating Online Advertising Instances

The advertiser’s problem can have hundreds of thousands to tens of millions of keywords. In online advertising, we experiment with $|K| = \{1, 5, 10, 25, 50\}$ million keywords and $|P_i| = |P| = \{10, 20, 30, 40, 50\}$ bid levels excluding the zero bid level. We set $c_{i0} = 0$ and $r_{i0} = 0$ for all $i \in K$ in every type of instance. Note that an instance with $|K| = 50$ million and $|P| = 50$ has 2.55 billion variables. For the CG algorithm, we did not devote special effort to tune κ . After preliminary computational experiments, we set $\kappa = 100$ and use uniform bucket intervals as described in Section 2.3.2 for all instances. To generate an instance for the OAPOP, we need to generate cost-revenue pairs for every keyword.

To generate online advertising instances, we sampled keywords from the Keyword Planner tool of Google Adwords. Given keyword i , bid b_{ij} , a time period, and a geographical location, the Keyword Planner tool provides an estimate for the number of clicks, number of impressions, cost, click-through-rate (CTR), average cost per click (CPC), and average position on the screen. The advertiser can estimate a revenue-per-click (RPC) for each keyword using historical browsing data. Using the estimated cost and the estimated number of clicks provided by the Keyword Planner tool, we generate cost-revenue pairs. For a bid amount b_{ij} , let φ_{ij} be the expected number of clicks, c_{ij} be the expected cost and let RPC_i be the revenue-per-click for keyword i , then the expected revenue can be calculated as $r_{ij} = RPC \times \varphi_{ij}$. In the MCKP, every level $j \in P_i$ for keyword i corresponds to a bid amount b_{ij} that can be used to generate the cost-revenue pairs (c_{ij}, r_{ij}) . In online advertising, ad

slot allocations are typically determined through a generalized second price auction. Therefore, there is a bid amount $b_{i,|P_i|}$ large enough such that the cost and number of clicks do not increase for bids larger than $b_{i,|P_i|}$. Bidding more than $b_{i,|P_i|}$ will not affect the number of clicks or the cost since the number of clicks depends on the page position and the cost depends on the value of the next highest bid (both of which do not change).

We sampled the estimated cost and the estimated number of clicks over a period of one day for 600 keywords and 20 bid levels using the Google Keyword Planner tool. We picked medium-high volume keywords from a wide variety of industry categories including retail (apparel, footwear, etc.), insurance, and financial services. For each keyword i , we plotted φ_{ij} vs c_{ij} and observed the relationship between φ_{ij} and c_{ij} generally follows a logarithmic function which can be captured as

$$\varphi_{ij} = \beta_{i0} + \beta_{i1} \ln c_{ij}. \tag{2.8}$$

Using the dataset, we estimated β_{i0} and β_{i1} for each keyword i using the ordinary least squares (OLS) method. For instance, in Figure 2.7, we present φ_{ij} vs c_{ij} for keyword “automotive insurance” with 20 bid levels. The relationship between the number of clicks and cost is captured by the function $\varphi_{ij} = \beta_{i0} + \beta_{i1} \ln c_{ij}$ where $\beta_{i0} = -43$ and $\beta_{i1} = 41$ are estimated by the OLS method. For “automotive insurance”, coefficient of determination (R^2) value is 0.99, which indicates a strong fit for the function. In fact, the average coefficient of determination (R^2) value was

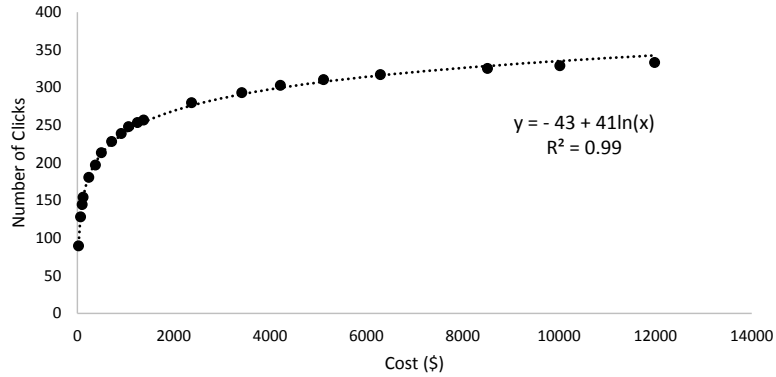


Figure 2.7: The estimated cost vs the estimated number of clicks for keyword “automotive insurance” for 20 bid levels collected from Google Keyword Planner.

0.92 over the 600 keywords we sampled. Therefore we used the logarithmic function (2.8) to generate the number of clicks as a function of the cost.

When we analyzed the distribution of the OLS estimates for β_{i0} and β_{i1} over the 600 keywords in our dataset, we found β_{i0} followed a Cauchy distribution and β_{i1} followed a log-normal distribution with a coefficient of correlation $\rho = -0.82$ (indicating they are highly correlated). We randomly draw correlated β_{i0} (from the Cauchy distribution) and β_{i1} (from the log-normal distribution) values using copulae. Copulae are widely used to draw correlated multivariate random numbers from different distributions. After experimenting with Normal, Student’s t, Clayton, and Gumbel copulae, we observed Gumbel copula provided the best fit for the sample. We therefore used Gumbel copula to draw correlated β_{i0} and β_{i1} values.

To generate an instance for online advertising, we draw β_{i0} and β_{i1} values for each keyword. The number of clicks is then set to $\varphi_{ij} = \beta_{i0} + \beta_{i1} \ln(c_{ij} + \psi_i)$ for $j = 0, 1, \dots, |P_i|$ for a given value of c_{ij} where $\psi_i = e^{-\beta_{i0}/\beta_{i1}}$. The second term ψ_i

inside the logarithm needs some explanation. Since, the number of clicks associated with not bidding (bid level 0) is 0, we need to ensure that the logarithmic function stays non-negative for all values of $c_{ij} \geq 0$. Setting $\psi_i = e^{-\beta_{i0}/\beta_{i1}}$ ensures this. It turns out that due to the correlation between β_{i0} and β_{i1} , ψ_i is usually a very small constant relative to c_{ij} (indicating a very small perturbation to the sampled functions). During data generation, we ensure that ψ_i values are within the range found for the sample of 600 keywords. In other words, if after drawing β_{i0} and β_{i1} values we find that $\psi_i = e^{-\beta_{i0}/\beta_{i1}}$ is outside the range for the sample we discard this draw.

We now discuss how we select the range of values for the bid levels—and thus the costs and number of clicks (which in turn will yield the revenue). We first select the cost associated with the largest bid level. We want to choose this largest bid level at a point where the logarithmic function has relatively flattened out (this ensures that increasing the bid amount beyond $b_{i,|P_i|}$ does not affect the number of clicks significantly). To do so, note the slope of the logarithmic function $\frac{d\varphi_{ij}}{dc_{ij}} = \frac{\beta_{i1}}{c_{ij} + \psi_i}$. Let ε denote the desired value of the slope at $c_{i,|P_i|}$. A simple calculation shows that $c_{i,|P_i|} = \beta_{i1}/\varepsilon - \psi_i$. Once we have the cost associated with the largest bid level, we generate all the other costs c_{ij} as

$$c_{ij} = \frac{j}{|P_i|} \cdot c_{i,|P_i|} \quad \text{for } j = 0, 1, \dots, |P_i| - 1.$$

In our experiments, we set $\varepsilon = 0.1$.

We experiment with three alternatives of RPC_i for the OA instances (the

revenue associated with a bid level is calculated by multiplying the expected number of clicks with the RPC). First, we set $RPC_i = 1$ for all $i \in K$, which we call “Constant OA” instances (this sets the revenue-per-click associated with different keywords to the same value). Second, we set $RPC_i \sim U(1, 100)$ for all $i \in K$ where $x \sim U(a, b)$ indicates x is uniformly distributed between a and b . We call these set of instances “Random OA” instances. Third, we set RPC_i equal to the average cost-per-click (CPC) for keyword i (in the φ_{ij} vs c_{ij} function), which we call “CPC OA” instances⁵. In practice, we would not expect RPC_i to be constant, totally random, or same as average CPC. However, the performance of the CG algorithm for these three extreme cases (with wide variety in behavior) should signal its ability to handle practical instances.

Notice that our logarithmic function does not have any random noise term (i.e., all points generated lie on this logarithmic function). This has a drawback in that all points generated are on the convex hull of the revenue-to-cost curve, thus making the instances harder to solve (as the algorithm cannot eliminate any points due to the fact that they are not on the convex hull).

2.5.2 Results for Online Advertising Instances

We experiment with five different $|K|$ and $|P|$ values, and three different RPC types as stated. For each instance, we experiment with budgets of $B =$

⁵This can be calculate analytically as follows. For keyword i , we define average cost-per-click as the ratio of the average cost to the average number of clicks. Over the range $[0, c_{i,|P_i|}]$, the average cost is calculated as $c_{i,|P_i|}/2$. The average number of clicks is calculated by taking the definite integral $\int_0^{c_{i,|P_i|}} \beta_{i0} + \beta_{i1} \ln(c - \psi_i) dc$ and dividing by $c_{i,|P_i|}$. After evaluating the integral, the average cost-per-click for keyword i is $\frac{c_{i,|P_i|}^2}{2(\beta_{i0}c_{i,|P_i|} + \beta_{i1}((c_{i,|P_i|} - \psi_i) \ln(c_{i,|P_i|} - \psi_i) - c_{i,|P_i|}))}$.

K	P	B = 0.5%		B = 1%		B = 5%		B = 10%		B = 20%	
		CG	DZ	CG	DZ	CG	DZ	CG	DZ	CG	DZ
1M	10	0.16	2.03	0.19	2.03	0.28	1.90	0.45	1.98	0.73	2.34
	20	0.22	2.56	0.23	2.62	0.59	2.87	0.91	3.03	1.58	3.70
	30	0.28	3.20	0.34	3.23	1.03	3.42	1.55	3.67	2.71	4.54
	40	0.38	3.71	0.41	3.65	1.26	4.29	2.64	4.42	4.36	5.74
	50	0.41	3.99	0.45	3.98	1.58	4.48	2.57	5.02	5.02	6.28
	Average	0.29	3.10	0.32	3.10	0.95	3.39	1.62	3.62	2.88	4.52
5M	10	0.87	10.09	0.94	10.11	1.22	10.14	2.28	10.31	3.62	12.24
	20	1.17	12.78	1.26	13.24	3.60	14.23	5.43	15.38	9.19	18.98
	30	1.65	16.37	1.89	16.15	6.12	17.57	9.06	18.63	15.48	23.61
	40	1.87	18.39	2.28	18.64	7.11	21.20	13.45	22.74	25.44	29.03
	50	2.17	19.91	2.51	20.11	8.35	22.92	14.82	24.55	28.50	32.21
	Average	1.55	15.51	1.77	15.65	5.28	17.21	9.01	18.32	16.45	23.21
10M	10	1.72	20.61	1.89	20.14	2.59	20.00	4.88	21.15	7.94	25.01
	20	2.50	26.50	2.65	25.60	7.32	29.17	11.65	30.47	20.65	38.47
	30	3.37	33.04	4.13	33.49	13.03	34.51	19.86	38.05	35.13	48.05
	40	3.95	38.21	4.62	37.74	15.62	42.84	28.84	45.80	55.44	57.33
	50	4.41	40.83	5.20	41.04	17.19	46.63	32.06	49.75	60.53	63.68
	Average	3.19	31.84	3.70	31.60	11.15	34.63	19.46	37.04	35.94	46.51
25M	10	4.65	51.62	4.88	51.76	7.02	52.59	13.46	50.97	23.43	61.74
	20	6.51	63.07	7.00	67.25	20.08	74.19	32.17	80.59	58.56	95.43
	30	8.67	82.38	10.47	84.60	35.44	89.12	54.15	96.08	94.60	119.43
	40	10.05	96.47	11.86	96.81	40.86	106.44	78.72	116.16	150.38	146.06
	50	11.36	104.40	13.43	103.91	45.33	115.47	85.32	124.13	163.55	162.72
	Average	8.25	79.59	9.53	80.87	29.75	87.56	52.76	93.58	98.11	117.08
50M	10	9.66	105.63	10.47	96.72	14.85	105.83	29.38	105.54	50.81	125.47
	20	13.56	134.30	14.65	131.15	42.01	149.23	67.91	159.07	124.63	188.87
	30	17.66	169.76	20.97	169.35	73.80	177.39	113.97	191.76	200.49	238.92
	40	20.51	189.90	24.93	190.43	85.54	211.02	166.75	231.12	317.24	293.84
	50	23.31	204.17	27.39	206.37	92.10	225.12	180.07	249.94	342.51	324.33
	Average	16.94	160.75	19.68	158.81	61.66	173.72	111.62	187.48	207.14	234.28

Table 2.1: CPU times (in seconds) for Constant OA instances

$\{0.5\%, 1\%, 5\%, 10\%, 20\%\}$ of $\sum_{i \in K} \max_{j \in P_i} c_{ij}$. In other words, $B = 20\%$ indicates the advertising budget is set to 20% of the maximum amount the advertiser can spend on the portfolio. In total, there are $5 \times 5 \times 5 \times 3 = 375$ combinations of number of keywords, number of bid levels, budget, and RPC type. We provide the CPU times of both the CG and DZ algorithms for all 375 instances in Tables 2.1 - 2.3. However, for ease of reference and discussion, we provide summary results for both algorithms in Figures 2.8 - 2.10. For each chart, y-axis provides the average CPU time per 10 million keyword levels in seconds while x-axis compares number of keywords ($|K|$), number of bid levels ($|P|$), and budget percentages (B). We discuss these results in terms of average CPU time per 10 million keyword levels

$ K $	$ P $	$B = 0.5\%$		$B = 1\%$		$B = 5\%$		$B = 10\%$		$B = 20\%$	
		CG	DZ	CG	DZ	CG	DZ	CG	DZ	CG	DZ
1M	10	0.16	1.76	0.19	1.81	0.27	1.79	0.48	1.94	0.84	2.32
	20	0.19	2.42	0.22	2.47	0.59	2.76	1.11	2.86	2.18	3.46
	30	0.28	3.01	0.36	2.98	1.11	3.37	1.95	3.53	3.58	4.39
	40	0.33	3.59	0.41	3.49	1.48	4.20	3.01	4.27	5.66	5.29
	50	0.39	3.74	0.47	3.78	1.67	4.42	3.18	4.62	6.22	5.94
	Average	0.27	2.90	0.33	2.90	1.02	3.31	1.95	3.44	3.69	4.28
5M	10	0.72	9.41	0.75	9.47	1.28	9.38	2.33	10.14	5.16	12.39
	20	1.05	12.73	1.26	12.68	3.53	14.10	7.25	14.24	14.91	18.02
	30	1.51	15.82	1.98	15.48	7.13	17.86	12.93	18.41	24.57	22.39
	40	1.93	17.94	2.32	18.07	9.41	20.70	18.78	22.07	36.99	27.30
	50	2.18	19.42	2.70	19.42	10.05	22.57	20.92	23.95	39.44	30.20
	Average	1.48	15.06	1.80	15.02	6.28	16.92	12.44	17.76	24.21	22.06
10M	10	1.45	18.81	1.56	19.05	2.89	18.92	5.37	20.05	11.51	23.56
	20	2.26	24.87	2.53	25.65	7.97	28.80	16.16	30.23	33.51	35.99
	30	3.18	31.68	4.18	31.37	15.30	35.63	28.95	36.77	53.70	44.68
	40	3.93	35.79	4.98	36.26	20.55	41.98	40.76	45.07	81.17	55.15
	50	4.32	38.52	5.62	39.09	22.15	45.60	43.31	47.97	85.64	60.56
	Average	3.03	29.93	3.77	30.28	13.77	34.19	26.91	36.02	53.11	43.99
25M	10	3.90	48.02	4.20	47.10	7.69	48.05	14.96	50.23	32.81	60.86
	20	5.73	64.65	6.72	65.35	21.37	72.09	44.41	75.47	93.10	90.67
	30	8.61	79.45	11.22	78.97	41.92	88.83	78.36	92.80	148.22	112.07
	40	9.73	91.64	13.51	88.75	55.38	106.67	112.94	111.79	223.31	137.98
	50	11.31	98.28	14.56	95.38	59.22	114.13	118.83	119.39	237.81	152.79
	Average	7.86	76.41	10.04	75.11	37.12	85.95	73.90	89.94	147.05	110.87
50M	10	7.97	94.21	8.81	96.22	16.21	96.80	31.29	102.96	71.35	124.68
	20	11.90	128.54	13.98	127.19	44.93	146.05	95.52	150.90	202.58	177.56
	30	16.97	158.93	22.87	160.84	88.39	180.49	165.78	181.13	320.67	222.41
	40	19.84	183.67	26.60	183.66	116.89	210.93	239.30	226.48	480.26	277.56
	50	22.40	197.40	29.33	197.03	124.05	225.80	251.36	241.47	507.10	308.18
	Average	15.82	152.55	20.32	152.99	78.09	172.01	156.65	180.59	316.39	222.08

Table 2.2: CPU times (in seconds) for Random OA instances

since the theoretical running time of the DZ algorithm is $O(n)$ where $n = \sum_{i \in K} |P_i|$. Note that in all charts, the CPU times are averaged over the omitted dimensions. For instance, in Figure 2.8, the CPU times are averaged over different number of bid levels and budget percentages.

Across different number of keywords, number of bid levels, and budget percentages, we observe the CG algorithm significantly outperforms the DZ algorithm and the RPC type does not significantly affect the running time of either algorithm. However, it is worth noting that Random OA and CPC OA instances take slightly longer to solve for the CG algorithm. In Figure 2.8, we observe the CPU time grows linearly for the DZ algorithm as expected from its theoretically linear running time.

$ K $	$ P $	$B = 0.5\%$		$B = 1\%$		$B = 5\%$		$B = 10\%$		$B = 20\%$	
		CG	DZ	CG	DZ	CG	DZ	CG	DZ	CG	DZ
1M	10	0.17	1.90	0.19	1.89	0.20	1.90	0.51	1.81	0.76	2.32
	20	0.20	2.50	0.23	2.50	0.59	2.76	1.09	2.89	2.06	3.64
	30	0.28	3.14	0.33	3.12	1.11	3.39	1.90	3.51	3.37	4.45
	40	0.33	3.68	0.39	3.60	1.37	4.04	2.73	4.31	5.27	5.43
	50	0.41	3.89	0.45	3.98	1.54	4.35	3.14	4.60	5.83	6.08
	Average	0.28	3.02	0.32	3.02	0.96	3.29	1.88	3.42	3.46	4.38
5M	10	0.78	10.11	0.81	9.98	1.15	10.08	2.29	9.39	4.32	12.25
	20	1.12	12.92	1.23	12.78	3.45	14.07	6.71	14.84	13.79	18.27
	30	1.53	16.21	1.90	15.94	7.01	17.46	11.90	18.17	22.89	22.62
	40	1.84	18.72	2.15	18.13	8.85	20.61	17.39	21.89	34.93	27.99
	50	2.14	20.08	2.45	19.61	9.53	22.20	18.31	23.49	37.41	30.30
	Average	1.48	15.61	1.71	15.29	6.00	16.88	11.32	17.56	22.67	22.28
10M	10	1.65	20.37	1.72	19.92	2.42	19.94	4.73	18.22	9.59	23.57
	20	2.40	25.74	2.67	25.99	7.41	28.41	14.45	29.67	30.36	36.30
	30	3.32	32.70	3.87	33.12	15.26	34.06	26.54	35.57	50.39	45.58
	40	3.79	37.19	4.56	37.32	18.89	40.69	36.88	44.44	76.22	56.11
	50	4.37	40.33	5.32	40.33	20.11	44.06	39.27	47.27	80.86	62.06
	Average	3.11	31.27	3.63	31.33	12.82	33.43	24.37	35.03	49.48	44.73
25M	10	4.52	49.27	4.84	49.90	6.88	51.04	13.49	49.14	27.52	60.97
	20	6.38	66.32	7.04	64.96	20.58	72.45	40.44	75.33	84.83	92.02
	30	8.67	80.92	10.55	82.15	42.09	86.39	72.91	91.71	137.34	112.13
	40	10.14	96.16	12.37	94.65	50.84	101.95	103.37	110.32	211.82	140.38
	50	11.50	101.49	13.93	102.06	53.98	109.22	108.25	119.36	226.22	154.64
	Average	8.24	78.83	9.74	78.74	34.87	84.21	67.69	89.17	137.55	112.03
50M	10	9.72	103.49	10.34	102.12	14.32	101.98	28.69	95.75	60.45	124.66
	20	13.31	132.41	14.68	132.80	43.06	141.88	85.96	151.04	184.72	187.67
	30	17.96	166.14	21.33	165.77	88.59	170.60	155.80	182.86	299.43	231.21
	40	20.84	189.01	24.59	190.13	106.66	209.26	217.37	225.73	455.97	281.91
	50	23.85	205.00	28.21	203.32	114.08	223.91	231.18	239.12	475.72	309.82
	Average	17.13	159.21	19.83	158.83	73.34	169.53	143.80	178.90	295.26	227.05

Table 2.3: CPU times (in seconds) for CPC OA instances

However, averaging across other dimensions, the CG algorithm also shows near linear growth as the number of keywords increases. The slight upward trend can be explained by the $O(|K| \log |K|)$ term in the theoretical running time of the CG algorithm. Recall that we use $\kappa = 100$ for both $|K| = 1\text{M}$ and $|K| = 50\text{M}$. When $|K| = 50\text{M}$, each bucket, on average, will contain 50 times the number of convex hull levels compared to when $|K| = 1\text{M}$. Therefore, sorting each bucket will take longer when $|K| = 50\text{M}$. A good choice for the number of buckets (κ) parameter would potentially mitigate the growth as $|K|$ gets larger. In any case, a near linear growth on the running time for the CG algorithm demonstrates the ability to scale up and handle even larger instances.

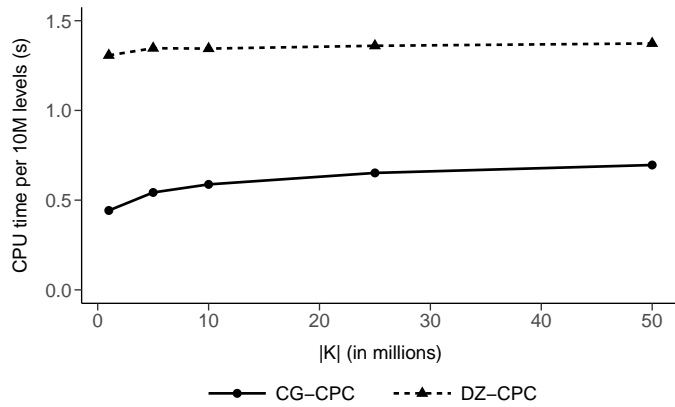
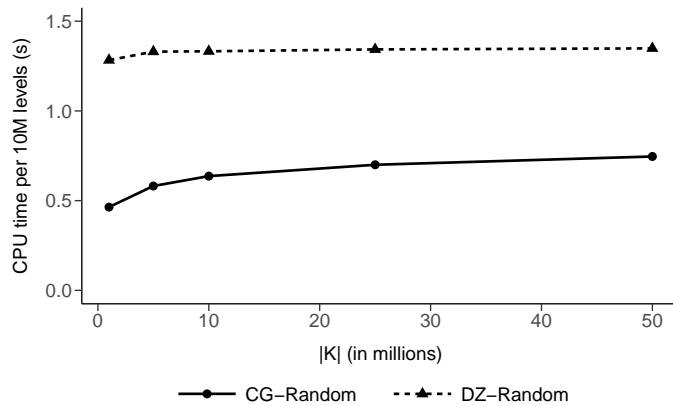
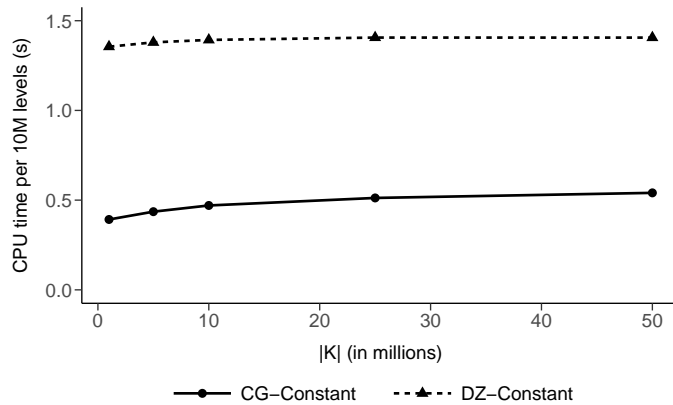


Figure 2.8: Average CPU time per 1 million keywords vs number of keywords for Constant, Random, and CPC OA instances

In Figure 2.9, the CPU time has a near linear increase for the CG algorithm whereas for the DZ algorithm, the CPU time decreases as the number of bid levels increases. For instances with larger bid levels, the computational overhead is distributed over a larger number of keyword levels (recall that we report the CPU time per 10 million keyword levels) resulting in a decrease in the CPU time of the DZ algorithm. However, the decrease starts leveling out at 40 bid levels and we expect it to be leveled out for larger number of bid levels. On the other hand, the increase in the number of bid levels has a near linear effect on the CPU time for the CG algorithm despite the nonlinear term for generating bid levels in its theoretical running time. In practice, the advertisers do not estimate large number of bid levels for a keyword due to the cost associated with estimating each bid level. Therefore, for practical purposes (where the number bid levels is typically less than 20), the CG algorithm provides significant computational improvement over the DZ algorithm.

In Figure 2.10, we can clearly observe the effect of budget increase on the CPU time of the CG algorithm. This is inevitable since many nonlinear terms in the theoretical running time of the CG algorithm are functions of the budget. On the other hand, despite its theoretically linear running time, the budget percentages over 1% results in an increase on the CPU time of the DZ algorithm. After a closer examination of the algorithm and the data set, we observe that the log-curvature of the cost-revenue function causes the DZ algorithm to remove less levels per iteration for larger budget percentages, which in turn increases the CPU time. For budget percentage 0.05%, the CG algorithm is about 10.5 times faster than the DZ algorithm across three RPC types. For budget percentage 5%, the CG

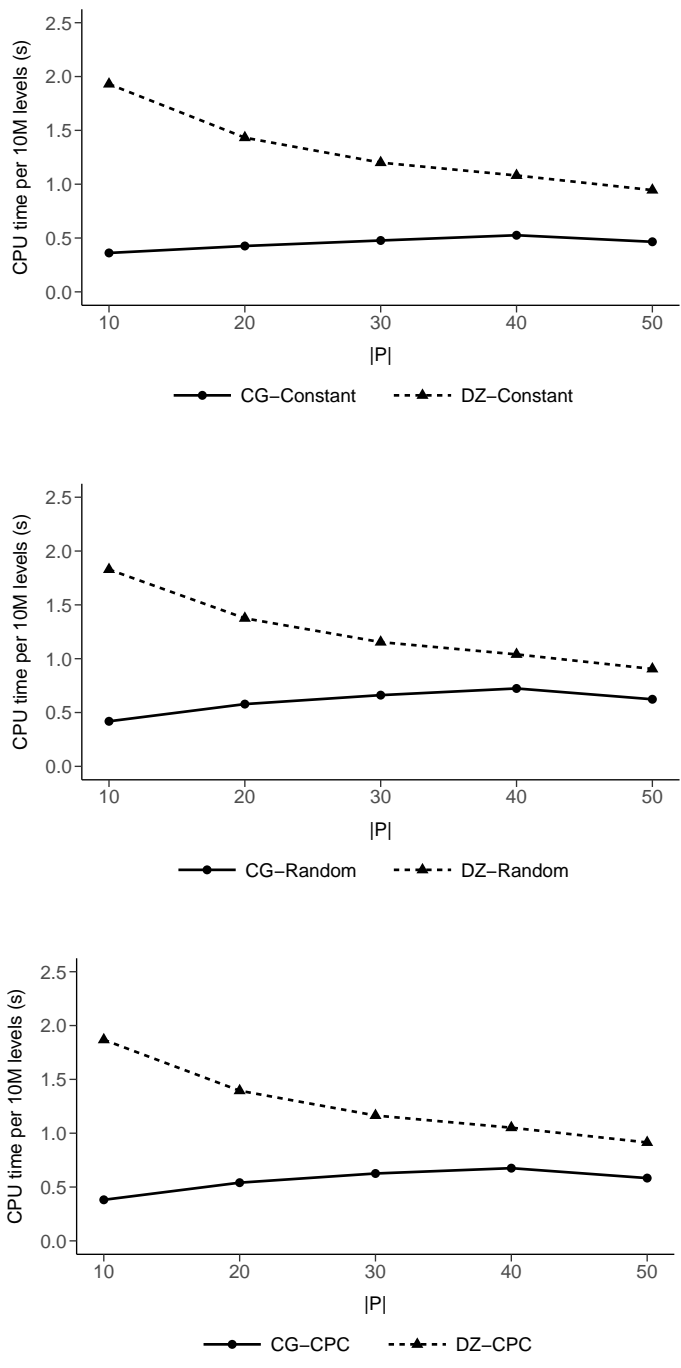


Figure 2.9: Average CPU time per 1 million keywords vs number of levels for Constant, Random, and CPC OA instances

algorithm is still about 3.3 times faster than the DZ algorithm. However, when the budget percentage is around 20%, the CG algorithm finally catches up with the DZ algorithm for Random and CPC instances. Although, it is important to note that in online advertising, the daily budget is typically set to a small amount compared to the maximum amount possible to spend. Therefore, for the purposes of online advertising, the CG algorithm would be preferable to the DZ algorithm.

K	P	B = 0.1%		B = 0.5%		B = 1%		B = 2.5%		B = 5%	
		CG	DZ	CG	DZ	CG	DZ	CG	DZ	CG	DZ
1M	10	0.19	1.86	0.28	1.90	0.33	1.86	0.45	1.97	0.50	1.92
	20	0.31	2.40	0.48	2.32	0.58	2.47	0.81	2.37	1.03	2.48
	30	0.44	2.84	0.77	2.87	0.98	2.81	1.34	2.86	1.69	2.79
	40	0.64	3.15	1.11	3.14	1.51	3.21	1.97	3.25	2.39	3.20
	50	0.84	3.63	1.54	3.54	1.98	3.57	2.70	3.51	3.18	3.56
	Average	0.48	2.78	0.84	2.76	1.08	2.78	1.45	2.79	1.76	2.79
5M	10	0.92	9.64	1.33	9.31	1.56	9.45	2.18	9.63	2.65	9.53
	20	1.51	12.36	2.51	12.48	3.21	12.48	4.37	12.81	5.45	12.39
	30	2.34	14.34	4.06	14.46	5.18	14.82	7.24	14.38	9.03	14.62
	40	3.25	16.12	5.96	15.79	7.69	16.19	10.33	16.66	12.70	16.30
	50	4.26	17.83	7.91	18.33	10.33	17.89	13.82	17.96	16.57	18.02
	Average	2.46	14.06	4.35	14.07	5.59	14.17	7.59	14.29	9.28	14.17
10M	10	1.76	19.28	2.62	19.59	3.23	19.38	4.51	19.02	5.49	18.88
	20	3.11	25.13	5.10	24.85	6.44	25.01	8.89	25.07	11.06	24.98
	30	4.79	28.72	8.25	29.41	10.73	28.80	14.81	29.34	18.55	29.52
	40	6.55	32.78	12.00	32.87	15.49	32.59	21.00	32.75	26.29	33.51
	50	8.58	36.10	16.12	36.04	20.97	36.50	28.13	35.44	33.67	35.24
	Average	4.96	28.40	8.82	28.55	11.37	28.45	15.47	28.32	19.01	28.42
25M	10	4.68	49.64	6.99	49.41	8.77	49.80	11.95	49.67	14.74	49.72
	20	7.99	62.00	13.03	62.53	16.63	62.76	23.03	61.12	29.25	63.13
	30	12.06	73.57	20.92	73.93	27.41	73.16	37.78	72.53	48.27	74.72
	40	16.61	82.04	29.95	82.26	38.99	81.90	53.91	83.27	67.38	82.24
	50	21.57	91.42	40.51	91.10	52.57	92.13	71.48	91.07	85.72	91.70
	Average	12.58	71.73	22.28	71.84	28.87	71.95	39.63	71.53	49.07	72.30
50M	10	9.42	93.49	14.15	98.59	17.58	99.59	24.01	100.22	30.55	100.25
	20	16.21	125.28	26.68	127.91	34.06	126.81	46.50	127.98	59.98	129.46
	30	24.06	144.00	42.40	147.09	54.54	148.81	76.03	146.73	98.47	148.61
	40	33.13	160.65	60.39	167.87	78.44	167.78	108.59	167.48	136.10	165.89
	50	43.46	183.80	81.06	181.38	105.50	182.26	144.61	181.30	174.67	183.16
	Average	25.26	141.45	44.93	144.57	58.02	145.05	79.95	144.74	99.95	145.47

Table 2.4: CPU times (in seconds) for UC instances

2.5.3 Results for Literature Instances

The most common approaches in the literature to generate MCKP instances are called uncorrelated (UC), weakly correlated (WC), and strongly correlated (SC)

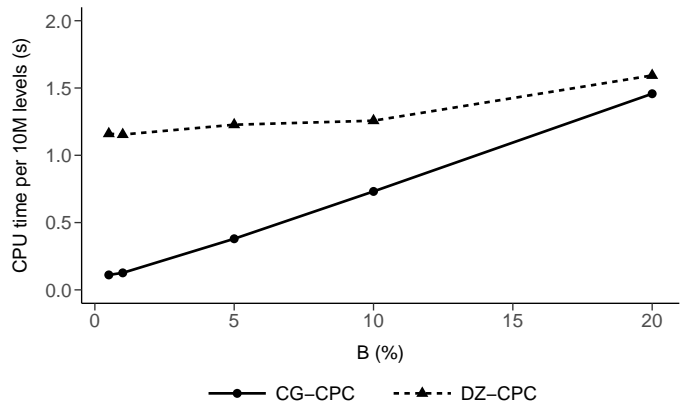
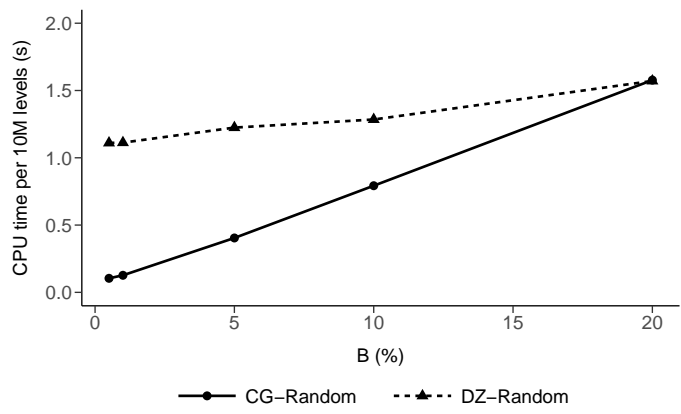
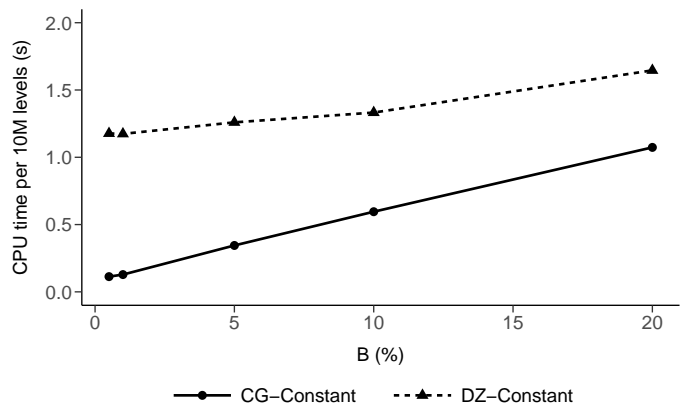


Figure 2.10: Average CPU time per 1 million keywords vs budget percentage for Constant, Random, and CPC OA instances

K	P	B = 0.1%		B = 0.5%		B = 1%		B = 2.5%		B = 5%	
		CG	DZ	CG	DZ	CG	DZ	CG	DZ	CG	DZ
1M	10	0.17	2.15	0.23	2.12	0.27	2.11	0.34	2.12	0.42	2.04
	20	0.25	2.96	0.37	2.92	0.51	2.95	0.66	2.96	0.84	2.83
	30	0.41	3.39	0.62	3.34	0.81	3.42	1.14	3.42	1.39	3.31
	40	0.53	3.84	0.89	3.88	1.20	3.87	1.70	3.82	2.02	3.95
	50	0.69	4.29	1.23	4.35	1.56	4.26	2.20	4.12	2.77	4.23
	Average	0.41	3.33	0.67	3.32	0.87	3.32	1.21	3.29	1.49	3.27
5M	10	0.80	10.80	1.08	11.08	1.29	10.83	1.69	10.94	2.20	10.87
	20	1.34	14.66	2.01	14.01	2.56	14.73	3.45	14.82	4.29	14.79
	30	2.00	17.49	3.23	17.53	4.21	17.66	5.68	17.47	7.18	17.47
	40	2.81	19.44	4.65	19.78	6.02	19.75	8.30	19.44	10.44	19.23
	50	3.53	21.70	6.27	22.01	8.30	21.54	11.28	21.67	13.95	21.70
	Average	2.09	16.82	3.45	16.88	4.48	16.90	6.08	16.87	7.61	16.81
10M	10	1.64	22.42	2.29	21.89	2.78	22.07	3.56	20.61	4.38	21.48
	20	2.71	29.59	4.23	29.34	5.20	29.86	7.04	29.69	8.99	30.16
	30	4.13	35.52	6.80	35.60	8.64	35.72	11.70	35.66	14.88	34.77
	40	5.60	39.28	9.58	39.83	12.42	39.39	17.07	39.83	21.28	39.39
	50	7.24	43.43	12.68	42.78	16.61	43.60	22.89	43.73	28.55	43.38
	Average	4.26	34.05	7.12	33.89	9.13	34.13	12.45	33.90	15.62	33.84
25M	10	4.29	56.44	5.99	55.75	7.18	55.47	9.45	56.44	11.95	54.52
	20	6.91	75.24	10.66	75.79	13.21	73.90	18.35	75.75	23.14	75.11
	30	10.45	88.48	16.86	88.03	21.53	89.00	30.11	89.01	37.99	86.35
	40	13.90	100.34	23.99	99.50	30.86	99.29	43.12	99.65	54.43	97.58
	50	17.99	109.54	32.14	109.14	41.59	109.33	57.95	110.07	72.43	108.92
	Average	10.71	86.01	17.93	85.64	22.87	85.40	31.80	86.19	39.99	84.50
50M	10	8.58	113.66	12.04	112.94	14.54	109.92	19.31	111.23	24.40	112.21
	20	13.99	151.94	21.42	150.65	27.19	151.60	37.05	150.31	47.27	151.93
	30	20.69	179.49	34.31	179.03	43.42	180.27	60.79	179.00	77.11	178.67
	40	28.30	197.62	48.42	200.69	62.34	198.98	86.85	198.48	110.14	199.87
	50	36.38	220.32	64.33	220.23	84.04	220.21	117.03	220.69	146.64	217.54
	Average	21.59	172.61	36.10	172.71	46.30	172.20	64.21	171.94	81.11	172.04

Table 2.5: CPU times (in seconds) for WC instances

instances. Since the CG algorithm applies to the MCKP-LP, we also test it on these instances (we note these instances are quite different in structure from the OAPOP instances). We adapt the methods described in [Pisinger \[1995\]](#) to generate large-scale UC, WC, and SC instances (the instances we generate are orders of magnitude larger than those available in the literature) with the settings below. We refer to these as “literature instances”.

UC instances. The uncorrelated instances are generated by setting $c_{ij} \sim U(1, 1000)$ and $r_{ij} \sim U(1, 1000)$ for all $i \in K$ and $j \in P_i \setminus \{0\}$.

WC instances. The weakly correlated instances are generated by setting $c_{ij} \sim U(1, 1000)$ and $r_{ij} \sim U(c_{ij} - 10, c_{ij} + 10)$ for all $i \in K$ and $j \in P_i \setminus \{0\}$.

SC instances. For the strongly correlated instances, we first generate $c'_{ij} \sim U(1, 1000)$ and $r'_{ij} = c'_{ij} + 10$ for a given keyword $i \in K$ and $j \in P_i \setminus \{0\}$. Then we sort c'_{ij} in nonincreasing and set $c_{ij} = \sum_{k=1}^j c'_{ik}$ and $r_{ij} = \sum_{k=1}^j r'_{ik}$ for all $j \in P_i \setminus \{0\}$. Note that for strongly correlated instances, $\Omega_i = P_i$, i.e. every level is in the convex hull.

In literature instances, we have 375 combinations of number of keywords, number of bid levels, budget, and correlation type. We experiment with $B = \{0.1\%, 0.5\%, 1\%, 2.5\%, 5\%\}$ of $\sum_{i \in K} \max_{j \in P_i} c_{ij}$. We discuss our findings in figures as in the online advertising instances. However, the detailed tables containing the results of the literature instances (that were used to generate the figures) are provided in Tables 2.4-2.6. In Figures 2.11 - 2.13, we discuss the performance of the CG algorithm vs the DZ algorithm for UC, WC, SC instances. For all three types of instances, the CG algorithm outperforms the DZ algorithm. In Figure 2.11, we observe that SC instances are difficult for both algorithms and take the longest time to solve. The CG algorithm performs a little faster for WC instances compared to UC instances while the opposite is true for the DZ algorithm. For UC and WC instances, the CPU time for the CG algorithm grows linearly. However, for SC instances, the CPU time shows a slight nonlinear growth. We assert the convex hull structure (i.e., number of levels in the convex hull) has more of an effect on the CPU time for both algorithms compared to the revenue structure. In OA instances, different revenue types do not affect the CPU time as significantly for either algorithm. On the other hand, since all levels are in the convex hull in SC instances and some levels may not be in the convex hull in UC and WC instances, CPU times

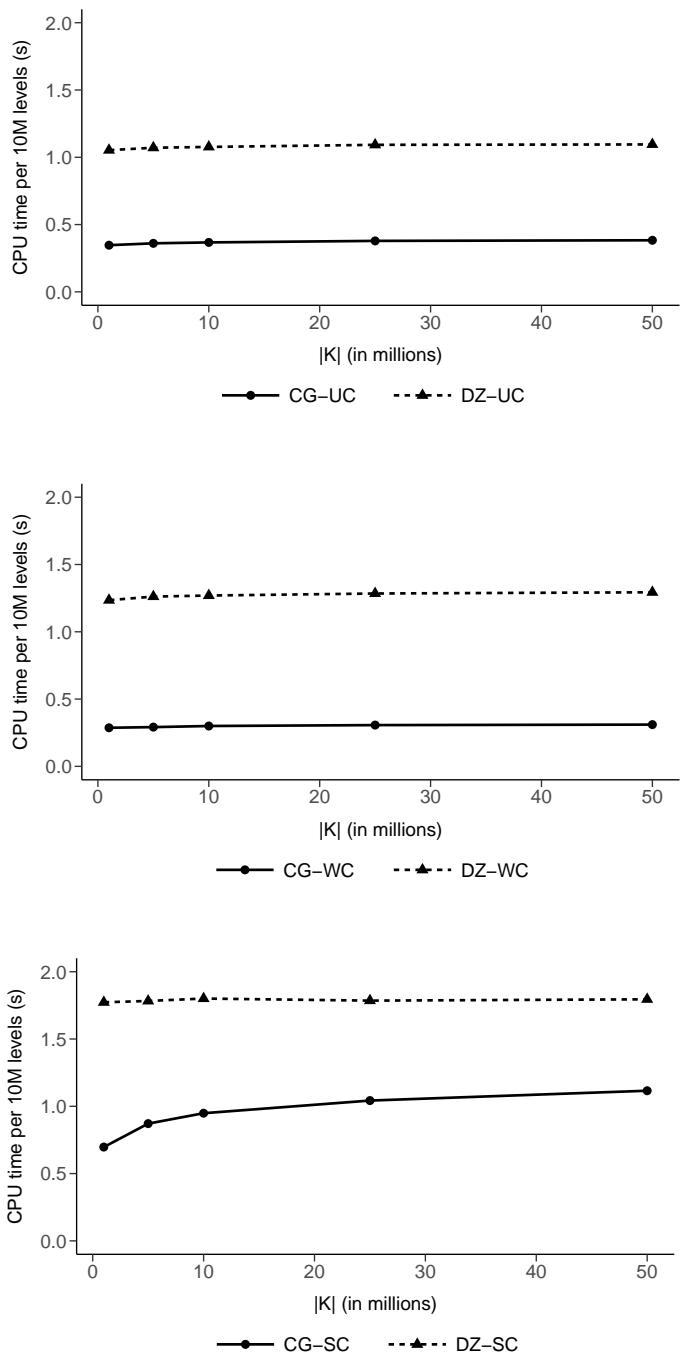


Figure 2.11: Average CPU time per 1 million keywords vs number of keywords for UC, WC, and SC instances

K	P	B = 0.1%		B = 0.5%		B = 1%		B = 2.5%		B = 5%	
		CG	DZ	CG	DZ	CG	DZ	CG	DZ	CG	DZ
1M	10	0.14	2.56	0.31	2.53	0.38	2.50	0.69	2.53	1.05	2.47
	20	0.37	4.12	0.81	4.01	1.12	4.07	1.81	4.03	2.57	4.12
	30	0.84	5.40	1.51	5.23	2.04	5.26	3.14	5.24	4.12	5.13
	40	1.03	6.52	2.23	6.55	3.28	6.47	4.70	6.35	6.61	6.46
	50	1.11	7.07	2.33	6.99	3.49	7.02	5.26	6.93	7.19	6.90
	Average	0.70	5.13	1.44	5.06	2.06	5.06	3.12	5.01	4.31	5.01
5M	10	0.72	12.67	1.45	12.56	2.09	12.75	3.99	12.76	5.99	12.53
	20	2.14	20.55	5.13	20.53	7.46	20.76	12.04	20.25	17.07	20.48
	30	5.10	26.63	9.67	26.26	13.20	26.40	20.02	25.88	27.75	25.66
	40	6.22	32.96	13.53	32.42	19.08	32.85	30.23	32.37	42.87	32.10
	50	6.68	35.58	14.54	35.57	20.48	35.37	32.35	34.95	45.01	34.99
	Average	4.17	25.68	8.86	25.47	12.46	25.62	19.73	25.24	27.74	25.15
10M	10	1.50	25.99	3.15	25.99	4.45	25.83	8.55	25.72	13.26	25.44
	20	4.66	41.87	11.22	41.43	16.46	40.97	26.85	40.73	38.27	41.31
	30	10.61	54.15	21.19	52.60	29.16	52.67	43.77	52.29	61.17	52.35
	40	13.04	66.13	29.09	67.17	41.61	65.61	65.54	65.44	92.01	64.58
	50	14.16	71.62	31.40	71.17	43.74	71.23	69.61	70.12	97.48	68.95
	Average	8.80	51.95	19.21	51.67	27.08	51.26	42.86	50.86	60.44	50.53
25M	10	4.07	65.91	8.81	63.73	12.51	61.11	24.26	63.29	37.24	63.17
	20	12.15	103.18	30.87	101.45	45.01	103.40	74.02	101.45	106.58	101.43
	30	28.66	135.13	57.63	131.71	78.84	132.46	121.76	130.00	168.31	127.27
	40	34.90	166.66	78.75	161.71	112.62	166.05	180.26	162.08	252.88	161.98
	50	37.30	177.87	84.01	176.87	119.76	177.67	190.77	174.71	266.93	173.61
	Average	23.42	129.75	52.01	127.09	73.75	128.14	118.21	126.30	166.39	125.49
50M	10	8.58	129.73	18.11	130.15	26.71	129.98	52.46	126.13	80.67	129.08
	20	24.66	208.81	64.23	207.17	95.72	202.55	159.84	204.56	231.32	203.07
	30	59.28	269.90	122.80	265.70	169.14	263.20	260.16	258.24	363.23	254.22
	40	72.32	332.30	168.48	332.59	240.90	329.47	385.20	325.90	543.41	324.25
	50	78.28	356.29	179.73	351.69	255.09	356.26	405.43	346.96	572.13	345.79
	Average	48.63	259.40	110.67	257.46	157.51	256.29	252.62	252.36	358.15	251.28

Table 2.6: CPU times (in seconds) for SC instances

differ significantly across UC, WC, and SC instances for both algorithms.

In Figure 2.12, we observe similar trends (compared to the OA instances) for both algorithms. In Figure 2.13, we observe the CG algorithm takes longer on SC instances as budget percentage gets larger whereas the growth in CPU time is not as aggressive in UC and WC instances. For the DZ algorithm the difference between UC, WC, and SC instances persists for any budget percentage and unlike in the OA instances, the algorithm keeps growing linearly as budget percentage increases. This cements our previous observation that the log-curvature of the cost-revenue function causes the DZ algorithm to have different CPU times for different budget percentages.

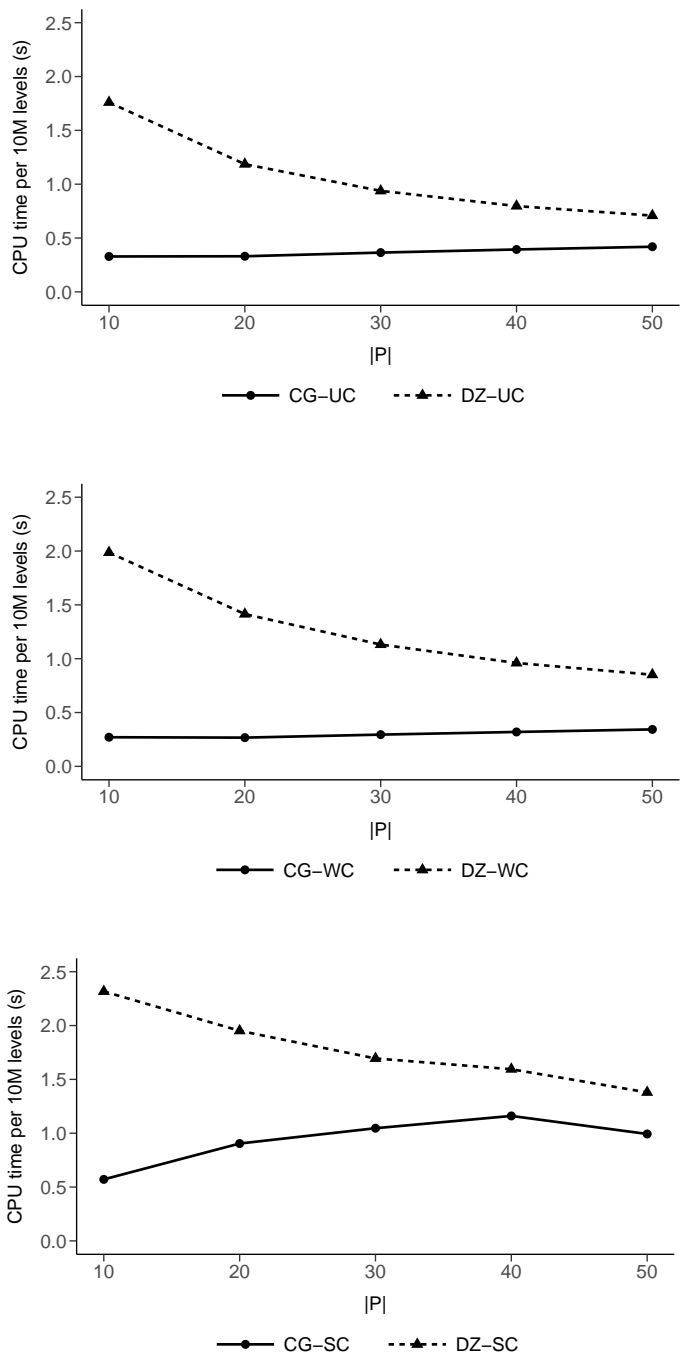


Figure 2.12: Average CPU time per 1 million keywords vs number of levels for UC, WC, and SC instances

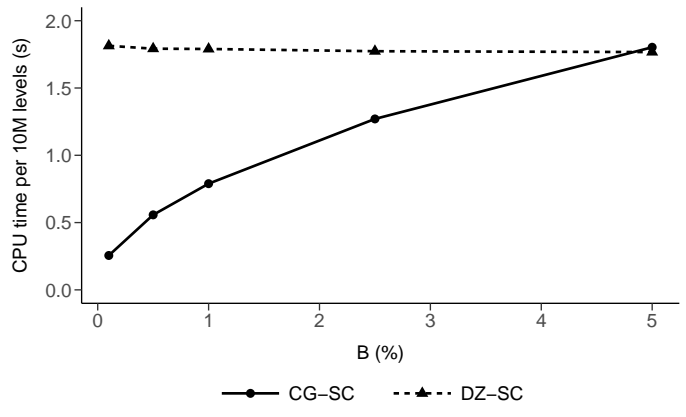
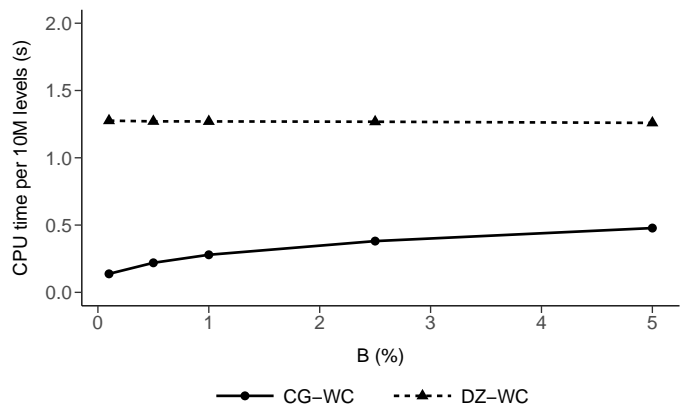
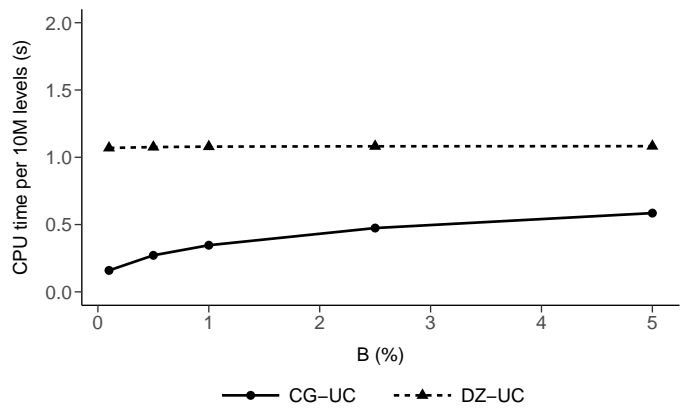


Figure 2.13: Average CPU time per 1 million keywords vs budget percentage for UC, WC, and SC instances

For budget percentage 0.1%, the CG algorithm is about 6 times faster than the DZ algorithm in SC instances. However, for budget percentage 5%, the CG algorithm takes slightly more time to solve than the DZ algorithm. For UC and WC instances, the CG algorithms is still faster than the DZ algorithm at 5% budget, however as the budget percentage further increases, we expect the DZ algorithm to outperform the CG algorithm due to its linear running time. The contrast between the algorithms under different budget percentages is by design. The CG algorithm only generates bid levels as needed while the DZ algorithm removes bid levels that it asserts cannot be in the optimal solution. In essence, the CG algorithm generates columns whereas the DZ algorithm removes columns. In fact, the DZ algorithm usually has to remove the majority of the levels from every keyword before it can find the optimal solution. The CG algorithm only generates levels as long as there is some budget remaining, therefore, the number of bid levels generated depends on the budget. And for a low budget, the CG algorithm would be preferable to the DZ algorithm.

Chapter 3: Targeted Online Advertising with Bid Adjustments

The year 2015 marked the twentieth anniversary of online advertising. No other major advertising medium (radio, television, and cable) achieved the growth online advertising had in its first twenty years [PwC Report, 2016]. In 2016, mobile ad revenues accounted for 50.5% of the total ad revenue, surpassing desktop for the first time [PwC Report, 2017]. Following a similar trend, social media advertising grew 54% on average each year between 2012 and 2016, and was responsible for 22.5% of the total ad revenue in 2016. With new ad delivery methods, platforms, and formats on the rise, the advertisers are now able to collect user characteristic data at an unprecedented level of granularity and harvest the data to gain insights on these characteristics. For instance, an advertiser may notice a trend where females between 25-35 years of age living in large cities and browsing on mobile devices create more return (e.g., revenue) per dollar spent on advertising. In this scenario, targeting this specific user characteristic might increase the return on investment. However, it was not possible to target user characteristics at this level of granularity in most advertising platforms and formats. Advertisers have devised ad-hoc methods (e.g., making copies of ad campaigns with different user characteristics) to be able to target user characteristics, albeit with limited effectiveness. In addition, these

ad-hoc methods have significantly increased the overhead of managing advertising portfolios. In response to the clear need for an updated bidding language, Google, in early 2013, introduced¹ “Enhanced Campaigns”. One of the biggest changes in the bidding language was the introduction² of “bid adjustments” essentially allowing advertisers to target user characteristics by modifying bids based on ad query features. In the current bidding language, these features include geographical location, time of day, device, and audience. On the one hand, bid adjustments create opportunities for advertisers to target desired user characteristics. On the other hand, a more sophisticated bidding language significantly increases the complexity of managing an advertising portfolio. In this chapter, we introduce the Bid Adjustment in Online Advertising (BAPOA), where the return from an advertising portfolio is maximized subject to an advertising budget in the presence of bid adjustments.

The introduction of bid adjustments altered the landscape of online advertising. A few months after Google, Bing adopted³ bid adjustments in their bidding language. The rules for bid adjustments slightly vary between Google and Bing, and search and display formats. Adjustments on location, time, and device can be made on both Google and Bing, and both search and display formats. However, audience adjustments vary based on platform and format. For instance, Google allows adjustments to be made on “age” and “gender” in both search and display formats whereas adjustments based on “parental status” can only be made in display format⁴. In addition, the interactions between bid adjustments and the calculation of an effec-

¹<https://adwords.googleblog.com/2013/02/introducing-enhanced-campaigns.html>

²<https://support.google.com/adwords/answer/2732132?hl=en>

³<https://help.bingads.microsoft.com/apex/index/3/en-us/51004#!>

⁴<https://support.google.com/adwords/answer/2580282?hl=en>

tive bid depend on where bid adjustments apply in the hierarchy of campaigns and ad groups. In both Google and Bing, a campaign is defined as a collection of ad groups and an ad group is defined as a collection of keywords (As in Chapter 2, we refer to “targeting items” as “keywords” for brevity, however the setting and the solution methods discussed in this chapter directly apply to any type of targeting item (e.g., cookies, websites, etc.) operating under the same bidding language.). Each keyword in an ad group needs to have a separate bid, which we refer to as the “base bid”. However, bid adjustments can only be made at the campaign or the ad group level depending on the type of feature and platform. For example, in Google search format, location adjustments can only be made at the campaign level, e.g., an adjustment for New York can be made for a campaign and the adjustment would apply to every keyword in that campaign regardless of the ad group of the keyword. On the other hand, device adjustments can be made at the ad group level. Operating under the new bidding language, the advertiser needs to determine a base bid for each keyword and a bid adjustment for each feature item (a feature item is a member of the feature set, e.g., New York is a feature item for the location feature) for each campaign (or ad group if adjustments can be made at the ad group level). Figure 3.1 schematically represents the hierarchy of a hypothetical Google advertising portfolio.

Under the new bidding language, the bid adjustments interact with each other and with the base bids in a multiplicative manner. Suppose an advertiser identifies users from District of Columbia (DC) on mobile devices browsing between 7-8 pm as a desired user characteristic. Then by increasing the bid for DC between 7-8

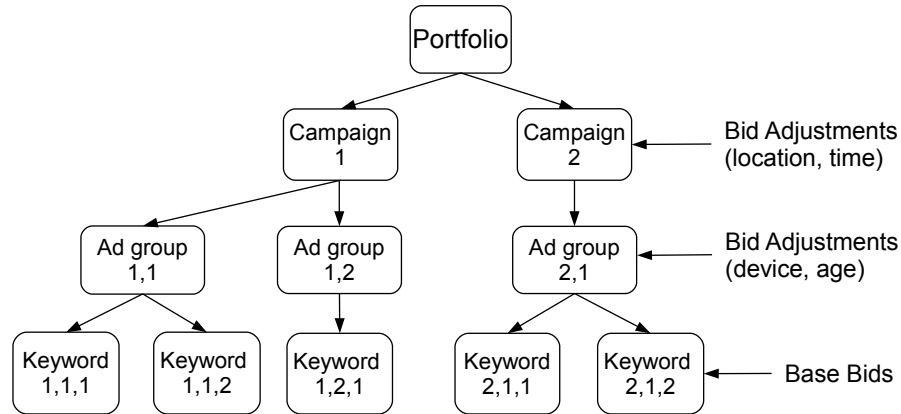


Figure 3.1: The hierarchy of an advertising portfolio with two ad campaigns, three ad groups and five keywords. There are four features for which the advertiser sets bid adjustments. Location and time adjustments can be made at the campaign level. Device and age adjustments can be made at the ad group level.

pm on mobile devices, the advertiser can make their ad appear at a higher position on the page thus better target this user characteristic. However, the rules prohibit adjustments to be made for the specific combination of user characteristics, i.e., feature combinations (e.g., DC, 7-8pm, mobile). Rather, the advertiser submits a base bid for each keyword and bid adjustments for each feature item for each campaign (assuming adjustments can be made at the campaign level for these features). Then the individual adjustments are compounded for each feature combination leading to an effective bid. For example, suppose the advertiser determines a base bid of \$1 for a targeting item (e.g., “running shoes”), and sets the bid adjustments for the campaign (containing “running shoes”) as 2 for DC, 0.8 for 7-8 pm, and 1.5 for mobile devices. Then the effective bid for an ad query for “running shoes” originating from DC between 7-8 pm on a mobile device is $\$1 \times 2 \times 0.8 \times 1.5 = \2.4 . Using bid adjustments, the advertisers can potentially increase their bids for desired user

characteristics. They can also decrease their bids or not bid at all for certain feature items. For instance, setting the bid adjustment to 0 for DC ensures the effective bid is 0 for every feature combination containing DC for that campaign. Without bid adjustments, the advertisers do not have this flexibility. They would have to submit a flat bid, i.e., only the base bid, regardless of specific user characteristic targeted.

To illustrate the difference between using bid adjustments (and its potential benefits) and using a flat bid, consider the following example. Suppose an advertiser has a simple portfolio with one campaign, one ad group, and one keyword. The advertiser is interested in submitting adjustments for two features; location and device. The feature items for location are “DC” and “New York (NY)”, the feature items for device are “mobile (M)” and “desktop (D)”. If the advertiser is using bid adjustments, then the advertiser is expected to provide a base bid for the keyword, and adjustments for DC, NY, M, and D for the campaign. On the other hand, if the advertiser is not using bid adjustments, then only the base bid is required. In Table 3.1, we provide bid amounts and corresponding number of clicks, cost, and revenue data across all feature combinations. For simplicity, we consider three discrete bid levels. For instance, if the advertiser bids \$0.5, then the estimated number of clicks is 120, the estimated cost is \$55, and the estimated revenue is \$753.6. In Table 3.2, the distribution of clicks and cost, and the revenue-per-click (RPC) for each feature combination is provided. For instance, mobile users in NY account for 32% of the clicks and cost, and each click from these users leads to \$12 in revenue. Suppose the total budget for the advertiser is \$165. If the advertiser is not using bid adjustments, i.e., using a flat bid, then the optimal solution is to

Bid (\$)	# of clicks	Cost (\$)	Revenue (\$)
0.5	120	55	753.60
1	175	165	1099.00
2	210	275	1318.80

Table 3.1: Bids and corresponding number of clicks, cost, and revenue data

Feature Combination	Distribution	RPC
DC & M	16%	5
NY & M	32%	12
DC & D	32%	2
NY & D	20%	5

Table 3.2: Distribution of clicks and cost, and revenue-per-click based on feature combinations.

bid \$1 which results in \$1099.00 in expected revenue with an expected cost of \$165. However, if the advertiser is using bid adjustments, then the optimal solution is to have a base bid of \$1, an adjustment of 2 for NY, and an adjustment of 0.5 for D. No adjustments are made for DC and M, i.e., the adjustments are set to 1. The resulting expected revenue becomes \$1198.20 with an expected cost of \$165. In this example, using bid adjustments provides a 9% increase in the expected revenue for the same expected cost.

Our Contributions: The bid adjustments, when determined optimally, allow an unprecedented opportunity to effectively target desired user characteristics. There is a clear practical need for a formal treatment of the problem and an efficient and effective approach to determine bid adjustments. To that end, we introduce the Bid Adjustment Problem in Online Advertising as an operational revenue management problem faced by the advertiser. Given a portfolio (of campaigns, ad groups, and keywords), the advertiser needs to determine base bids for keywords and bid adjustments for ad groups and campaigns (depending on where the adjustments are

applied for a feature) such that the total expected revenue is maximized and the total expected cost does not exceed the advertising budget. Based on predictive models (provided by the advertiser) that estimate the expected cost and revenue given a bid amount and the specific user characteristics, the BAPOA automatically assesses the cost-revenue trade-off and determines bid adjustments such that desired user characteristics are targeted effectively. We propose a (nonlinear) mathematical formulation for the BAPOA. We then develop an algorithm where the mathematical formulation is decomposed into two tractable subproblems where base bids and adjustments are iteratively determined. The algorithm provides a feasible set of base bids and bid adjustments. We perform computational experiments on data generated based on a sample collected from Google Keyword Planner. In small instances (where it is possible to obtain upper bounds via CPLEX), we demonstrate the iterative algorithm finds near optimal solutions. We show that the quality of the iterative algorithm is robust under varying portfolio size, number of feature items, revenue and cost structures, and budget. Our findings indicate in an environment where there is significant variability in the cost-revenue trade-off as we vary user characteristics, effectively using bid adjustments creates significant increase in expected revenue. Finally, we explain how the iterative algorithm can be used to construct campaigns and ad groups and assign keywords to these campaigns and ad groups (as opposed to campaigns and ad groups being already setup). The advertiser only needs to determine the set of keywords of interest, the portfolio of campaigns and ad groups can be constructed, and the base bids and bid adjustments can be automatically determined with this procedure.

The rest of the chapter is organized as follows. In Section 3.1, we highlight and discuss related work in the literature. In Section 3.2, we define the BAPOA and develop a mathematical programming formulation for the BAPOA. In Section 3.3, we show the mathematical programming formulation proposed in Section 3.2 can be reduced to two types of subproblems and propose an iterative algorithm based on the subproblems. In Section 3.4, we show how the BAPOA can be modeled as mixed integer program with a discretization procedure. In Section 3.5, we discuss the computational results on simulated advertising portfolios.

3.1 Related Work

The introduction of bid adjustments is a relatively recent development in online advertising so virtually no research is conducted on the subject. Since Chapter 2 provides a comprehensive review of the traditional problems (that do not include bid adjustments) in online advertising, we focus on studies highlighting the importance of user characteristic targeting and discuss the only article (to the best of our knowledge) studying bid adjustments from a budget optimization standpoint.

The effectiveness of targeted advertising has been demonstrated in various studies in the marketing community. Iyer et al. [2005] show that targeted advertising eliminates waste by focusing advertising efforts away from consumers who have a preference for competing products. The authors argue while conventional wisdom suggests targeting should lead to lower advertising costs, the opposite is true when advertising is expensive since increased effectiveness due to targeting justifies

additional advertising spending. The authors show the ability to target is more important for profitability than the ability to price discriminate. [Beales \[2010\]](#) studies the effect of behavioral targeting in the context of online advertising. The author argues behavioral targeting is more effective and results in more than twice the conversion rate compared to run-of-network (where ads are placed in the advertising platform without targeting) advertising. [Yan et al. \[2009\]](#) perform an empirical study on a click-through log of an online advertising platform. The authors show the click-through rates can be improved significantly by targeted advertising in sponsored search. [Farahat and Bailey \[2012\]](#) study the marginal effectiveness of targeted advertising in online advertising. The authors show that the average click is over four times cheaper with targeted advertising and inducing brand-related searches from an ad is nine times cheaper with targeted advertising.

All studies mentioned in Chapter 2 suffer from one common shortcoming, in that none of the models make use of bid adjustments hence unable to capture the user characteristic cost-revenue trade-off. To the best of our knowledge, [Bateni et al. \[2014\]](#) performed the only study approaching bid adjustments from a budget optimization standpoint. The authors consider a problem with a single keyword with a unit base bid and two features. Each feature combination has a take-it-or-leave-it price for a single-slot auction and the adjustments are multiplied to obtain the effective bid. If the effective bid is more than the price, the price is paid and a reward is collected. The objective is to maximize the total reward such that the total price does not exceed the budget. The authors propose greedy algorithms and provide approximation results for various price and reward structures. Unless the price and

reward are neatly structured (e.g., prices are multiplicative and reward-to-price ratio is monotone), the problem has an approximation ratio of $O(\sqrt{n})$ where n denotes the number of feature items. The authors conclude that even with a single keyword with a unit base bid, two features, and single-slot auctions, the problem is hard to approximate. Though it demonstrates the difficulty of the problem, this work is of little practical importance since the assumptions (e.g., single keyword, single slot auction, price-reward structure, unit base bid, etc.) are not valid in problems in practice. In addition, the algorithms proposed have large approximation ratios and does not extend to problems where there is more than one keyword and the number of features is more than two.

3.2 The Bid Adjustment Problem in Online Advertising

In the BAPOA, ad queries for keywords arrive many times over a time horizon (e.g., one hour, one day, etc.) and each query has a feature combination (e.g., New York, 7-8pm, mobile device, female, 28 years old, etc.). The advertiser has available the expected cost and revenue for every keyword given a bid amount and a feature combination. Given the operational nature of the problem, the time horizon is typically one day in practice. The forecasts provided by advertising platforms (e.g., Google Keyword Planner) tend to be more accurate for the near future spanning shorter time horizons and for advertisers with longer advertising histories. For example, an advertiser with a few years of advertising history can get a relatively reliable forecast looking one day into the future as opposed to an advertiser with

one week of history looking three months into the future. In addition to the tools and forecasts provided by the advertising platforms, the advertisers possess ample historical performance and user browsing data to create accurate predictive models.

In the BAPOA, the objective is to determine a set of base bids for every keyword and a set of bid adjustments for every feature item and every campaign and ad group that would maximize the total expected revenue such that the total expected cost does not exceed the advertising budget. Let H be the set of campaigns and let G_h be the set of ad groups in campaign $h \in H$ such that $G = \bigcup_{h \in H} G_h$ denotes the set of all ad groups. For ad group $g \in G$, let K_g be the set of keywords such that $K = \bigcup_{g \in G} K_g$ denotes the set of all keywords. Define F as the set of features such that $F = \{F_1, F_2, \dots\}$ and define $F^H \subseteq F$ as the set of features for which the adjustments can be made at the campaign level. Similarly, define $F^G = F \setminus F^H$ as the set of features for which the adjustments can be made at the ad group level. Let \mathcal{F} be the feature combination set of F where $\mathcal{F} = \{F_1 \times F_2 \times \dots\}$. As an example, suppose there are two features; location and device. There are two feature items for location; District of Columbia (DC) and New York (NY). There are two feature items for device; mobile (M) and desktop (D). Then the set of features can be defined as $F = \{F_1 = \{\text{DC}, \text{NY}\}, F_2 = \{\text{M}, \text{D}\}\}$ and the set of feature combinations becomes $\mathcal{F} = \{\{\text{DC}, \text{M}\}, \{\text{DC}, \text{D}\}, \{\text{NY}, \text{M}\}, \{\text{NY}, \text{D}\}\}$. If location adjustments can be made at the campaign level and device adjustments can be made at the ad group level, then $F^H = \{F_1\}$ and $F^G = \{F_2\}$.

Let β be the set of base bids such that $\beta = \bigcup_{i \in K} \beta_i$ where $\beta_i \in [0, \infty)$. In both Google and Bing, a bid adjustment for a feature item can either be zero,

in that case the advertiser chooses not to bid on any combination containing the feature item, or the bid adjustment has to be in a positive interval. To model the adjustments, we use continuous and binary variables. Let $\boldsymbol{\alpha} = \bigcup_{F_t \in F} \boldsymbol{\alpha}_t$ be the set of continuous adjustment variables and $\mathbf{y} = \bigcup_{F_t \in F} \mathbf{y}_t$ be the set of binary adjustment variables such that if $F_t \in F^H$, then $\boldsymbol{\alpha}_t = \bigcup_{h \in H, \ell \in F_t} \alpha_h^\ell$ and $\mathbf{y}_t = \bigcup_{h \in H, \ell \in F_t} y_h^\ell$, and if $F_t \in F^G$, then $\boldsymbol{\alpha}_t = \bigcup_{g \in G, \ell \in F_t} \alpha_g^\ell$ and $\mathbf{y}_t = \bigcup_{g \in G, \ell \in F_t} y_g^\ell$. For each feature item $\ell \in F^H$, the continuous adjustment variable $\alpha_h^\ell \in [L_h^\ell, U_h^\ell]$ is applied to every keyword in campaign h if binary variable y_h^ℓ is 0. If $y_h^\ell = 1$ then the bid adjustment is zero and the advertiser does not bid on any combination $f \in \mathcal{F}$ with $\ell \in f$. The domains for α_g^ℓ and y_g^ℓ for $\ell \in F^G$ are defined analogously. Simply put, the bid adjustment for campaign h and feature item $\ell \in F^H$ can be stated as $\alpha_h^\ell(1 - y_h^\ell)$ whereas the bid adjustment for ad group g and feature item $\ell \in F^G$ is stated as $\alpha_g^\ell(1 - y_g^\ell)$. Given a base bid and a set of bid adjustments, the effective bid for keyword $i \in K_g$ of ad group g of campaign h and combination $f \in \mathcal{F}$ can be calculated as $b_i^f = \beta_i \prod_{\ell \in f^H} \alpha_h^\ell(1 - y_h^\ell) \prod_{\ell \in f^G} \alpha_g^\ell(1 - y_g^\ell)$ where $f^H \subseteq f$ and $f^G = f \setminus f^H$ denote the sets of feature items in f for which the adjustments can be made at the campaign and the ad group level, respectively. Let $c_i^f(b)$ and $r_i^f(b)$ denote the expected cost and revenue functions (obtained via predictive models) for keyword i and combination f , i.e., if the effective bid is b_i^f , the expected cost is predicted to be $c_i^f(b_i^f)$ and the expected revenue is predicted to be $r_i^f(b_i^f)$. The BAPOA can be modeled mathematically as follows.

BAPOA($\beta, \alpha, \mathbf{y}$):

$$\begin{aligned} & \max_{\beta, \alpha, \mathbf{y}} \sum_{i \in K} \sum_{f \in \mathcal{F}} r_i^f(b_i^f) \\ \text{subject to } & \sum_{i \in K} \sum_{f \in \mathcal{F}} c_i^f(b_i^f) \leq B \end{aligned} \quad (3.1)$$

$$\begin{aligned} b_i^f = \beta_i \prod_{\ell \in f^H} \alpha_h^\ell (1 - y_h^\ell) \prod_{\ell \in f^G} \alpha_g^\ell (1 - y_g^\ell) \quad & h \in H, g \in G_h, \\ & i \in K_g, f \in \mathcal{F} \end{aligned} \quad (3.2)$$

$$\beta_i \geq 0 \quad i \in K \quad (3.3)$$

$$L_h^\ell \leq \alpha_h^\ell \leq U_h^\ell \quad h \in H, \ell \in F^H \quad (3.4)$$

$$L_g^\ell \leq \alpha_g^\ell \leq U_g^\ell \quad g \in G, \ell \in F^G \quad (3.5)$$

$$y_h^\ell \in \{0, 1\} \quad h \in H, \ell \in F^H \quad (3.6)$$

$$y_g^\ell \in \{0, 1\} \quad g \in G, \ell \in F^G \quad (3.7)$$

In the BAPOA($\beta, \alpha, \mathbf{y}$), we need to determine the base bids and the bid adjustments such that the total expected revenue is maximized subject to a budget. Constraint (3.1) ensures that the total expected cost does not exceed the budget. Constraint set (3.2) calculates the effective bid for every combination of every keyword by using the base bid and bid adjustments. Even though this formulation reflects the current state of the bidding language where adjustments can only be made at one of two levels (campaign or ad group level), observe that it can be extended to cases where adjustments can be made in arbitrary number of levels. The

BAPOA($\beta, \alpha, \mathbf{y}$) is a non-smooth and non-convex problem, which makes it very hard and impractical to solve. Therefore, we develop a mathematical programming based heuristic algorithm to obtain a high quality solution to the BAPOA($\beta, \alpha, \mathbf{y}$).

3.3 Iterative Adjustment Algorithm

We develop the Iterative Adjustment Algorithm (IAA) to obtain high quality feasible solutions for the BAPOA in reasonable computational time. In each iteration of the IAA, we solve a series of subproblems to obtain base bids and bid adjustments. If the subproblem is solved to obtain base bids, we assume the adjustments for all features are known and fixed. If the subproblem is solved to obtain bid adjustments for one feature (e.g., device), then we assume the base bids and bid adjustments for other features (e.g., location, time) are known and fixed. Either base bids or bid adjustments for one feature are determined every time a subproblem is solved and they become inputs for the next subproblem. In Sections 3.3.1 and 3.3.2, we describe how the subproblems are set up. We then show that when the feasible region is discretized, both types of subproblems can be modeled as a Multiple Choice Knapsack Problem (MCKP). In Section 3.3.3, we show how the solution to the linear programming relaxation of the MCKP can be used to obtain feasible base bids and bid adjustments. In Section 3.3.4, we put the pieces together and summarize the Iterative Adjustment Algorithm.

3.3.1 Base Bid Subproblem

Suppose α and \mathbf{y} are known and fixed, then the BAPOA($\beta, \alpha, \mathbf{y}$) can be reduced to a subproblem where β is determined. Let BAPOA(β) denote the base bid subproblem formulated as follows.

BAPOA(β):

$$\max_{\beta} \sum_{i \in K} \sum_{f \in \mathcal{F}} r_i^f(b_i^f)$$

(3.1) – (3.7)

Constraint sets (3.4)-(3.7) can be removed from the BAPOA(β) since α and \mathbf{y} are fixed parameters. Furthermore, we only need β_i to calculate the bid b_i^f , effectively making c_i^f and r_i^f functions of β_i . Therefore, we define $c_i(\beta_i) = \sum_{f \in \mathcal{F}} c_i^f(b_i^f)$ and $r_i(\beta_i) = \sum_{f \in \mathcal{F}} r_i^f(b_i^f)$ as the total expected cost and revenue for keyword i corresponding to base bid β_i , and remove constraint set (3.2). We can now express the BAPOA(β) more compactly as follows.

BAPOA(β):

$$\max_{\beta} \sum_{i \in K} r_i(\beta_i)$$

subject to $\sum_{i \in K} c_i(\beta_i) \leq B$

$$\beta_i \geq 0$$

$$i \in K$$

The BAPOA(β) is a simpler problem than BAPOA($\beta, \alpha, \mathbf{y}$), however, it is still a challenging problem to solve optimally. We now describe a discrete approximation to the BAPOA(β) and show that it can be modeled as a Multiple Choice Knapsack Problem.

Proposition 3.3.1 *Discretize the domain of the base bid β_i . Let $\mathcal{P} = \{0, 1, \dots\}$ denote the set of base bid levels such that $\beta_{ij} \geq 0$ is defined as the base bid value corresponding to base bid level $j \in \mathcal{P}$. Then the discretized BAPOA(β) can be modeled as a Multiple Choice Knapsack Problem, which we denote as D-BAPOA(β).*

Proof Given that the base bid β_i has a discrete domain over the set of base bid levels \mathcal{P} , the D-BAPOA(β) is modeled as follows.

D-BAPOA(β):

$$\begin{aligned} & \max_{\beta} \sum_{i \in K} r_i(\beta_i) \\ & \text{subject to } \sum_{i \in K} c_i(\beta_i) \leq B \\ & \beta_i \in \{\beta_{ij} \mid j \in \mathcal{P}\} \qquad i \in K \end{aligned}$$

Define $c_{ij} = c_i(\beta_{ij})$ and $r_{ij} = r_i(\beta_{ij})$ as the total expected cost and revenue from keyword i corresponding to base bid value β_{ij} . Let x_{ij} be a binary variable taking the value 1 if β_{ij} is selected as the base bid level, 0 otherwise. Then the D-BAPOA(β)

can be restated as follows.

D-BAPOA(β):

$$\begin{aligned}
& \max_{\mathbf{x}} \sum_{i \in K} \sum_{j \in \mathcal{P}} r_{ij} x_{ij} \\
& \text{subject to } \sum_{i \in K} \sum_{j \in \mathcal{P}} c_{ij} x_{ij} \leq B \\
& \sum_{j \in \mathcal{P}} x_{ij} = 1 \quad i \in K \quad (3.8) \\
& x_{ij} \in \{0, 1\} \quad i \in K, j \in \mathcal{P}
\end{aligned}$$

To ensure exactly one β_{ij} is selected as the base bid level, we replace the domain constraint of β_i with the constraint set (3.8) in the D-BAPOA(β). A discrete base bid domain allows us to evaluate $c_i(\beta_i)$ and $r_i(\beta_i)$ over a discrete set of points and parameterize them for the D-BAPOA(β). Thus, the D-BAPOA(β) is an MCKP where each class corresponds to a keyword and each level corresponds to a base bid value. Note that when we discretize the domain of the base bid, we designate level $0 \in \mathcal{P}$ as the zero base bid level, i.e., $\beta_{i0} = 0$ for every keyword. ■

3.3.2 Feature Adjustment Subproblem

For $F_t \in F^G$, suppose β , $\alpha \setminus \alpha_t$, and $\mathbf{y} \setminus \mathbf{y}_t$ are known and fixed. Similar to the base bid subproblem, the BAPOA($\beta, \alpha, \mathbf{y}$) can be reduced to a subproblem where α_t and \mathbf{y}_t are determined. Note that we only derive the feature adjustment subproblem and discuss the solution method for features where the adjustments are determined

at the ad group level, i.e., $F_t \in F^G$. However, the feature adjustment subproblem can be analogously derived and the solution method (discussed in Section 3.3.3) directly applies for features where the adjustments are determined at the campaign level, i.e., $F_t \in F^H$. Let $\text{BAPOA}(\boldsymbol{\alpha}_t, \mathbf{y}_t)$ denote the feature adjustment subproblem formulated as follows.

$\text{BAPOA}(\boldsymbol{\alpha}_t, \mathbf{y}_t)$:

$$\max_{\boldsymbol{\alpha}_t, \mathbf{y}_t} \sum_{i \in K} \sum_{f \in \mathcal{F}} r_i^f(b_i^f)$$

(3.1) – (3.7)

Constraint set (3.3) can be removed from the $\text{BAPOA}(\boldsymbol{\alpha}_t, \mathbf{y}_t)$ since $\boldsymbol{\beta}$ is a set of fixed parameters. In addition, constraints pertaining to $\boldsymbol{\alpha} \setminus \boldsymbol{\alpha}_t$ and $\mathbf{y} \setminus \mathbf{y}_t$ in constraint sets (3.4)-(3.7) can be removed from the $\text{BAPOA}(\boldsymbol{\alpha}_t, \mathbf{y}_t)$ since $\boldsymbol{\alpha} \setminus \boldsymbol{\alpha}_t$ and $\mathbf{y} \setminus \mathbf{y}_t$ are known. Finally, c_i^f and r_i^f become functions of α_g^ℓ and y_g^ℓ where $i \in K_g$ and $\ell \in f$ since we only need α_g^ℓ and y_g^ℓ to calculate b_i^f . Therefore, constraint set (3.2) can also be removed from the $\text{BAPOA}(\boldsymbol{\alpha}_t, \mathbf{y}_t)$. Define $c_g^\ell(\alpha_g^\ell, y_g^\ell) = \sum_{i \in K_g} \sum_{f \in \mathcal{F} | \ell \in f} c_i^f(b_i^f)$ and $r_g^\ell(\alpha_g^\ell, y_g^\ell) = \sum_{i \in K_g} \sum_{f \in \mathcal{F} | \ell \in f} r_i^f(b_i^f)$ as the total expected cost and revenue for ad group g and feature item ℓ corresponding to adjustment $\alpha_g^\ell(1 - y_g^\ell)$. The objective function of the $\text{BAPOA}(\boldsymbol{\alpha}_t, \mathbf{y}_t)$ can be restated as follows.

$$\begin{aligned} \max_{\boldsymbol{\alpha}_t, \mathbf{y}_t} \sum_{i \in K} \sum_{f \in \mathcal{F}} r_i^f(b_i^f) &= \max_{\boldsymbol{\alpha}_t, \mathbf{y}_t} \sum_{g \in G} \sum_{i \in K_g} \sum_{f \in \mathcal{F}} r_i^f(b_i^f) \\ &= \max_{\boldsymbol{\alpha}_t, \mathbf{y}_t} \sum_{g \in G} \sum_{i \in K_g} \sum_{\ell \in F_i} \sum_{f \in \mathcal{F} | \ell \in f} r_i^f(b_i^f) \end{aligned}$$

$$\begin{aligned}
&= \max_{\boldsymbol{\alpha}_t, \mathbf{y}_t} \sum_{g \in G} \sum_{\ell \in F_t} \sum_{i \in K_g} \sum_{f \in \mathcal{F} | \ell \in f} r_i^f(b_i^f) \\
&= \max_{\boldsymbol{\alpha}_t, \mathbf{y}_t} \sum_{g \in G} \sum_{\ell \in F_t} r_g^\ell(\alpha_g^\ell, y_g^\ell)
\end{aligned}$$

Constraint (3.1) in the BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$) can be restated in a similar fashion. Then we can express the BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$) more compactly as follows.

BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$):

$$\begin{aligned}
&\max_{\boldsymbol{\alpha}_t, \mathbf{y}_t} \sum_{g \in G} \sum_{\ell \in F_t} r_g^\ell(\alpha_g^\ell, y_g^\ell) \\
&\text{subject to } \sum_{g \in G} \sum_{\ell \in F_t} c_g^\ell(\alpha_g^\ell, y_g^\ell) \leq B \\
&L_g^\ell \leq \alpha_g^\ell \leq U_g^\ell \quad g \in G, \ell \in F_t \\
&y_g^\ell \in \{0, 1\} \quad g \in G, \ell \in F_t
\end{aligned}$$

Similar to the base bid subproblem, we describe a discrete approximation (where $\boldsymbol{\alpha}_t$ has discrete domain) to the BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$) and show that it can modeled as a Multiple Choice Knapsack Problem.

Proposition 3.3.2 *Discretize the domain of the adjustment α_g^ℓ . Let $\mathcal{P} = \{1, 2, \dots\}$ denote the set of adjustment levels such that $L_g^\ell \leq \alpha_{gj}^\ell \leq U_g^\ell$ is defined as the continuous adjustment value corresponding to adjustment level $j \in \mathcal{P}$. Then the discretized BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$) can be modeled as a Multiple Choice Knapsack Problem, which we denote as D-BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$).*

Proof Given that the continuous adjustment α_g^ℓ has a discrete domain over the set

of adjustment levels \mathcal{P} , the D-BAPOA(α_t, \mathbf{y}_t) is modeled as follows.

D-BAPOA(α_t, \mathbf{y}_t):

$$\begin{aligned}
& \max_{\alpha_t, \mathbf{y}_t} \sum_{g \in G} \sum_{\ell \in F_t} r_g^\ell(\alpha_g^\ell, y_g^\ell) \\
& \text{subject to } \sum_{g \in G} \sum_{\ell \in F_t} c_g^\ell(\alpha_g^\ell, y_g^\ell) \leq B \\
& \alpha_g^\ell \in \{\alpha_{gj}^\ell \mid j \in \mathcal{P}\} \quad g \in G, \ell \in F_t \\
& y_g^\ell \in \{0, 1\} \quad g \in G, \ell \in F_t
\end{aligned}$$

Define $c_{gj}^\ell = c_g^\ell(\alpha_{gj}^\ell, 0)$ and $r_{gj}^\ell = r_g^\ell(\alpha_{gj}^\ell, 0)$ as the total expected cost and revenue from ad group g and feature item ℓ corresponding to adjustment α_{gj}^ℓ where y_g^ℓ is 0. Let x_{gj}^ℓ be a binary variable taking the value 1 if adjustment value α_{gj}^ℓ is selected, 0 otherwise. Then the D-BAPOA(α_t, \mathbf{y}_t) can be restated as follows.

D-BAPOA(α_t, \mathbf{y}_t):

$$\begin{aligned}
& \max_{\mathbf{x}, \mathbf{y}_t} \sum_{g \in G} \sum_{\ell \in F_t} \sum_{j \in \mathcal{P}} r_{gj}^\ell x_{gj}^\ell \\
& \text{subject to } \sum_{g \in G} \sum_{\ell \in F_t} \sum_{j \in \mathcal{P}} c_{gj}^\ell x_{gj}^\ell \leq B \\
& \sum_{j \in \mathcal{P}} x_{gj}^\ell + y_g^\ell = 1 \quad g \in G, \ell \in F_t \quad (3.9) \\
& y_g^\ell \in \{0, 1\} \quad g \in G, \ell \in F_t \\
& x_{gj}^\ell \in \{0, 1\} \quad g \in G, \ell \in F_t, j \in \mathcal{P}
\end{aligned}$$

We replace the domain constraint of α_g^ℓ with the constraint set (3.9) in the

D-BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$) to ensure either an adjustment value α_{gj}^ℓ corresponding to adjustment level j is selected or y_g^ℓ is 1. If y_g^ℓ is 1, the adjustment is 0 for ad group g and feature item ℓ , then the expected cost and revenue for all feature combinations f in ad group g such that $\ell \in f$ are also 0. We now show the D-BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$) is equivalent to the Multiple Choice Knapsack Problem. First we replace y_g^ℓ by a binary variable x_{g0}^ℓ (i.e., $y_g^\ell = x_{g0}^\ell$). We expand \mathcal{P} to include the zero adjustment level $\{0\}$ corresponding to $y_g^\ell = 1$ where $c_{g0}^\ell = 0$ and $r_{gj}^\ell = 0$. Let $\mathcal{G} = G \times F_t$ be the set of ad group-feature item tuples. An element $\mathbf{g} \in \mathcal{G}$ corresponds to a specific ad group $g \in G$ and feature item $\ell \in F_t$ tuple. For instance, suppose we set up the BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$) for the location feature and the set of feature items is $F_t = \{\text{DC}, \text{NY}\}$ and the set of ad groups is $G = \{\text{apparel}, \text{footwear}\}$, then the set of ad group-feature item tuples is $\mathcal{G} = \{\{\text{apparel}, \text{DC}\}, \{\text{apparel}, \text{NY}\}, \{\text{footwear}, \text{DC}\}, \{\text{footwear}, \text{NY}\}\}$. Finally, we define $c_{\mathbf{g}j} = c_{gj}^\ell$, $r_{\mathbf{g}j} = r_{gj}^\ell$, and $x_{\mathbf{g}j} = x_{gj}^\ell$. Then the D-BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$) can be expressed as follows.

D-BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$):

$$\begin{aligned}
& \max_{\mathbf{x}} \sum_{\mathbf{g} \in \mathcal{G}} \sum_{j \in \mathcal{P}} r_{\mathbf{g}j} x_{\mathbf{g}j} \\
& \text{subject to } \sum_{\mathbf{g} \in \mathcal{G}} \sum_{j \in \mathcal{P}} c_{\mathbf{g}j} x_{\mathbf{g}j} \leq B \\
& \sum_{j \in \mathcal{P}} x_{\mathbf{g}j} = 1 \quad \mathbf{g} \in \mathcal{G} \\
& x_{\mathbf{g}j} \in \{0, 1\} \quad \mathbf{g} \in \mathcal{G}, j \in \mathcal{P}
\end{aligned}$$

The D-BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$) is a Multiple Choice Knapsack Problem where each class

corresponds to an ad group-feature item tuple, the level 0 corresponds to the binary adjustment, and the remaining multiple choice levels correspond to adjustments.

■

Showing that both the base bid and the feature adjustment subproblems can be modeled as an MCKP when the domain of the base bids and the continuous adjustments are discretized results in a practical computational procedure to solve the BAPOA.

3.3.3 The Multiple Choice Knapsack Problem

We discuss the Multiple Choice Knapsack Problem at length in Chapter 2. In this section, we explain how a solution to the linear programming relaxation to the MCKP (MCKP-LP) can be used to obtain a feasible solution to the BAPOA. We use the column generation algorithm developed in Chapter 2 to solve the MCKP-LPs corresponding to the base bid and feature adjustment subproblems. Generically, the MCKP can be represented as follows where N is the set of classes and \mathcal{P}_k is the set of levels for class k .

$$\begin{aligned} & \text{Maximize} && \sum_{k \in N} \sum_{j \in \mathcal{P}_k} r_{kj} x_{kj} \\ & \text{Subject to} && \sum_{k \in N} \sum_{j \in \mathcal{P}_k} c_{kj} x_{kj} \leq B \end{aligned} \tag{3.10}$$

$$\sum_{j \in \mathcal{P}_k} x_{kj} = 1 \quad k \in N \tag{3.11}$$

$$x_{kj} \in \{0, 1\} \quad k \in N, j \in \mathcal{P}_k$$

$N = K$ in the base bid subproblem and $N = \mathcal{G}$ in the feature adjustment subproblem. Let MCKP-LP denote the linear programming relaxation of the MCKP.

Proposition 2.2.5 in Chapter 2 states that regardless of the number of classes and levels in the problem, the number of fractional variables in the optimal solution to the MCKP-LP is at most two. Since the MCKP is a discretized approximation of the (base bid or feature adjustment) subproblem, this important result allows us to easily obtain feasible solutions for the subproblem from the MCKP-LP solution. The variables with integer values (i.e., $x_{kj} = 1$) in the optimal MCKP-LP solution directly correspond to discrete input values of base bids (in the base bid subproblem) and bid adjustments (in the feature adjustment subproblem). We therefore set the values of base bids and bid adjustments for these variables directly to their corresponding discretized input values. However, taking the convex combination of base bid (or bid adjustments) values corresponding to the two fractional variables may result in an infeasible solution for the subproblem since the MCKP is only a discretized *approximation* of the subproblem. Therefore, any base bid or bid adjustment value not corresponding to a discretized input value in the MCKP may violate the budget (and for the feature adjustment subproblem, the domain) constraint of the BAPOA. To overcome this issue, we perform binary search between the base bid (or bid adjustment) values corresponding to the two fractional variables until we find a feasible base bid (or bid adjustment) value that exhausts the budget.

We now describe, mathematically, how the MCKP-LP solution can be used to obtain a feasible solution for the (base bid or feature adjustment) subproblem. After solving the MCKP-LP, let \mathbf{x}^* denote the optimal solution. In the base bid

subproblem, base bid $\beta_{i'}$ can be set as $\beta_{i'} = \beta_{i'j^*}$ for classes where $x_{k'j^*} = 1$ and keyword i' corresponds to class k' . Let $x_{kj'}$ and $x_{kj''}$ be the two fractional variables from class k corresponding to keyword i such that $\beta_{ij'} < \beta_{ij''}$. Then define \bar{c}_i as the budget allocated in the MCKP for keyword i such that $\bar{c}_i = x_{kj'}c_{kj'} + x_{kj''}c_{kj''}$. We first set the base bid for keyword i as $\beta_i = x_{kj'}\beta_{ij'} + x_{kj''}\beta_{ij''}$ and compare the total expected cost of keyword i (i.e., $c_i(\beta_i)$) with the budget allocated in the MCKP. If $c_i(\beta_i) > \bar{c}_i$, then β_i is not a feasible base bid for keyword i since the expected cost exceeds the budget. Under the assumption⁵ that $c_i^f(b)$ and $r_i^f(b)$ are nondecreasing functions of b , there exists a feasible base bid value in the range $[\beta_{ij'}, x_{kj'}\beta_{ij'} + x_{kj''}\beta_{ij''}]$ that exhausts the budget. If $c_i(\beta_i) \leq \bar{c}_i$ then β_i is a feasible base bid but we might still find a better value for the base bid in the range $[x_{kj'}\beta_{ij'} + x_{kj''}\beta_{ij''}, \beta_{ij''}]$. In either case, we perform binary search to efficiently search the range such that β_i is a feasible base bid and $c_i(\beta_i)$ is as close to \bar{c}_i as possible.

In the feature adjustment subproblem, suppose ad group-feature item tuple $\mathbf{g}' = (g', \ell')$ corresponds to class k' such that $x_{k'j^*} = 1$, then the continuous and binary adjustments can be set as follows.

$$\alpha_{g'}^{\ell'} = \alpha_{g'j^*}^{\ell'} \text{ if } j^* > 0$$

⁵There are various reports listed at <https://moz.com/blog/google-organic-click-through-rates-in-2014> demonstrating the click-through-rate is a nondecreasing function of the bid, coupled with the fact that cost-per-click (or cost-per-mille in impression based pricing models) is a nondecreasing function of the bid due to the auction mechanism, implies $c_i^f(b)$ and $r_i^f(b)$ are nondecreasing functions of b

$$y_{g'}^{\ell'} = \begin{cases} 1 & \text{if } j^* = 0 \\ 0 & \text{otherwise} \end{cases}$$

Let $x_{kj'}$ and $x_{kj''}$ be the two fractional variables from class k corresponding to ad group-feature item tuple $\mathbf{g} = (g, \ell)$ such that $\alpha_{gj'}^{\ell} < \alpha_{gj''}^{\ell}$, then we initially set the continuous adjustment $\alpha_g^{\ell} = x_{kj'}\alpha_{gj'}^{\ell} + x_{kj''}\alpha_{gj''}^{\ell}$ if $j' \neq 0$, and $\alpha_g^{\ell} = x_{kj''}\alpha_{gj''}^{\ell}$ otherwise. We initially set the binary adjustment $y_g^{\ell} = 0$. Then we perform binary search (analogous to the base bid subproblem) to find a feasible adjustment value for $\alpha_{gj''}^{\ell}$ that would use as much of the budget as possible. However, if α_g^{ℓ} found via binary search does not satisfy $L_g^{\ell} \leq \alpha_g^{\ell}$, then we set $y_g^{\ell} = 1$.

3.3.4 Summary of the Steps in the Iterative Adjustment Algorithm

We outline the Iterative Adjustment Algorithm in Algorithm 5. Let $R(\boldsymbol{\beta})$ and $R(\boldsymbol{\alpha}_t, \mathbf{y}_t)$ denote the revenue obtained from the base bid subproblem D-BAPOA($\boldsymbol{\beta}$) and feature adjustment subproblem D-BAPOA($\boldsymbol{\alpha}_t, \mathbf{y}_t$), respectively. Initially, we set $\boldsymbol{\alpha} = \mathbf{1}$ and $\mathbf{y} = \mathbf{0}$. Solving the base bid subproblem when $\boldsymbol{\alpha} = \mathbf{1}$ and $\mathbf{y} = \mathbf{0}$ provides a solution for the case where bid adjustments are not used, which we refer to as the ‘‘Flat Bid’’ solution. In line 5, we solve the base bid subproblem and update the base bids according to the MCKP-LP solution. At each iteration of the for loop between lines 9 - 14, we solve the feature adjustment subproblem and update the adjustments according to the MCKP-LP solution. The algorithm terminates when it can no longer improve the best known solution with expected revenue R^* .

In the first iteration of the while loop in Algorithm 5, the adjustment values

Algorithm 5 Iterative Adjustment Algorithm

```
1: set  $R^* := 0, R' := -\infty$ 
2: set  $\alpha = 1, \mathbf{y} = 0$ 
3: while  $R^* > R'$  do
4:   set  $R' := R^*$ 
5:   solve D-BAPOA( $\beta$ ), set  $\beta$ 
6:   if  $R(\beta) > R^*$  then
7:     set  $R^* := R(\beta)$ 
8:   end if
9:   for  $t = 1, \dots, |F|$  do
10:    solve D-BAPOA( $\alpha_t, \mathbf{y}_t$ ), set  $\alpha_t, \mathbf{y}_t$ 
11:    if  $R(\alpha_t, \mathbf{y}_t) > R^*$  then
12:      set  $R^* := R(\alpha_t, \mathbf{y}_t)$ 
13:    end if
14:  end for
15: end while
16: return  $R^*$ 
```

α_{gj}^ℓ corresponding to adjustment levels $j \in \mathcal{P}$ in the discretization are selected such that they span their respective ranges, i.e., $\alpha_{gj}^\ell \in [L_g^\ell, U_g^\ell]$. In subsequent iterations of the while loop, the adjustment values α_{gj}^ℓ in the discretization are selected from some neighborhood of α_g^ℓ found in the previous iteration, i.e., $\alpha_{gj}^\ell \in [\alpha_g^\ell - \gamma, \alpha_g^\ell + \gamma]$. In other words, in the initial iteration the discretization covers the entire domain, but in later iterations the discretization is refined in the neighborhood of the current solution.

Unlike the continuous adjustments, the base bid does not have an explicit upper bound, i.e., the bidding language does not prohibit the advertiser from using an arbitrarily large base bid. However, since the ad positions on the page are awarded according to a generalized second price auction (in bidding languages using bid adjustments), there exists a large enough bid where bidding more than that amount has no effect on the cost-per-click, the number of clicks and the corresponding rev-

enue. In other words, there exists a threshold bid amount b_i^{f*} large enough such that for any bid $b_i^f > b_i^{f*}$, we have $c_i^f(b_i^f) = c_i^f(b_i^{f*})$ and $r_i^f(b_i^f) = r_i^f(b_i^{f*})$. For the IAA, we set the threshold bid amount b_i^* for keyword i as $b_i^* = \max_{f \in \mathcal{F}} b_i^{f*}$. In the first iteration of the while loop we select base bid levels $j \in \mathcal{P}$ such that the discretization covers the entire domain $\beta_{ij} \in [0, b_i^*]$. In the subsequent iterations, the discretization of the base bid values is in the neighborhood of the β_i found in the previous iteration, i.e., $\beta_{ij} \in [\beta_i - \gamma, \beta_i + \gamma]$. For both the continuous bid adjustments and the base bids, γ is made smaller after each iteration to increase the precision of the search (and discretization).

3.4 Formulating the BAPOA as an MIP

We model the BAPOA as a mixed integer program and compare (in Section 3.5.2) the upper bound obtained from the mixed integer program with the feasible solution obtained from the IAA for very small instances.

Proposition 3.4.1 *Suppose the effective bid amounts b_i^f are discretized on the domain and any effective bid value between two discrete values is rounded down to the lower one. Then the BAPOA can be modeled as a mixed integer program.*

Proof To model the BAPOA as a mixed integer program, we use the discretized effective bid amounts (e.g., corresponding to one cent increments since the effective bid is rounded down to the nearest cent in Bing⁶). Let $P_i^f = \{0, 1, \dots\}$ denote the set of bid levels for each increment in the discretization of the range $[0, b_i^{f*}]$ (where

⁶<https://help.bingads.microsoft.com/apex/index/3/en-au/51004>

b_i^{f*} denotes the threshold bid). For instance, if $b_i^{f*} = \$3$ and the discretization is at a “cent” level, then the set of bid levels is $P_i^f = \{0, 1, \dots, 300\}$ corresponding to bid amounts $\{\$0, \$0.01, \$0.02, \dots, \$3\}$. To generalize, we have a set of bid levels P_i^f for every keyword i and combination f corresponding to bid amounts $\{b_{i0}^f, b_{i1}^f, \dots\}$. For instance, if the effective bid b_i^f satisfies $b_{ij}^f \leq b_i^f < b_{i,j+1}^f$, then the bid is rounded down to b_{ij}^f and the expected cost and revenue corresponding to bid level j are realized. A discrete bid space allows us to evaluate $c_i^f(b)$ and $r_i^f(b)$ for every bid level in P_i^f and parameterize them for the mixed integer program. Define $c_{ij}^f = c_i^f(b_{ij}^f)$ and $r_{ij}^f = r_i^f(b_{ij}^f)$ for each $j \in P_i^f$ as the expected cost and revenue corresponding to bid level j . In addition, we take the logarithm (by using log adjustment (α_h^ℓ and α_g^ℓ) and log base bid (β_i') variables) of the effective bid and express it as a summation rather than a product. Let $b_{ij}^f = \ln b_{ij}^f$, $L_h^\ell = \ln L_h^\ell$, $U_h^\ell = \ln U_h^\ell$, $L_g^\ell = \ln L_g^\ell$, and $U_g^\ell = \ln U_g^\ell$. The BAPOA can be modeled with the mixed integer programming formulation presented as follows.

BAPOA(MIP):

$$\max_{\xi, \beta', \alpha', \mathbf{y}} \sum_{i \in K} \sum_{f \in \mathcal{F}} \sum_{j \in P_i^f} r_{ij}^f \xi_{ij}^f$$

$$\text{subject to } \sum_{i \in K} \sum_{f \in \mathcal{F}} \sum_{j \in P_i^f} c_{ij}^f \xi_{ij}^f \leq B \quad (3.12)$$

$$\sum_{j \in P_i^f} \xi_{ij}^f = 1 \quad i \in K, f \in \mathcal{F} \quad (3.13)$$

$$\sum_{j \in P_i^f} b_{ij}^f \xi_{ij}^f \leq \beta_i' + \sum_{\ell \in f^H} \alpha_h^\ell + \sum_{\ell \in f^G} \alpha_g^\ell - s_i^f \quad h \in H, g \in G_h, \quad i \in K_g, f \in \mathcal{F} \quad (3.14)$$

$$\beta_i' + \sum_{\ell \in f^H} \alpha_h^\ell + \sum_{\ell \in f^G} \alpha_g^\ell - s_i^f \leq \quad (3.15)$$

$$\sum_{j \in P_i^f \setminus \{|P_i^f| - 1\}} b_{i,j+1}^f \xi_{ij}^f + M_i^f \xi_{i,|P_i^f| - 1}^f \quad h \in H, g \in G_h, \quad i \in K_g, f \in \mathcal{F} \quad (3.16)$$

$$L_h^\ell \leq \alpha_h^\ell \leq U_h^\ell \quad h \in H, \ell \in F \quad (3.17)$$

$$L_g^\ell \leq \alpha_g^\ell \leq U_g^\ell \quad g \in G, \ell \in F \quad (3.18)$$

$$M_i^f \left(\sum_{\ell \in f^H} y_h^\ell + \sum_{\ell \in f^G} y_g^\ell \right) \geq s_i^f \quad i \in K, f \in \mathcal{F} \quad (3.19)$$

$$\xi_{i0}^f \geq y_h^\ell \quad h \in H, i \in K, \quad f \in \mathcal{F}, \ell \in f \quad (3.20)$$

$$\xi_{i0}^f \geq y_g^\ell \quad g \in G, i \in K_g, \quad f \in \mathcal{F}, \ell \in f \quad (3.21)$$

$$s_i^f \geq 0 \quad i \in K, f \in \mathcal{F}$$

$$y_h^\ell \in \{0, 1\} \quad h \in H, \ell \in F$$

$$y_g^\ell \in \{0, 1\} \quad g \in G, \ell \in F$$

$$\xi_{ij}^f \in \{0, 1\} \quad i \in K, f \in \mathcal{F},$$

$$j \in P_i^f$$

ξ_{ij}^f is a binary variable taking the value 1 if bid level j is selected for combi-

nation f and for keyword i where $\xi = \bigcup_{i \in K, f \in \mathcal{F}, j \in P_i^f} \xi_{ij}^f$. If binary variable y_h^ℓ (y_g^ℓ) is 1 then the bid adjustment for feature item ℓ is 0 for campaign h (ad group g). The log adjustments α_h^ℓ and α_g^ℓ , and log base bid β_i are continuous variables and unrestricted in sign. The objective function maximizes the total expected revenue across all keywords, combinations, and bid levels. Constraint (3.12) states the total expected cost cannot exceed the budget. Constraint set (3.13) ensures that only one bid level is chosen for each combination of each keyword. In constraint sets (3.14) and (3.16), the logarithm of the effective bid is calculated based on the log adjustments and the log base bid, then the corresponding bid level is identified. If the logarithm of the effective bid is greater than $\ln b_i^{f*}$, the bid level corresponding to the threshold bid b_i^{f*} is selected. We set M_i^f to a large enough constant such that no effective bid can exceed M_i^f . We set $M_i^f = \ln(b_i^{f*} \prod_{\ell \in f^H} U_h^\ell \prod_{\ell \in f^G} U_g^\ell)$ such that keyword i is in ad group g of campaign h . The scaling variable s_i^f is used to ensure an appropriate amount is subtracted from the effective bid in the event that the bid adjustment for any feature item $\ell \in f$ is set to zero by the binary variable y_h^ℓ or y_g^ℓ . By constraint sets (3.17) and (3.18), the adjustments are confined to their ranges. Constraint sets (3.19) states s_i^f has to be zero unless y_h^ℓ or y_g^ℓ is 1 for at least one feature item $\ell \in f$. Finally, constraint sets (3.20) and (3.21) ensure that if any of the adjustments for $\ell \in f$ is set to 0 by y_h^ℓ or y_g^ℓ , then the zero bid level must be selected. After solving the model, the bid adjustment for campaign h and feature item ℓ is $e^{\alpha_h^\ell}(1 - y_h^\ell)$. The bid adjustment for ad group g and feature item ℓ is $e^{\alpha_g^\ell}(1 - y_g^\ell)$. The base bid for keyword i is $\beta_i = e^{\beta_i}$. ■

The assumption that effective bids are discretized might seem restrictive at first. However, both Google and Bing operate with bids up to two decimal places (so the discretization is at the cent level). In fact, Bing⁶ explicitly states the bids are rounded down to the nearest cent after bid adjustments are applied. Using this assumption, we are able to model the BAPOA as a mixed integer program. However, it is not a practically viable solution method since the size of real-world problems renders the mixed integer program intractable to solve optimally. Suppose an advertiser wishes to adjust bids for 30 locations, 24 time intervals, and 3 device types (as in Section 3.5.3). It results in 2160 feature combinations for a single keyword. Given that the advertiser has thousands of keywords, the problem may contain millions of combinations. In addition, the number of bid levels exacerbates the intractability. Since the bids are rounded down to the nearest cent, if the threshold bid for a keyword and combination is \$10, it results in 1001 bid levels in the MIP formulation. Therefore, we use the mixed integer program as a tool to validate the quality of the Iterative Adjustment Algorithm (in small instances) as discussed in Section 3.5.2 .

3.5 Computational Results

We now assess the solution quality and the efficiency of the Iterative Adjustment Algorithm in solving the BAPOA. The computational experiments are targeted towards 1) evaluating the quality (in terms of solution produced) and the efficiency (in terms of running time) of the IAA algorithm, and 2) demonstrating the benefits

of using bid adjustments. In Section 3.5.1 describe the data generation process. In Section 3.5.2, on a set of instances with small number of keywords and feature items, we compare the revenue obtained from the IAA solution and the Flat Bid solution with the upper bound obtained from an MIP formulation (discussed in Section 3.4). In Section 3.5.3, on instances with large number of keywords and feature items, we perform solution quality and running time analysis and discuss benefits of using bid adjustments. In Section 3.5.4, we discuss how we can use bid adjustments provided by the IAA to form campaigns and ad groups, and how keywords can be assigned to ad groups and campaigns to maximize the benefit of using bid adjustments.

In the IAA, all continuous adjustments are set to 1 and all binary adjustments are set to 0 when the base bid subproblem is solved for the first time. Recall that the base bids obtained from this subproblem constitutes a solution for the case where bid adjustments are not used, i.e., the “Flat Bid” solution. To obtain a good quality Flat Bid solution, we use $|\mathcal{P}| = 51$ discrete base bid levels that uniformly span the base bid range $[0, b_i^*]$ of keyword i the first time the base bid subproblem is solved. In consequent iterations of the while loop of the IAA, we set $|\mathcal{P}| = 21$ for the base bid subproblem. In all iterations of the IAA, we use $|\mathcal{P}| = 20$ discrete adjustment levels for the adjustment subproblem. After preliminary computational experiments, we found setting $\gamma = 1$ and discounting γ by 20% at the end of each while loop iteration provides high quality solutions. In both Google and Bing, continuous adjustments are limited to values in $[0.1, 10]$. Therefore, for all feature items $\ell \in F$ we set $L_g^\ell = 0.1$ and $U_g^\ell = 10$ if adjustments for feature item ℓ can be made at the ad group level, and $L_h^\ell = 0.1$ and $U_h^\ell = 10$ if adjustments can be made at the campaign level.

The budget parameter B is set as a percentage of the maximum amount possible to spend. We calculate this amount as $\sum_{i \in K} \sum_{f \in \mathcal{F}} c_i^f(b_i^{f*})$ where b_i^{f*} is the threshold bid amount for keyword i and combination f . The IAA is implemented in C++. All computational experiments are performed on a computer with an Intel Xeon CPU E5-1620 v3 @ 3.50 GHz and 32 GB RAM running Windows 7.

3.5.1 Data Generation

As in Chapter 2, we use a sample collected from Google Keyword Planner to generate the data for our computational experiments. We created a portfolio of 400 medium-high volume keywords from various industry categories (e.g., retail, finance, technology, etc.), and collected forecast data for 20 bid amounts for each keyword. For this sample data set, we found the following linear regression model accurately predicts the number of clicks (i.e., $\psi_i(b)$) for a given bid amount (b),

$$\psi_i(b) = m_{i0} + m_{i1} \ln b.$$

We fit the model parameters m_{i0} and m_{i1} using the ordinary least squares (OLS) method for each keyword in our portfolio. The average coefficient of determination (R^2) value for the portfolio is 0.9, which indicates a strong fit. In Chart 3.2, we present the OLS estimation for keyword “refinance mortgage” with $m_{i0} = 122.46$ and $m_{i1} = 34.35$. The data suggests diminishing returns on the number of clicks as we increase the bid amount. In fact for a large enough bid amount, the ad will be displayed at the top of the page in every search query. However, any bid amount

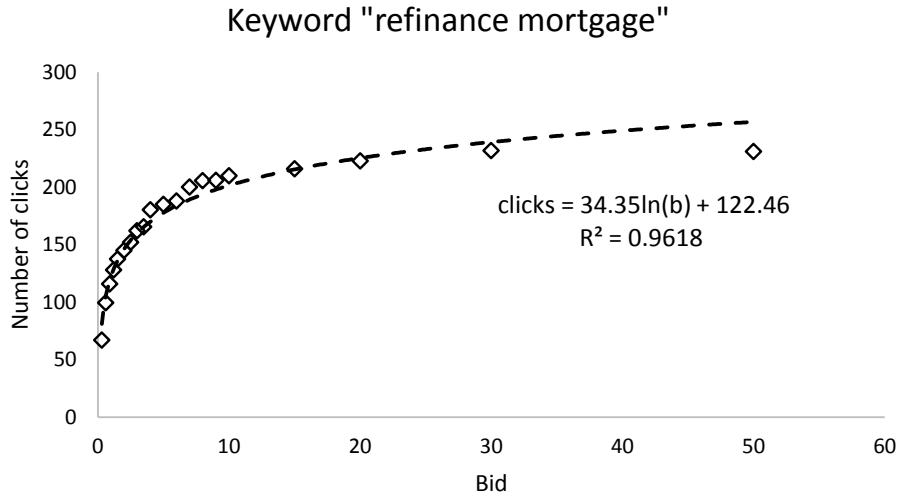


Figure 3.2: OLS estimation for $\psi_i(b) = m_{i0} + m_{i1} \ln(b)$ for keyword “refinance mortgage”.

more than that will not increase the number of clicks, which supports our earlier assumption. The limitation of using the model $\psi_i(b) = m_{i0} + m_{i1} \ln b$ to predict the number of clicks is that as b approaches 0, $\psi_i(b)$ approaches $-\infty$. To remedy that, we make the following modification,

$$\psi_i(b) = m_{i0} + m_{i1} \ln(b + e^{-m_{i0}/m_{i1}}),$$

which ensures $\psi_i(0) = 0$. Note that the correlation between m_{i0} and m_{i1} leads to $e^{-m_{i0}/m_{i1}}$ being a very small constant.

In a similar fashion, after analyzing the sampled data for cost-per-click vs bid amount, we found the following linear regression model accurately predicts the

cost-per-click (i.e., $CPC_i(b)$) for a given bid amount,

$$CPC_i(b) = m'_{i0}b + m'_{i1}b^2.$$

As in $\psi_i(b)$, we fit the model parameters m'_{i0} and m'_{i1} using the OLS method for each keyword in our portfolio. The average coefficient of determination (R^2) value for the portfolio is 0.98. In Chart 3.3, we present the OLS estimation for keyword “refinance mortgage” with $m'_{i0} = 0.6332$ and $m'_{i1} = -0.0074$. Initially, we observe an almost linear relationship between cost-per-click and bid amount. As we increase the bid amount, the competition eventually eases up and the CPC decreases slowly, and for a large enough bid, cost-per-click does not increase because of the generalized second price auction mechanism. Note that $m'_{i1} \leq 0$ for all keywords in our sample leading to this observation. We define $b_i^* = \max_{b_i} CPC_i(b_i) = \frac{-m'_{i0}}{2m'_{i1}}$ as the threshold bid. This value is found by setting the derivative $\frac{dCPC_i(b)}{db}$ to zero and solving for b . We assume for any bid b_i such that $b_i > b_i^*$, we have $CPC_i(b_i) = CPC_i(b_i^*)$ and $\psi_i(b_i) = \psi_i(b_i^*)$.

Over the sample of 400 keywords, we found (by using the distribution fitting package “fitdistrplus” in R) the parameters m_{i0} and m_{i1} follow a bivariate log-normal distribution. In a similar fashion, we found the parameters m'_{i0} and m'_{i1} follow normal and uniform distributions, respectively. To capture the correlation between parameters m'_{i0} and m'_{i1} while sampling their respective distributions, we use copulae. Copulae are widely used to generate jointly distributed random numbers from correlated univariate distributions. After experimenting with Normal,

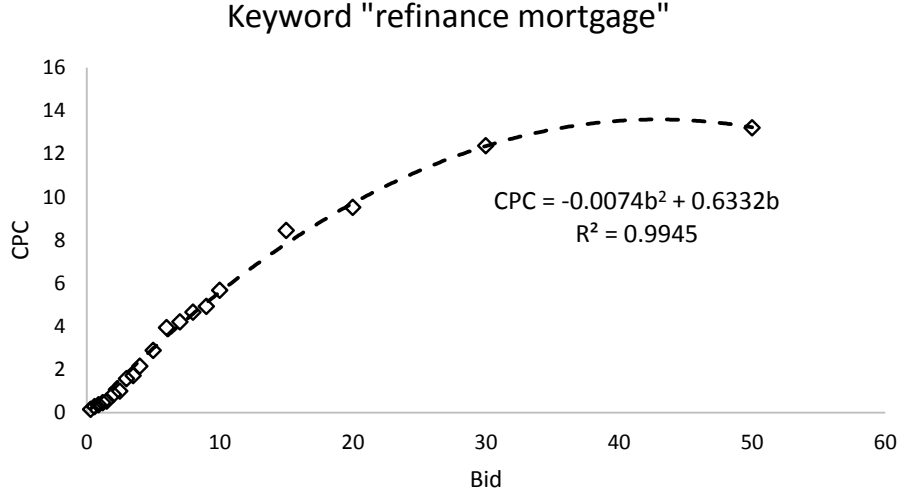


Figure 3.3: OLS estimation for $CPC_i(b) = m'_{i0}b + m'_{i1}b^2$ for keyword “refinance mortgage”.

Student’s t, Gumbel, and Clayton copulae, we observed Gumbel copulae provided the best fit for our sample and therefore used a Gumbel copula. By drawing m_{i0} , m_{i1} , m'_{i0} , and m'_{i1} values from their respective distributions for each keyword, we can generate the number of clicks function ($\psi_i(b)$) and the CPC function ($CPC_i(b)$) for a keyword i . Given $\psi_i(b)$ and $CPC_i(b)$ functions, the cost function $c_i(b)$ for a keyword can be expressed as the product of number of clicks and cost-per-click, i.e., $c_i(b) = \psi_i(b)CPC_i(b)$.

The number of clicks function $\psi_i(b)$ provides us with the aggregate number of clicks for keyword i for all feature combinations, i.e., $\psi_i(b) = \sum_{f \in \mathcal{F}} \psi_i^f(b)$. However, to create an instance for the BAPOA, we need the number of clicks function $\psi_i^f(b)$ for combination f of keyword i . In other words, we need the break down of the total number of clicks for a keyword at the feature combination level. Keyword Planner offers forecast on the fractions of clicks for location and device features.

For instance, Keyword Planner reports an estimate of 200 clicks over a day for keyword “refinance mortgage” for a bid amount of \$10. In addition, it also reports 50% of clicks will come from desktop computers and 20% of clicks will originate from DC. For a sample of 200 keywords and 10 bid amounts, we have collected data on fractions of clicks for location and device features. For location feature, we have selected 30 most populous cities in the United States. For device feature, the data provides fractions for desktop computers, tablets, and mobile devices. Keyword Planner does not report forecast on the fractions of clicks for time feature. However, we were able to obtain a typical distribution of fractions of clicks by time of day (for every hour of the day) through our industry contacts.

Using the sample collected from Keyword Planner, we observed the fractions of clicks follow a triangular distribution for both the location and the device features. By sampling their respective distributions, we can generate fractions of clicks for these features. Let p_i^ℓ denote the value generated for the fraction of clicks for keyword i and feature item $\ell \in F$. Then we define the number of clicks function $\psi_i^f(b)$ as,

$$\psi_i^f(b) = \psi_i(b) \prod_{\ell \in f} p_i^\ell.$$

For example, suppose 10% of clicks originate from DC, 5% of clicks are between 5-6 pm, 50% of clicks are on mobile devices. Then we allocate $10\% \times 5\% \times 50\% = 0.25\%$ of the clicks to originate from DC between 5-6 pm on mobile devices. Therefore, $\psi_i^f(b) = 0.25\% \times \psi_i(b)$ where f corresponds to (DC, 5-6pm, mobile) combination.

The cost function $c_i(b) = \psi(b)CPC_i(b)$ provides us with the aggregate cost for

keyword i for all feature combinations, i.e., $c_i(b) = \sum_{f \in \mathcal{F}} c_i^f(b)$. At a disaggregated level, the cost function for keyword i and combination f is represented as $c_i^f(b) = \psi_i^f(b)CPC_i^f(b)$ where $CPC_i^f(b)$ denotes the function of cost-per-click for keyword i and feature combination f . When we analyzed data collected on Keyword Planner, we observed the difference between the fractions of clicks and cost across feature items (specifically location and device) to be negligible. For instance, if 10% of total clicks originate from DC, then DC accounts for roughly 10% of the total cost. This suggests the $CPC_i^f(b)$ function to be (presently) invariant to feature combinations, i.e., $CPC_i^f(b) = CPC_i(b)$. In addition, we set $b_i^{f*} = b_i^*$ as the threshold bid amount for keyword i and combination f .

We define the revenue function for a feature combination as the product of number of clicks and revenue generated per click from that combination. In other words, the revenue function for keyword i and feature combination f is, $r_i^f(b) = \psi_i(b)RPC_i^f$ where RPC_i^f denotes the revenue-per-click for keyword i and feature combination f . In our computational experiments, we aim to evaluate the benefits of using bid adjustments under different scenarios. To that end, we create revenue-per-click values for keyword i and feature combination f as follows.

$$RPC_i^f = \overline{CPC}_i \prod_{\ell \in f} \varphi_{hg}^\ell \omega_i^\ell \delta_i^{\ell, f}$$

The first component of RPC_i^f , \overline{CPC}_i , denotes the average cost-per-click for keyword i which is $\overline{CPC}_i = CPC_i(b_i^*/2)$. Suppose keyword i is assigned to campaign h and ad group g , then the second component, φ_{hg}^ℓ , denotes the random variation

generated at the campaign and ad group level for feature item ℓ . We define φ_{hg}^ℓ as $\varphi_{hg}^\ell = 10^{U[\log_{10} \frac{1}{\nu}, \log_{10} \nu]}$ where $U[a, b]$ denotes a draw from the continuous uniform distribution in the range $[a, b]$. We use the parameter ν to express the range of random variation, as ν gets larger, φ_{hg}^ℓ will be drawn from a larger range causing the revenue-per-click to vary more and more across feature items. The way we generate φ_{hg}^ℓ ensures the random draws are centered around 1 with roughly half of them below, and half above. In essence, this component allows us to capture cases where some feature items have more revenue potential than others. If adjustment $\ell \in f$ can only be made at the campaign level, then we drop the subscript g ; and all keywords assigned to campaign h share the same φ_h^ℓ value for feature item ℓ . If adjustment $\ell \in f$ is made at the ad group level, then all keywords assigned to campaign h and ad group g share the same φ_{hg}^ℓ value for feature item ℓ .

The third component, ω_i^ℓ , denotes the random noise generated at the keyword level for each feature item defined as $\omega_i^\ell = U[1 - \rho, 1 + \rho]$. We use the parameter ρ to express the range of random noise, as ρ gets larger, the revenue-per-click varies more and more across the keywords in the same ad group (and campaign) for the same feature item. Finally, the fourth component, $\delta_i^{\ell, f}$, denotes the random noise generated at the feature combination level for each keyword and feature item defined as $\delta_i^{\ell, f} = U[1 - \varepsilon, 1 + \varepsilon]$. We use the parameter ε to express the range of random noise, as ε gets larger, the revenue-per-click varies more and more across feature combinations for the same feature item.

For instance, if $\nu = 1$, $\rho = 0$, and $\varepsilon = 0$, all feature combinations f of a keyword have the same revenue-per-click without any variation or noise, which is

equal to the average cost-per-click, i.e., $RPC_i^f = \overline{CPC}_i$. When $\nu > 1$, $\rho = 0$, and $\varepsilon = 0$, the revenue-per-click for each feature combination will be the product of the average cost-per-click and φ_{hg}^ℓ . Suppose combination f is (DC, 5-6pm, mobile) and combination f' is (NY, 5-6pm, mobile), if $\varphi_{hg}^{DC} = 2$ and $\varphi_{hg}^{NY} = 1$, then $RPC_i^f = 2 \times RPC_i^{f'}$ for every keyword in campaign h and ad group g . When $\nu > 1$, $\rho > 0$, and $\varepsilon = 0$, we add some noise at the keyword level. Finally, when $\nu > 1$, $\rho > 0$, and $\varepsilon > 0$, we add some noise at the keyword and feature combination level. With these parameters, RPC_i^f may not be exactly twice as much as $RPC_i^{f'}$ in the above example. In fact, depending on the choice of ρ and ε , the ratio of RPC_i^f to $RPC_i^{f'}$ may vary significantly across keywords and feature combinations.

3.5.2 MIP Instances

We first assess the quality of solutions obtained from the IAA by comparing it against instances of the BAPOA where it is possible to obtain high quality upper bounds. Discretizing the domain of the effective bids naturally leads to a mixed integer programming model that can only be solved for tiny instances. This MIP model is discussed in Section 3.4. We call these tiny instances MIP Instances. We solve these MIP instances using CPLEX 12.71 implemented in C++. However, due to excessive computational times, we have limited CPLEX runs to one hour and use the best available upper bound at the time of termination. We generated 5 MIP instances each with 5 keywords (in MIP instances, every keyword is assigned to a different campaign, therefore, there are 5 campaigns for 5 keywords), 4 location, 4

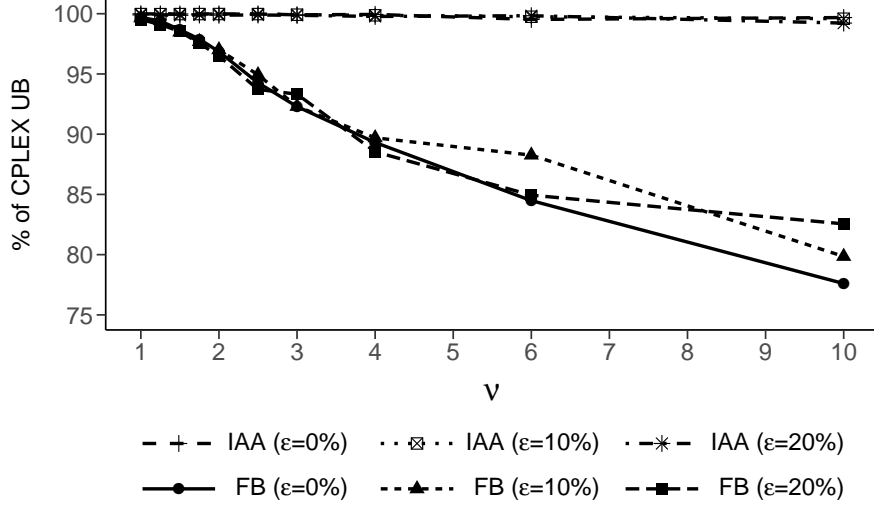


Figure 3.4: IAA and Flat Bid revenues as percentages of the CPLEX upper bound for $\varepsilon = \{0\%, 10\%, 20\%\}$ and budget set to 5%.

time of day, and 3 device feature items. For each instance, we generated the functions for number of clicks $\psi_i^f(b)$ and cost-per-click $CPC_i^f(b)$ for each keyword and feature combination. In addition, we varied the RPC_i^f value by generating it based on the combination of $\nu = \{1, 1.25, 1.5, 1.75, 2, 2.5, 3, 4, 6, 10\}$ and $\varepsilon = \{0\%, 10\%, 20\%\}$ parameters. Since we assign every keyword to a different campaign, we set $\rho = 0$, i.e., do not introduce any random noise at the keyword level. To isolate the effect of ν and ε , we use the same $\psi_i^f(b)$ and $CPC_i^f(b)$ for keyword i and combination f for varying RPC_i^f . Therefore, each $\psi_i^f(b)$ and $CPC_i^f(b)$ is used with $10 \times 3 = 30$ different RPC_i^f values. There are effectively $5 \times 30 = 150$ instances. We set the budget to $\{5\%, 10\%, 15\%, 20\%\}$ of the maximum amount possible to spend. Henceforth, we will refer to the percentage as the budget amount, e.g., $B = 10\%$ implies the budget is set to 10% of the maximum amount possible to spend.

In Charts 3.4-3.7, we report the revenue obtained from the IAA solution (i.e.,

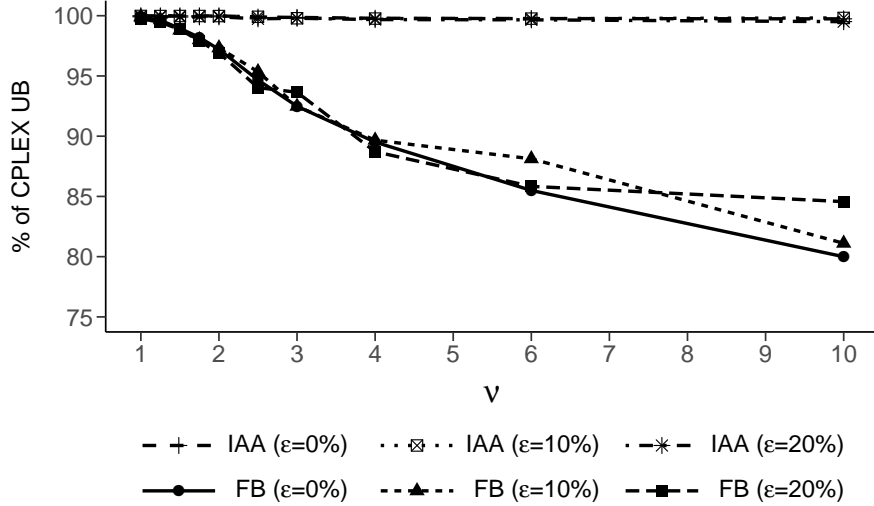


Figure 3.5: IAA and Flat Bid revenues as percentages of the CPLEX upper bound for $\epsilon = \{0\%, 10\%, 20\%\}$ and budget set to 10%.

IAA revenue) and revenue obtained from the Flat Bid solution (i.e., Flat Bid revenue) as a percentage of the upper bound obtained via CPLEX (i.e., CPLEX UB) where the percentages are averaged over 5 instances for each ν - ϵ combination. For example, for $B = 10\%$, $\nu = 3$, and $\epsilon = 10\%$, the IAA revenue is 99.8% of the CPLEX UB whereas the Flat Bid revenue is 93.8% of the CPLEX UB. In other words, the revenue lift⁷ is 6.4%. In Charts 3.4-3.7, we observe that the IAA provides near optimal solutions regardless of ν , ϵ , or budget, which demonstrates approximating the subproblems by discretizing the domains of base bids and bid adjustments does not jeopardize the solution quality. We also observe that as ν increases, the revenue lift increases as well, which implies the increased variation in the revenue-per-click across features is captured by bid adjustments in the IAA. The Flat Bid solution,

⁷We use “revenue lift” as a measure of revenue benefit obtained from bid adjustments against an environment without bid adjustments, i.e. Flat Bid. The revenue lift is defined as the IAA Revenue minus the Flat Bid revenue divided by the Flat Bid Revenue.

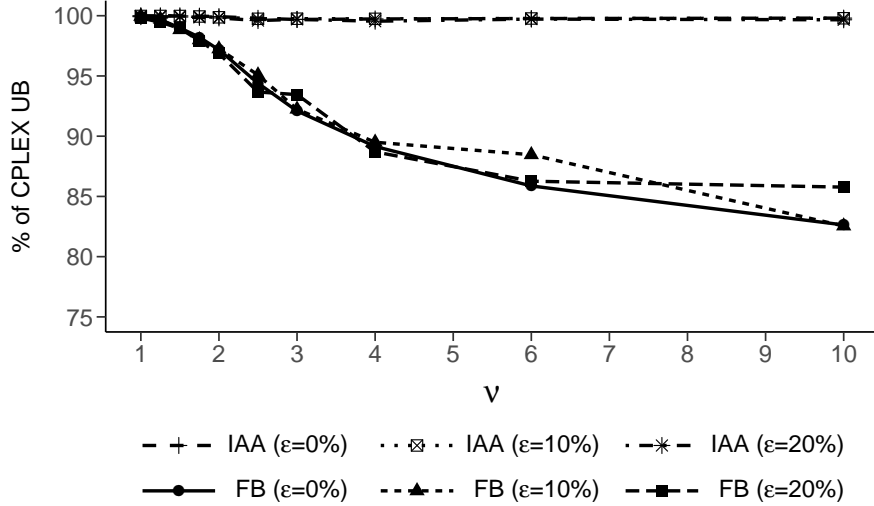


Figure 3.6: IAA and Flat Bid revenues as percentages of the CPLEX upper bound for $\epsilon = \{0\%, 10\%, 20\%\}$ and budget set to 15%.

on the other hand, does not use bid adjustments and the increased revenue-per-click variation causes the relative revenue of the Flat Bid solution to deteriorate. Even with random noise added on the feature combination level, the IAA still finds adjustments that yield a near optimal solution. The random noise parameter ϵ seems to cause significant differences in the revenue lift. However, the effect of ϵ is not consistent. This is due to the size of the MIP instances leading to small samples of RPC_i^f values. The computational experiments with larger instances demonstrate ϵ having negligible effect on the amount of revenue lift.

In Figure 3.8, we observe using bid adjustments provide slightly less revenue lift as budget increases for larger ν . For $\nu \leq 4$, the revenue lift under different budget amounts are virtually the same. However, when $\nu = 10$, the revenue lift is approximately 9% less when $B = 20\%$ compared to $B = 5\%$. As budget increases, we expect the Flat Bid revenue getting closer and closer to the upper bound. At

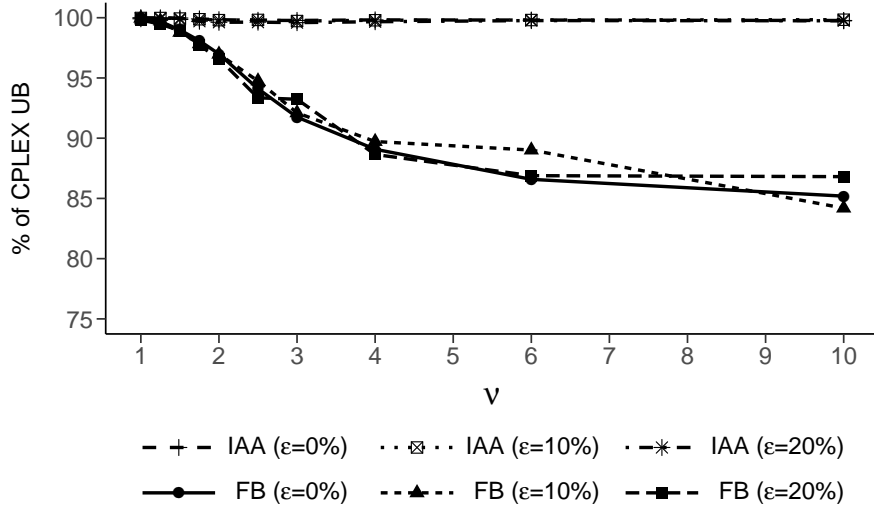


Figure 3.7: IAA and Flat Bid revenues as percentages of the CPLEX upper bound for $\epsilon = \{0\%, 10\%, 20\%\}$ and budget set to 20%.

100% budget, the Flat Bid solution will be the optimal solution since it is optimal to bid the threshold bid for every combination of every keyword. In general, we observe the IAA finds near optimal solutions regardless of the level of revenue-per-click variation, random noise, and budget amount. We also observe the revenue lift from bid adjustments increases as the revenue-per-click variation increases and the budget decreases.

The MIP instances we generated represent the largest BAPOA instances for which CPLEX can provide high quality upper bounds after one hour of computations. In contrast, the average running time of the IAA is only 0.02 seconds per instance for MIP instances.

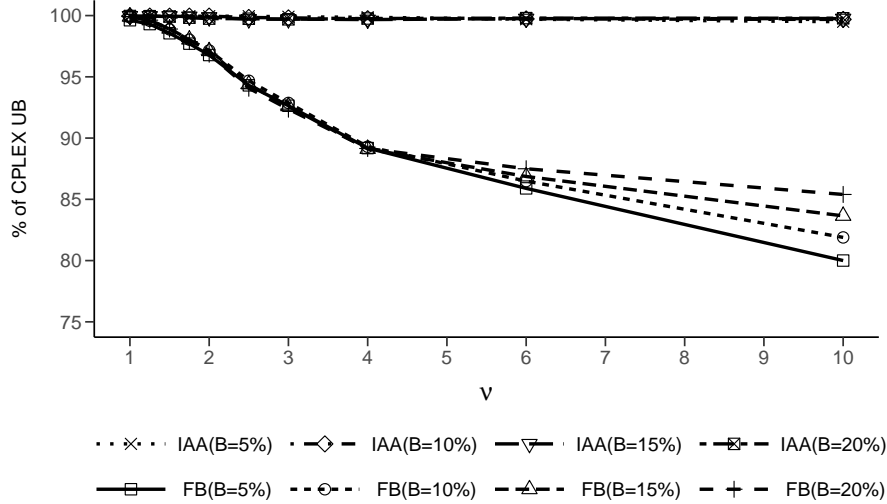


Figure 3.8: IAA and Flat Bid revenues as percentages of the CPLEX upper bound for $B = \{5\%, 10\%, 15\%, 20\%\}$. The results are averaged over $\varepsilon = \{0\%, 10\%, 20\%\}$.

3.5.3 Online Advertising Instances

The MIP instances demonstrate the quality of the IAA solution compared to the Flat Bid solution and the upper bound obtained from CPLEX. To test the efficiency and quality of the IAA in industry settings, we experiment on instances with a large number of keywords and feature items. We first focus on instances (as in the MIP instances) where every keyword is assigned to a different campaign (there are the same number of campaigns as keywords) as we will later show this approach allows for an effective way of constructing campaigns and ad groups based on the values of the bid adjustments for different features. We present results for 1,000, 5,000, 10,000, and 25,000 keyword instances. For each keyword set, the data generation setting is identical to Section 3.5.2 (five sets of instances, same ranges for ν and ε resulting in 150 instances, $\rho = 0$ since every keyword is assigned to a

different campaign). However unless otherwise stated, each instance has 30 location (corresponding to most populous cities in the US), 24 time of day (for each hour interval of the day), and 3 device feature items. In Table 3.3, we report the Flat Bid revenue as a percentage of IAA revenue where the budget is set to 10%. The percentages are averaged over 5 instances for each ν - ϵ combination. As in the MIP instances, we observe significant increase in revenue lift (note that when Flat Bid revenue decreases as a percentage of the IAA revenue, the revenue lift increases) as ν increases. As we stated in Section 3.5.2, we observe ϵ having virtually no effect on the revenue lift. In addition, the increase in the number of keywords also have no effect on revenue lift.

To further investigate the effect of budget on the revenue lift observed in the MIP instances, we experimented with 1000 keyword instances for $B = \{5\%, 10\%, 15\%, 20\%\}$. In Figure 3.9, we report the Flat Bid revenue as a percentage of the IAA revenue where the results are averaged over $\epsilon = \{0\%, 10\%, 20\%\}$. The findings are consistent with those of the MIP instances, as budget decreases and ν increases, the revenue lift from bid adjustments increases.

The advertisers might be tempted to reduce the number of feature items in an attempt to simplify their problem of determining bid adjustments. For instance, an advertiser might group tablets and mobile devices together, in essence treating them as one feature item and determining one adjustment value per campaign (or ad group if adjustments can be made at the ad group level) that would apply to all feature combinations containing tablets or mobile devices in that campaign (or ad group). However, if there is revenue variation across tablets and mobile devices that

ε	ν	1000	5000	10000	25000	Average
0%	1	99.94%	99.94%	99.94%	99.94%	99.94%
	1.25	99.60%	99.59%	99.59%	99.59%	99.59%
	1.5	98.79%	98.78%	98.78%	98.77%	98.78%
	1.75	97.80%	97.76%	97.75%	97.76%	97.77%
	2	96.66%	96.65%	96.64%	96.64%	96.65%
	2.5	94.41%	94.34%	94.35%	94.36%	94.37%
	3	92.24%	92.16%	92.20%	92.16%	92.19%
	4	88.83%	88.58%	88.52%	88.55%	88.62%
	6	84.06%	84.01%	84.08%	83.97%	84.03%
10	80.10%	80.04%	80.11%	80.10%	80.09%	
10%	1	99.94%	99.95%	99.94%	99.94%	99.94%
	1.25	99.60%	99.59%	99.59%	99.59%	99.59%
	1.5	98.77%	98.77%	98.77%	98.77%	98.77%
	1.75	97.75%	97.75%	97.75%	97.75%	97.75%
	2	96.64%	96.62%	96.65%	96.64%	96.64%
	2.5	94.40%	94.36%	94.33%	94.35%	94.36%
	3	92.16%	92.09%	92.16%	92.16%	92.14%
	4	88.50%	88.47%	88.54%	88.52%	88.50%
	6	83.93%	84.05%	84.00%	84.00%	83.99%
10	80.33%	80.04%	80.11%	80.10%	80.14%	
20%	1	99.94%	99.94%	99.94%	99.94%	99.94%
	1.25	99.58%	99.58%	99.58%	99.58%	99.58%
	1.5	98.77%	98.76%	98.76%	98.76%	98.76%
	1.75	97.73%	97.74%	97.74%	97.74%	97.74%
	2	96.64%	96.64%	96.62%	96.63%	96.64%
	2.5	94.39%	94.33%	94.34%	94.33%	94.35%
	3	92.16%	92.13%	92.17%	92.15%	92.15%
	4	88.43%	88.50%	88.48%	88.53%	88.48%
	6	84.21%	83.87%	84.00%	84.00%	84.02%
10	80.24%	80.06%	80.19%	80.19%	80.17%	

Table 3.3: The Flat Bid revenue as a percentage of the IAA revenue for $K = \{1000, 5000, 10000, 25000\}$. The budget is fixed to 10%

cannot be captured by the same bid adjustment value, then grouping them together has a significant effect on the revenue lift. We demonstrate this behavior on 1,000 keyword instances where the number of feature items for the time of day (TOD) feature is set to $\{3, 6, 12, 24\}$ while the number of location and device feature items are left at 30 and 3 respectively. Note that we do not alter the original data where the instances were generated with 24 hours in a day. We merely reduce the number of bid adjustments the IAA can determine by grouping the hours such that there

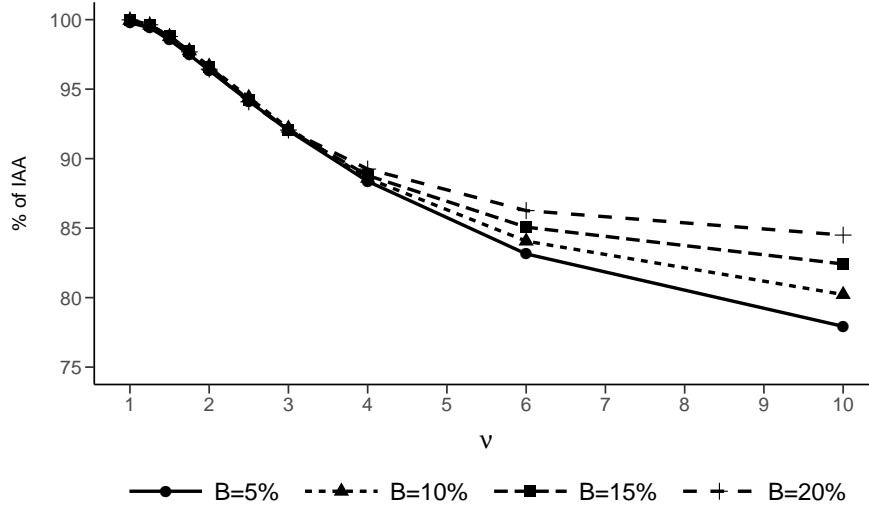


Figure 3.9: The Flat Bid revenue as a percentage of the IAA revenue for 1000 keywords and $\{5\%, 10\%, 15\%, 20\%\}$ budget. The results are averaged over $\varepsilon = \{0\%, 10\%, 20\%\}$.

are $\{8, 4, 2, 1\}$ hours in each grouping. In Chart 3.10, we report the Flat Bid revenue as a percentage of the IAA revenue where the percentages are averaged over $\varepsilon = \{0\%, 10\%, 20\%\}$. We observe the revenue lift from bid adjustments decreases with the number of feature items for the time of day feature. Recall, due to the procedure we used to generate the data, there is revenue variation (increases with ν) across hours within a grouping. However, since only one bid adjustment is determined per grouping, the revenue variation cannot be fully captured for every hour in the grouping. In addition, the more hours we pack within a grouping, the less effective the sole bid adjustment is at capturing the variation. The decline of revenue lift gets even worse as ν increases, when there are 3 vs 24 time of day feature items, the revenue lift difference gets as high as 5.5%. Therefore, unless the feature items do not have revenue variations that can be captured by the same bid adjustment

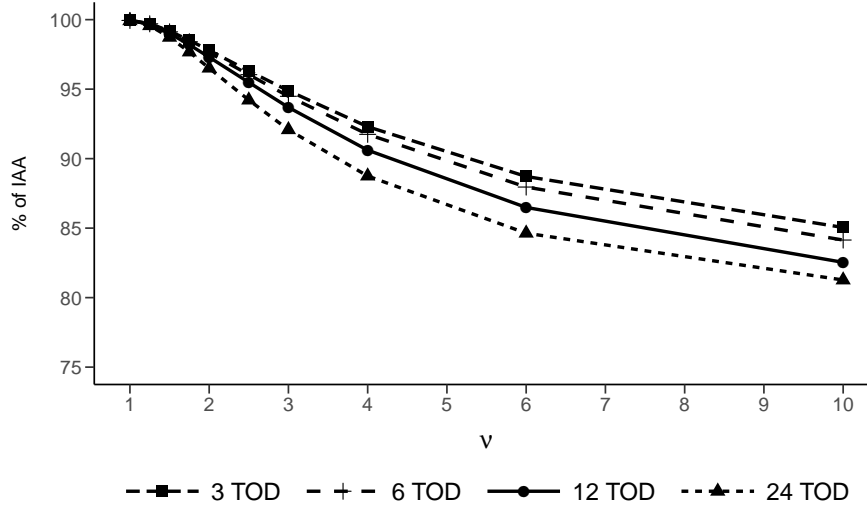


Figure 3.10: The Flat Bid revenue as a percentage of the IAA revenue for 1000 keywords and $\{3, 6, 12, 24\}$ number of times of day. The results are averaged over $\varepsilon = \{0\%, 10\%, 20\%\}$.

value, they should not be grouped together.

In Charts 3.11 and 3.12, the running time of the IAA per 1000 keywords is reported in seconds for the data set generated for Table 3.3. In Chart 3.11, we report the running time vs the number keywords for $\varepsilon = \{0\%, 10\%, 20\%\}$ values. We observe that the running time of the IAA shows near linear growth as we increase the number of keywords. In addition, ε has virtually no effect on the running time of the algorithm. A near linear running time is of high practical importance since it shows the IAA can scale up to handle very large portfolios. In Chart 3.12, we report the running time vs ν for $\varepsilon = \{0\%, 10\%, 20\%\}$ where the results are averaged over 1,000, 5,000, 1,000, and 25,000 keyword sets. We once again observe ε does not effect the running time of the IAA. However, for $\nu \geq 3$, the running time increases. Recall that the IAA algorithm executes the while loop until the revenue

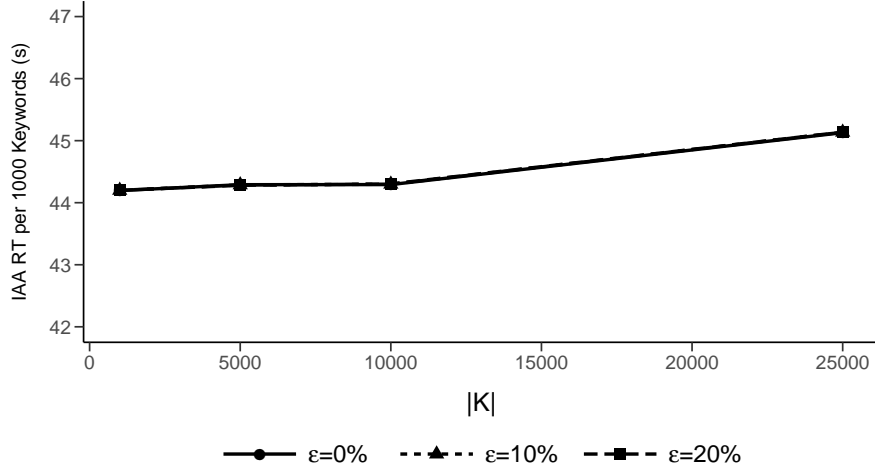


Figure 3.11: The running time of the IAA per 1000 keywords vs $|K|$

converges. When $\nu < 3$, the IAA takes (most of the time) 2 iterations to converge. In contrast, when $\nu \geq 3$, the algorithm takes (most of the time) 3 iterations to converge. Therefore, the overall running time of the IAA increases with ν . When ν is large, the bid adjustments are further away from 1 (i.e., the initial value of bid adjustments) when the algorithm terminates, which requires more fine tuning and results in more iterations.

In Chart 3.13, we report the running time of the IAA per while loop iteration (since the number of iterations vary with ν , we report the running time per iteration) per 1 million feature combinations for the set of 1000 keyword instances. For example, if an instance has 1,000 keywords, 30 location, 24 time of day, and 3 device feature items, then there are $1,000 \times 30 \times 24 \times 3 = 2,160,000$, feature combinations in the instance. In Chart 3.14, we report the total running time of the subproblems (the running time of the MCKP) in the IAA per while loop iteration per 1 million feature items for the set of 1000 keyword instances. For example, if an instance has

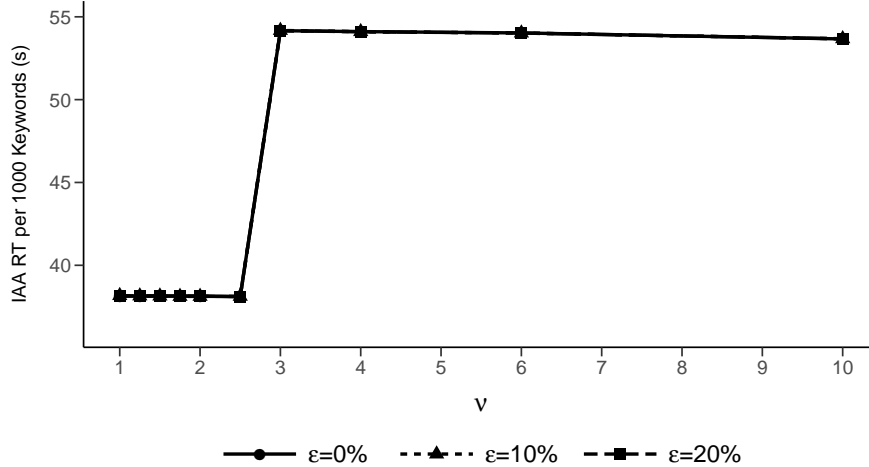


Figure 3.12: The running time of the IAA per 1000 keywords vs ν

1,000 keywords, 30 location, 24 time of day, and 3 device feature items, then there are $1,000 \times (30 + 24 + 3) = 57,000$ feature items in the instance. In both charts, the running times are averaged over ε and ν values. In Chart 3.13, we observe the overall running time of the IAA is slightly sublinear in the number of combinations in the instance. The main reason for this behavior is when there are more feature combinations, the computational overhead is distributed over more combinations (recall that we report the running time per 1 million feature combinations). However, in Chart 3.14, we observe the running time of the subproblems in the IAA is proportional to the number of feature items in the instance. The MCKP corresponding to the base bid subproblem has one class for every keyword. The MCKP corresponding to the feature adjustment subproblem has one class for every campaign (or ad group) and feature item. For instance, suppose location adjustments can be made at the campaign level, if there are 1,000 campaigns and 30 location feature items, then the number of classes in the feature adjustment subproblem for location feature is

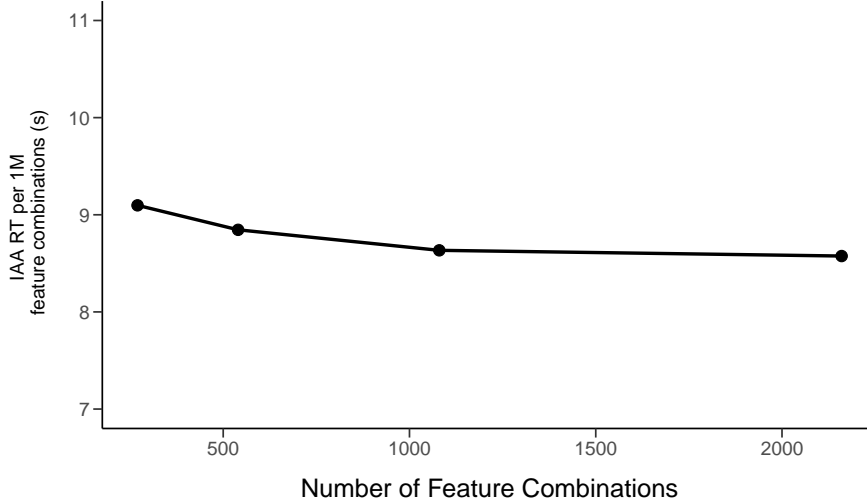


Figure 3.13: The running time of the IAA per 1M feature combinations vs the number of feature combinations.

$30 \times 1,000 = 30,000$. Since the column generation procedure shows near linear time performance in computational experiments (discussed in Chapter 2), the running time of the subproblems is proportional to the number of feature items.

The total running time of the subproblems constitutes roughly 1% of the total running time of the IAA. The remainder of the time, the algorithm sets up the data for subproblems by performing look-ups for the expected cost and revenue for each discrete base bid and adjustment level. Suppose the data for the base bid subproblem is being set up, then for every base bid level β_{ij} , we need to calculate the resulting effective bid b_{ij}^f for keyword i and feature combination f . Based on the effective bid, we evaluate $c_i^f(b_{ij}^f)$ and $r_i^f(b_{ij}^f)$ and set $c(\beta_{ij}) = \sum_{f \in \mathcal{F}} c_i^f(b_{ij}^f)$ and $r(\beta_{ij}) = \sum_{f \in \mathcal{F}} r_i^f(b_{ij}^f)$. In other words, setting up the data for a subproblem involves calculating the resulting effective bid for every keyword and feature combination and evaluating the cost and revenue functions. Therefore, the set up time is linear in the

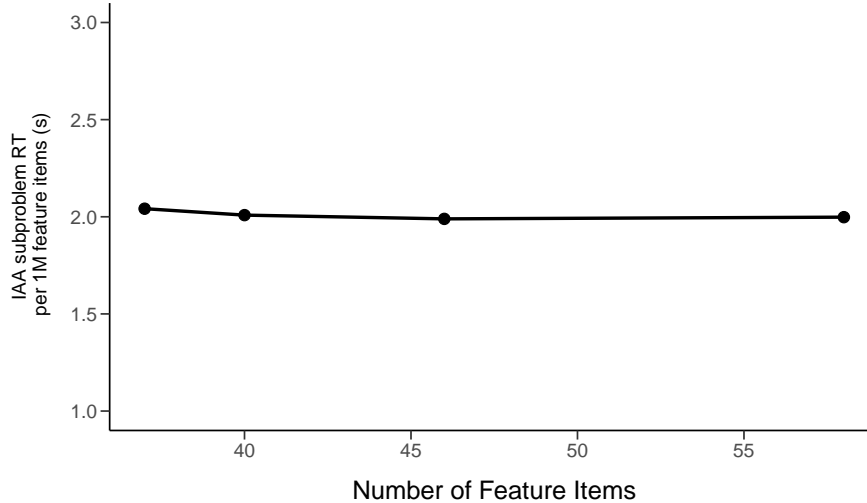


Figure 3.14: The total running time of the subproblems in IAA per 1M feature items vs the number of feature items.

number of look-ups performed, i.e., the number of keywords and feature combinations. Since there are many more feature combinations than there are feature items (e.g., 2160 feature combinations for 57 feature items), the set up time constitutes 99% of the running time of the IAA. Fortunately, the set up process is highly parallelizable. By parallelizing the set up process, it would be possible to speed up the IAA significantly, which would allow us to tackle problems with massive number of keywords and feature items.

3.5.4 Creating Advertising Campaigns

When we model the BAPOA, we make the implicit assumption that the advertiser has already created campaigns and ad groups, and determined the ad group and campaign of each keyword. We now show that the BAPOA model can help in creating campaigns and ad groups, and assigning keywords to campaigns and ad

groups. Recall that the adjustments are determined for campaigns and ad groups whereas the base bids are determined for each keyword. To maximize revenue, we would place each keyword in a campaign by itself with only one ad group (as is the case in Sections 3.5.2 and 3.5.3), which allows us to effectively adjust bids at the keyword level. However, the advertising platforms (both Google and Bing) impose limitations on the number of campaigns and ad groups that can be created. Therefore, we aim to create campaigns and ad groups such that the total expected revenue is as close as possible to the ideal case where adjustments are determined at the keyword level. To that end, we suggest the following procedure.

Step 1: place each keyword in a campaign by itself with one ad group and execute the IAA

Step 2: cluster keywords based on the bid adjustments (that can be made at the campaign level) provided by the IAA in Step 1

Step 3: create campaigns based on the clusters determined in Step 2

Step 4: cluster keywords in each campaign based on the bid adjustments (that can be made at the ad group level) provided by the IAA in Step 1

Step 5: create ad groups in each campaign based on the clusters determined in Step 4 and execute the IAA

In Step 1, we relax the limitation on the number of campaigns and ad groups and solve the BAPOA where each keyword is in its own campaign. The adjustments obtained in this step represent the best case scenario for each keyword. In Step 2,

we form clusters of keywords such that keywords with similar adjustments (only by using features where adjustments can be made at the campaign level) are in the same cluster. In Step 3, we use the clusters formed in Step 2 to create the campaigns. In Step 4, for each campaign, we form clusters of keywords such that keywords with similar adjustments (only by using features where adjustments can be made at the ad group level) are in the same cluster. In Step 5, we create the campaigns and ad groups based on clusters determined in Steps 2 and 4 and solve the BAPOA. At the termination of the procedure, the advertiser will have determined campaign and ad group assignments, base bids for each keyword as well as the bid adjustments for each campaign and ad group. For example, suppose location and time adjustments can be made at the campaign level whereas device adjustments can be made at the ad group level. Then we ignore device adjustments while forming clusters in Step 2 and ignore location and time adjustments while forming clusters in Step 4.

In Google Adwords, the advertiser is allowed to create at most 10,000 campaigns per account and at most 20,000 ad groups per campaign. If the number of keywords is less than 10,000, then it is advantageous to cluster each keyword to a separate campaign with one ad group. However, if there are more than 10,000 keywords, then after clustering keywords to campaigns (in Step 2) based on adjustments that can be made at the campaign level, it is optimal (in terms of maximizing revenue) to cluster each keyword to a separate ad group in the campaign as long as the number of keywords clustered to that campaign does not exceed 20,000. From a practical standpoint, it is highly unlikely that a campaign will contain more than 20,000 keywords after Step 2 of the procedure. We experiment on 5 sets of in-

stances with 10,000 keywords randomly assigned to 100 campaigns. Each instance has 30 location, 24 time of day, and 3 device feature items as in Section 3.5.3 and we assume adjustments can only be made at the campaign level for all three features. When keywords are assigned to the same campaign, it means they share the same random variation φ_h^ℓ (we drop subscript g from φ_{hg}^ℓ since adjustments can be made at the campaign level for all features) whereas the random noise ω_i^ℓ is added at the keyword level. To demonstrate the effect of ν parameter (which is used in φ_h^ℓ) in conjunction with ρ parameter (which is used in ω_i^ℓ), we experiment with $\nu = \{1, 1.25, 1.5, 1.75, 2, 2.5, 3, 4, 6, 10\}$, $\rho = \{0\%, 20\%, 40\%\}$. In addition we also experiment with $\varepsilon = \{0\%, 10\%, 20\%\}$. We set the budget percentage parameter to $B = 10\%$.

There are several unsupervised learning algorithms that can be used to cluster keywords. We have used an off-the-shelf implementation of “agglomerative clustering” provided in Python’s *scikit-learn* library. Agglomerative clustering is a type of hierarchical clustering algorithm where each item starts in its own cluster and clusters are merged to create larger clusters until a desired number of clusters is reached. After experimenting with different distance metrics and linkage criteria, we have chosen Euclidean distance metric and Ward linkage criteria as it provided the best performance. In Charts 3.15 - 3.17, we provide the expected revenue obtained from the IAA-Agglo and IAA-Omni as a percentage of the expected revenue from IAA-Individual where the percentages are averaged over 5 instances for each ν - ρ - ε combination. IAA-Individual refers to the solution of the IAA where each keyword is placed in a campaign by itself with one ad group, i.e., solution obtained

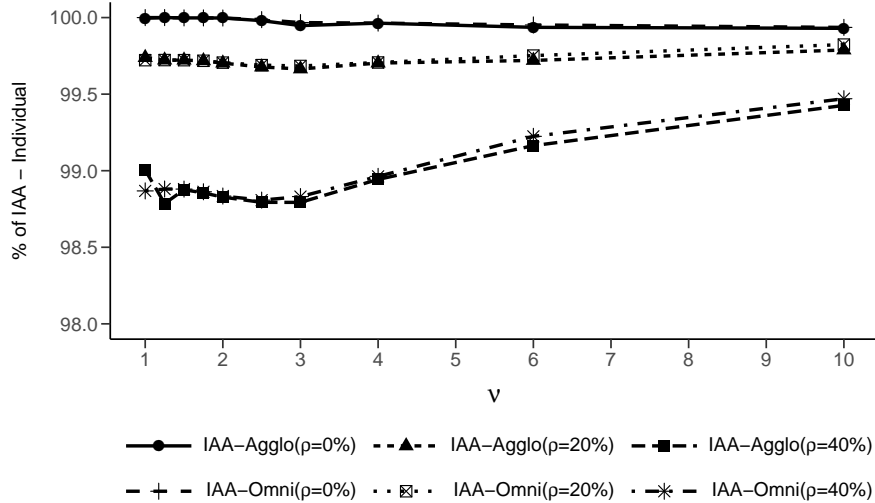


Figure 3.15: The IAA-Agglo and IAA-Omni revenue as a percentage of the IAA-Individual revenue for $\varepsilon = 0\%$

in Step 1 of the procedure. IAA-Agglo refers to the solution of the IAA executed after Step 3 where campaigns are created using the agglomerative clustering in Step 2 of the procedure. Since adjustments for all three features can be made at the campaign level, we skip Steps 4 and 5 of the procedure and execute the IAA after Step 3. IAA-Omni refers to the solution of the IAA where the actual campaign assignments from data generation are used. In other words, IAA-Omni executes IAA for an “omniscient” advertiser who has perfect information of the underlying campaign assignments of the data.

First, we observe creating campaigns with agglomerative clustering provides a solution very close to campaigns created by an omniscient advertiser. This implies agglomerative clustering can, for the most part, capture the underlying campaign structure in the data. We also observe the effect of ε to be negligible on the relative revenue of IAA-Agglo and IAA-Omni for varying ρ and ν . It is clear that as we

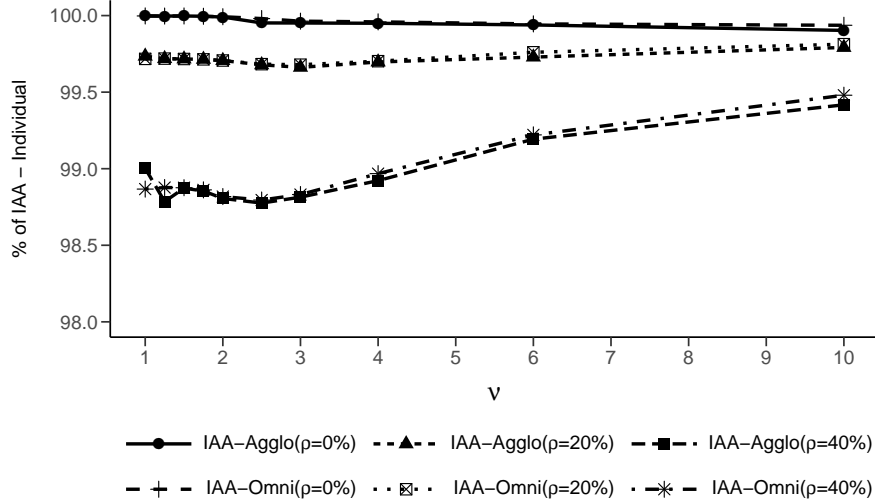


Figure 3.16: The IAA-Agglo and IAA-Omni revenue as a percentage of the IAA-Individual revenue for $\varepsilon = 10\%$

increase ρ , expected revenues of IAA-Agglo and IAA-Omni deteriorate relative to IAA-Individual. Even though keywords in the same campaign have the same φ_h^ℓ , the random noise added on the keyword level (increases with ρ) causes the expected revenue to decline. Intuitively, keywords in the same campaign are indistinguishable (in terms of revenue-cost trade-off) across features for small ρ . Therefore bid adjustments determined at the campaign level tends to work well for all keywords in the campaign. Note that any other difference at the keyword level can be captured by base bids as long as keywords in the same campaign are similar across features. In this case, bid adjustments determined at the campaign level work almost as well as adjustments determined at the keyword level. However, as we increase ρ , keywords become more and more distinguishable across features due to the random noise added via ω_i^ℓ . In other words, adjustments determined at the campaign level do not work well for every keyword in the campaign since different keywords have different

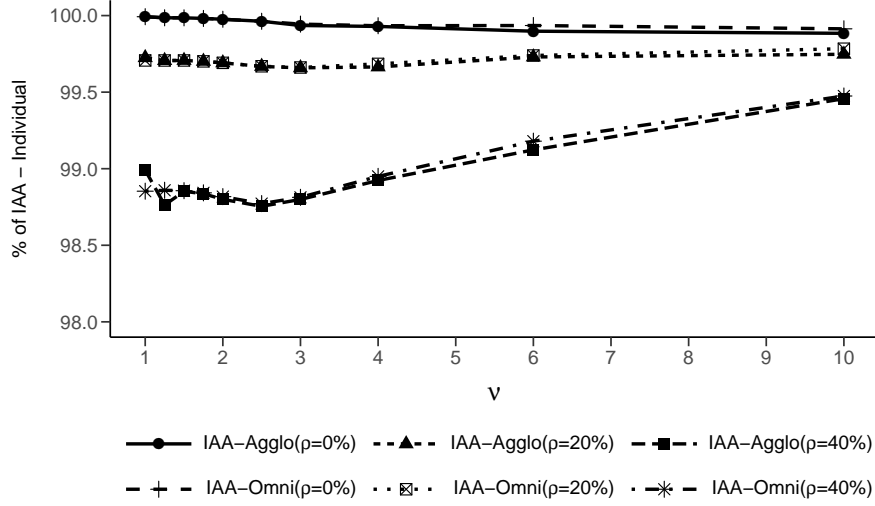


Figure 3.17: The IAA-Agglo and IAA-Omni revenue as a percentage of the IAA-Individual revenue for $\varepsilon = 20\%$

revenue-cost trade-offs across features. In this case, setting adjustments at the keyword level create significant revenue increase because adjustments can capture the revenue-cost trade-off across features separately for each keyword. It is important to note that even when $\rho = 40\%$, which results in significant random noise, the revenue loss compared to IAA-Individual is only around 1%. For large ν , the revenue loss is even smaller since the random variation (φ_h^ℓ) dominates the random noise (ω_i^ℓ). While the IAA-Individual revenue is not attainable in practice due to limitations on the number of campaigns, the IAA-Agglo revenue is only slightly less than the IAA-Individual revenue and still almost as good as an omniscient advertiser can have.

The running time of this procedure depends on the clustering algorithm used. The running time of IAA-Agglo is nearly the same as IAA-Individual. Even though there are fewer adjustment variables in IAA-Agglo resulting in smaller subproblems,

the number of look-ups performed are the same. If the look-ups are parallelized as suggested in Section 3.5.3, then IAA-Agglo would run faster than the IAA-Individual due to smaller subproblems. We demonstrate the effectiveness of agglomerative clustering in terms of solution quality. However, agglomerative clustering may not scale well for very large problems with large number of clusters. In this case, k -means maybe a better suited option. k -means is one of the most popular clustering algorithms in existence. Unlike agglomerative clustering, k -means clustering can scale up to handle very large instances. We did experiment with an off-the-shelf implementation of k -means clustering provided in Python's *scikit-learn* library. In our experiments, the expected revenue of the IAA executed based on k -means clustered campaigns is at most 0.24% less than the expected revenue of IAA-Agglo. Therefore, when agglomerative clustering is too computationally expensive, k -means clustering provides a viable alternative.

Chapter 4: The Capacitated Mobile Facility Location Problem

In this chapter, we study the capacitated mobile facility location problem (CMFLP). The CMFLP is defined on a network where clients and facilities are initially located at vertices on the network. Associated with each client is a demand and each facility has a specified capacity available to service demand. A destination vertex must be determined for each facility and each client should be assigned to one of the facilities so that the total demand of the clients assigned to a facility respects the capacity. The objective is to minimize the total weighted distance traveled by the facilities and the clients.

Formally, the CMFLP is set on a graph $G(V, E)$ where V denotes the set of vertices and E denotes the set of edges. A non-negative distance d_{ij} is defined for each edge $(i, j) \in E$. We interchangeably use cost and distance henceforth to indicate d_{ij} . The initial locations of the clients are represented by the subset $C \subseteq V$. Each client $i \in C$ has demand q_i and a positive weight u_i . There are different types of facilities with differing capacities. Each facility is of a type from the set T and the subset $F = \bigcup_{t \in T} F_t \subseteq V$ of vertices denotes the initial locations of the facilities (so F_t denotes the set of initial locations of facilities of type t). Each facility $j \in F_t$ has capacity Q_t and a positive weight w_j for relocation. All facilities are assumed to be

equipped with the same capabilities and therefore a client can get service from any one of them as long as the capacity limitations are satisfied. In a feasible solution to the CMFLP, each facility $j \in F$ moves to a destination vertex $v(j) \in V$ and each client $i \in C$ moves to a destination vertex $v(i) \in V$ with the condition that $v(i) = v(j)$ for some j . We assume a facility cannot share a destination vertex with another facility and a client can only be served by a single facility, i.e. demand cannot be split. Total demand assigned to a type t facility cannot exceed Q_t , for all $t \in T$. Clients or facilities may stay put (i.e., have their destination equal to their origin). Clients and facilities are also permitted to start at the same vertex. The objective is to minimize the total weighted distance traveled by the facilities and the clients, that is, $\sum_{j \in F} w_j d_{j,v(j)} + \sum_{i \in C} u_i d_{i,v(i)}$.

By including the capacity restrictions, the CMFLP extends the mobile facility location problem (MFLP) introduced previously by [Demaine et al. \[2009\]](#) to a practical setting. The CMFLP finds applications in logistics planning of community outreach programs delivered via mobile facilities such as library outreach programs in rural areas, mobile daycare delivered to farm children, and mobile schools that provide basic education to street children, as well as temporary schools servicing refugee camps. The deployment of mobile healthcare facilities (e.g. cancer screening units, blood banks, eye clinics, vaccination booths in case of a disease outbreak) that serve beneficiaries residing in either urban districts or rural regions is another important application area of the CMFLP. In these applications, districts (population centers) that have patients residing in them are represented by client vertices in the CMFLP. Mobile medical facilities currently located at some of the districts

are represented by facility vertices. The demand of a district shows the number of patients and their demands (i.e., visits to the medical facility) in the district and the capacity of a facility is the total number of patient visits it can handle within a time frame. Weights may be assigned to facilities and client locations according to priority, patient criticality, number of patient visits, etc. The objective of the problem is to move the mobile facilities so that every patient is served and the total weighted distance traveled by the facilities and the patients is minimized. After demand is served in an area or demand patterns have significantly changed, facilities may be relocated to a new area. The facility destinations in the previous network will be the originating facility vertices in the current network. Then, the problem can be solved with new clients and their respective demands.

The importance of mobile facilities is noted both in the medical and the operations research communities. [Geoffroy et al. \[2014\]](#) discuss the benefits of mobile healthcare facilities as a complementary service to fixed clinics by expanding access to healthcare for hard-to-reach areas. It is well-known that ease of geographical access to a healthcare facility has a major impact on the likelihood of participation in preventive healthcare services [see [Weiss et al., 1971](#)]. [Bingham et al. \[2003\]](#) investigated factors affecting the utilization of preventive services for cervical cancer and found the screening rates to be much lower in areas where services are distant or difficult to access. They reported greater transportation cost and distance as the main reasons for low participation rates. These examples motivate the use of the weighted distance objective in the CMFLP.

Studies addressing location decisions for healthcare facilities focus mainly on

fixed clinics and hospitals, and typically aim to maximize coverage of demand locations. For example, [Verter and Lapierre \[2002\]](#) model the preventive healthcare facility location problem as an extension of the Maximal Coverage Location Problem (MCLP). [Doerner et al. \[2007\]](#) study a tour planning problem for a single mobile healthcare facility with criteria concerned with the number of stops and tour length, and the distance to the nearest tour stop. [Ha et al. \[2013\]](#) discuss applications of the multi-vehicle covering tour problem related to deployment of mobile healthcare teams and mobile library teams and the distribution of relief items after a disaster. The problem involves choosing the stops of the vehicles from a set of potential locations so that every person can reach one of these stops within an acceptable time limit. The CMFLP differs from these studies significantly as it addresses capacity limitations of the facilities while minimizing the total distances traveled by both the facilities and the clients.

Our Contributions: In this chapter, we develop exact and heuristic algorithms to solve the CMFLP. We first compare two Integer Linear Programming (IP) formulations for the CMFLP. The first formulation, which we call the *layered graph* formulation, extends the one given in [Halper et al. \[2015\]](#) for the MFLP to account for the capacity constraints. The second formulation is a *set partitioning* formulation where each variable corresponds to a type of facility to be moved to a vertex in order to serve a feasible set of clients (i.e. the total demand of the clients cannot exceed the capacity). We prove that the LP relaxation of the set partitioning formulation provides a lower bound to the CMFLP that is greater than or equal to the LP relaxation bound from the layered graph formulation (and can be strictly better).

Next, we provide a branch-and-price algorithm for the set partitioning formulation where a column generation procedure is used on the set partitioning formulation to obtain lower bounds. Furthermore, we present two heuristic approaches for the CM-FLP. The first is an LP rounding heuristic that is also used to obtain good quality upper bounds within the branch-and-price algorithm. The second is a local search heuristic called 1-OptSwapBI that is adapted from one of the local search heuristics described in [Halper et al. \[2015\]](#).

To show the efficacy of the branch-and-price algorithm and the underlying column generation procedure, we conducted computational tests on instances adapted from [Halper et al. \[2015\]](#) (where each vertex hosts a client). We found out the ratio of the number of clients to the number of facilities plays an important role on the performance of both the branch-and-price algorithm and the heuristics. We solved the layered graph formulation using CPLEX as a benchmark. We observe that in general the problem is harder to solve when the the average number of clients per facility is relatively small (i.e., the ratio of $|C|$ to $|F|$ is small)—all tested algorithms struggle for these types of instances. However, in these instances the branch-and-price algorithm outperforms the CPLEX benchmark. Furthermore, the local search heuristic complements the branch-and-price algorithm by obtaining good solutions quickly when the average number of clients per facility is larger.

The rest of the chapter is organized as follows. Section [4.1](#) discusses related work in the literature. Section [4.2](#) describes two integer programming formulations. Section [4.3](#) describes the column generation procedure and the branch-and-price algorithm. Section [4.4](#) discusses heuristics, and Section [4.5](#) presents our computational

results.

4.1 Related Work

To the best of our knowledge the CMFLP has not been considered in the literature, its uncapacitated version, the MFLP was introduced by [Demaine et al. \[2009\]](#) as one of a class of movement problems. The majority of the previous work on the MFLP deals with the approximability of the problem and mainly consists of deriving theoretical bounds [e.g., [Friggstad and Salavatipour, 2011](#), [Armon et al., 2012](#), [Anari et al., 2016](#)]. [Halper et al. \[2015\]](#) introduced an IP formulation for the MFLP and developed various local search heuristics based on a decomposition of the problem. [Ahmadian et al. \[2013\]](#) showed that the local search heuristic n -OptSwap introduced by [Halper et al. \[2015\]](#) is a $3 + O\left(\sqrt{\frac{\log \log n}{\log n}}\right)$ -approximation algorithm for the MFLP.

The CMFLP concerns heterogenous facilities. When all facilities have identical capacities, the special case of CMFLP with homogeneous facilities is obtained. The CMFLP with homogeneous facilities generalizes the well-studied capacitated p -median with single sourcing problem (CPMSP), in which facilities are not relocated from their initial locations, but their locations are to be determined. We can easily see that by setting the cost of moving each facility to zero in the CMFLP with homogeneous facilities, the CPMSP is obtained. A recent paper by [Stefanello et al. \[2015\]](#) provides a nice discussion of earlier work on this problem. They also develop a matheuristic that solves large-scale CPMSP instances (with up to 4500 nodes and

1000 facilities) and obtain small optimality gaps within an hour of computation time. Their heuristic approach mainly relies on eliminating variables iteratively from the mathematical model.

In the single source capacitated facility location problem (SCFLP), an opening cost is associated with each facility instead of specifying the number of facilities. [Guastaroba and Speranza \[2014\]](#), [Yang et al. \[2012\]](#), [Cortinhal and Captivo \[2003\]](#), [Chen and Ting \[2008\]](#), [Holmberg et al. \[1999\]](#), and [Ahuja et al. \[2004\]](#), among others, propose solution methods for this problem. [Guastaroba and Speranza \[2014\]](#) develop a kernel search algorithm and achieve near optimal solutions for large scale instances (with up to 1500 nodes and 300 potential facility locations, as well as 1000 nodes and 1000 potential facility locations). [Klose \[1999\]](#) and [Tragantalerngsak et al. \[2000\]](#) study an extension of the SCFLP by considering two echelons of facilities. Each second-echelon facility can be supplied by only one first-echelon facility, and each customer is serviced by only one second-echelon facility. While only the locations of the second-echelon facilities are selected in [Klose \[1999\]](#), the locations of first-echelon facilities are also selected in [Tragantalerngsak et al. \[2000\]](#). Similarly, two sets of facilities (intermediate and upper level) are located in [Addis et al. \[2012\]](#) and [Addis et al. \[2013\]](#), but the capacity of intermediate level facilities should also be determined by installing devices that provide different capacities at different costs. All upper level facilities have the same given capacity. The objective includes the cost of assigning clients to intermediate level facilities, and of intermediate level facilities to upper level facilities, in addition to the cost of locating facilities. To solve the two-level problem, [Addis et al. \[2012\]](#) propose a branch-and-price algorithm.

In dynamic facility location problems, facilities are relocated over a time horizon consisting of multiple periods [see [Arabani and Farahani, 2012](#), [Nickel and Saldanha da Gama, 2015](#), for overviews of studies on such problems]. Most of the existing multi-period location problems associate a fixed cost for opening and closing facilities or resizing the capacities that depends on the location of the facility. For instance, [Torres-Soto and Uster \[2011\]](#) develop exact solution methods for capacitated multi-period relocation problems with fixed relocation costs, where demand of a customer can be serviced by multiple facilities partially. On the other hand, [Melo et al. \[2006\]](#) include a unit variable cost of moving capacity from an existing facility to a new facility, in addition to the fixed opening and closing costs. In their model, relocation decisions are constrained by budget limitations, and the objective includes production/supply costs, transportation costs between facilities, inventory holding costs, and fixed facility operating costs. To the best of our knowledge, none of the existing dynamic facility location models consider a fixed cost of relocating a facility that depends on the initial and destination locations, as in the CMFLP.

Column generation and branch-and-price approaches have been widely used in the literature to solve the CPMSP and the SCFLP. [Lorena and Senne \[2004\]](#) implement a column generation approach to solve the LP relaxation of the set covering formulation of the CPMSP. The new columns are generated by solving a 0-1 knapsack problem for pricing and a Lagrangean/surrogate relaxation identified from the dual of the master problem to accelerate convergence. The relaxation also provides lower bounds. [Ceselli and Righini \[2005\]](#) describe a branch-and-price algorithm that uses column generation for the CPMSP. At each iteration of column generation, the

current values of the dual variables are used as Lagrangian multipliers to compute a lower bound as in [Lorena and Senne \[2004\]](#). The authors experiment with two branching strategies and computational experiments suggest that the performance of the branch-and-price algorithm is closely related to ratio of the number of clients ($|C|$) to the number of facilities $|F|$. [Klose and Görtz \[2007\]](#) describe a column generation and branch-and-price algorithm for the SCFLP. The method is based on a Lagrangean relaxation of the demand constraints and a stabilized column generation method for solving the corresponding master problem to optimality.

4.2 Integer Programming Formulations

We present two IP formulations for the CMFLP. The first one is the capacitated version of the formulation in [Halper et al. \[2015\]](#), which we refer to as the layered graph formulation. We describe a decomposition based on this formulation which leads to a local search algorithm. The second formulation is a set partitioning formulation for which we describe a branch-and-price algorithm where the variables in the layered graph formulation are used for branching and the LP relaxation is solved via a column generation procedure.

4.2.1 Layered Graph Formulation

An instance of the CMFLP can be represented in a graph with three layers. After making copies of the client vertices C and the facility vertices F , the copies of the facility vertices make up the first layer. The vertex set V makes up the

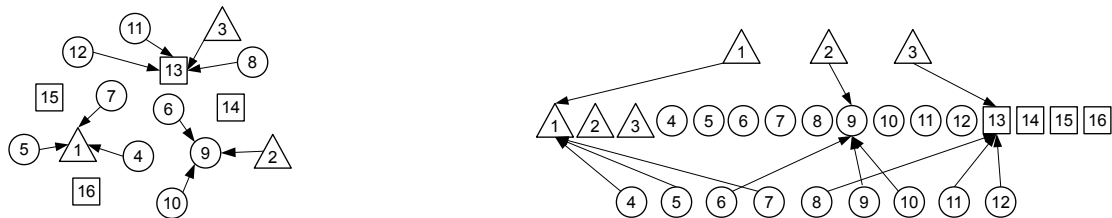


Figure 4.1: The original graph representation (left) and the layered graph representation (right) of a solution to an instance of the CMFLP. Triangles denote the facility vertices, circles denote the client vertices and squares denote the remaining vertices.

second layer and the copies of the client vertices make up the last layer. Figure 4.1 shows an example of the transformation from the original graph to the layered graph representation. The layered graph representation aids visualizing the formulation and the decomposition technique described next.

We define a binary variable x_{iv} for each $i \in C$ and $v \in V$, and a binary variable y_{jv} for each $j \in F$ and $v \in V$. Let $x_{iv} = 1$, if the destination of client i is vertex v ; and $x_{iv} = 0$, otherwise. Similarly, let $y_{jv} = 1$, if the destination of facility j is vertex v ; and $y_{jv} = 0$, otherwise. For each vertex $v \in V$ and each type $t \in T$, we define a binary variable z_{tv} such that $z_{tv} = 1$, if vertex v is the destination of some facility of type t ; and $z_{tv} = 0$, otherwise. The CMFLP is formulated as follows:

$$(IP1) \text{ Minimize } \sum_{i \in C} \sum_{v \in V} u_i d_{iv} x_{iv} + \sum_{j \in F} \sum_{v \in V} w_j d_{jv} y_{jv} \quad (4.1)$$

$$\text{subject to } \sum_{v \in V} x_{iv} = 1 \quad i \in C \quad (4.2)$$

$$\sum_{v \in V} y_{jv} = 1 \quad j \in F \quad (4.3)$$

$$\sum_{j \in F_t} y_{jv} = z_{tv} \quad v \in V, t \in T \quad (4.4)$$

$$\sum_{t \in T} z_{tv} \leq 1 \quad v \in V \quad (4.5)$$

$$x_{iv} \leq \sum_{t \in T} z_{tv} \quad i \in C, v \in V \quad (4.6)$$

$$\sum_{i \in C} q_i x_{iv} \leq \sum_{t \in T} Q_t z_{tv} \quad v \in V \quad (4.7)$$

$$z_{tv}, y_{jv}, x_{iv}, \in \{0, 1\}. \quad i \in C, j \in F, v \in V, t \in T \quad (4.8)$$

In IP1, the objective function (4.1) calculates the total weighted distance traveled by the facilities and clients. Constraints (4.2) and (4.3) ensure that each client and facility has a destination vertex. If $z_{tv} = 1$, constraints (4.4) and (4.5) specify that vertex v is the destination of a facility of type t and cannot host more than one facility. In the case that $z_{tv} = 0$, no facility of type t may have vertex v as its destination. Constraint (4.6) states that client i may travel to location v only if there is a facility moving to v . By constraint (4.7), total demand for a facility cannot exceed its capacity. Constraint (4.8) defines the binary variables. Leaving z_{tv} variables binary, y_{jv} variables can be relaxed in the interval $[0, 1]$ since constraints (4.3) and (4.4) correspond to the totally unimodular assignment constraints.

This formulation lends itself to a decomposition when the z_{tv} variables are fixed. Suppose we are given destination vertices for each type $t \in T$ such that the values of z_{tv} variables are fixed to 1 for all $v \in Z_t$, and z_{tv} is fixed to 0 for the remaining vertices. Let $Z = \bigcup_{t \in T} Z_t \subset V$. Then, the problem decomposes into a total of $|T| + 1$ disjoint subproblems of assigning each facility in F_t to a vertex in Z_t for every $t \in T$ (the $|T|$ facility assignment problems), and assigning each client to

a vertex in Z (the client assignment problem).

In the facility assignment problems, the objective is to find a minimum cost bipartite matching between the initial facility locations in F_t and the destination locations in Z_t . For each type $t \in T$, if $z_{tv} = 0$, then constraints (4.4) imply $y_{jv} = 0$ for all $j \in F_t$ and $v \in V \setminus Z_t$. Then for $z_{tv} = 1$, constraints (4.4) can be rewritten as $\sum_{j \in F_t} y_{jv} = 1$ and $v \in Z_t$. Given a subset $Z_t \subset V$, the facility assignment problem (FA(Z_t, t)) can be formulated as,

$$\begin{aligned}
 \text{FA}(Z_t, t) = \text{Minimize} & \quad \sum_{j \in F_t} \sum_{v \in Z_t} w_j d_{jv} y_{jv} \\
 \text{subject to} & \quad \sum_{v \in Z_t} y_{jv} = 1 & \quad j \in F_t \\
 & \quad \sum_{j \in F_t} y_{jv} = 1 & \quad v \in Z_t \\
 & \quad y_{jv} \geq 0 & \quad j \in F_t, v \in Z_t.
 \end{aligned}$$

which models the least cost bipartite matching problem. Since the constraint matrix is totally unimodular, the integrality of y_{jv} is relaxed. The facility assignment problem can be solved in polynomial time via the Hungarian Algorithm [see [Kuhn, 1955](#)].

In the client assignment problem, the objective is to assign each client to one of the facility destination locations in Z such that if $z_{tv} = 1$, then the facility assigned to location v has a total demand less than Q_t and the total weighted distance traveled by the clients is minimized. For $v \in V \setminus Z$, we have $\sum_{t \in T} z_{tv} = 0$ and constraints (4.6) imply that $x_{iv} = 0$ for $i \in C$ since a client cannot be located at a destination

without a facility. Therefore, constraints (4.6) can be rewritten as $x_{iv} \leq 1$ for $i \in C, v \in Z$. However, these constraints are redundant. Given a subset $Z \subset V$, the client assignment problem (CA(Z)) is the well-studied Generalized Assignment Problem (GAP) [see [Cattrysse and Van Wassenhove, 1992](#)] and can be formulated as,

$$\begin{aligned}
 \text{CA}(Z) = \text{Minimize} \quad & \sum_{i \in C} \sum_{v \in Z} u_i d_{iv} x_{iv} \\
 \text{Subject to} \quad & \sum_{v \in Z} x_{iv} = 1 && i \in C \\
 & \sum_{i \in C} q_i x_{iv} \leq Q_t && t \in T, v \in Z_t \\
 & x_{iv} \in \{0, 1\} && i \in C, v \in Z,
 \end{aligned}$$

A similar decomposition technique was described in [Halper et al. \[2015\]](#) for the MFLP, which has identical facilities without capacity restrictions by design, i.e., $|T| = 1$ and $Q_t = \infty$. The decomposition in [Halper et al. \[2015\]](#) is extended here to the CMFLP, such that the facility assignment problem is solved separately for each type $t \in T$. In the client assignment problem described for the MFLP, each client is assigned to its closest facility while in the CMFLP, the client assignment problem is the GAP which is NP-Hard.

4.2.2 Set Partitioning Formulation

Let \mathcal{S}_{tv} denote the set of all feasible client assignments to the facility of type t to be located in v . A client assignment S_{tv} is feasible if $\sum_{i \in S_{tv}} q_i \leq Q_t$. Let $a_{iS_{tv}}$ be a binary coefficient taking the value 1 if client i appears in assignment S_{tv} , and 0, otherwise. For an assignment $S_{tv} \in \mathcal{S}_{tv}$, $d_{S_{tv}}$ denotes the weighted travel cost of clients in S_{tv} . That is, $d_{S_{tv}} = \sum_{i \in S_{tv}} u_i d_{iv}$. Furthermore, for all $S_{tv} \in \mathcal{S}_{tv}$, let $\pi_{S_{tv}}$ be a binary variable indicating if customers in S_{tv} are assigned to the facility of type t that will be located at v . Let y_{jv} be a binary variable indicating if the facility j is moved to v . The set partitioning formulation is as follows.

$$\text{(IP2) Minimize} \quad \sum_{t \in T} \sum_{v \in V} \sum_{S_{tv} \in \mathcal{S}_{tv}} d_{S_{tv}} \pi_{S_{tv}} + \sum_{j \in F} \sum_{v \in V} d_{jv} y_{jv} \quad (4.9)$$

$$\text{subject to} \quad \sum_{v \in V} \sum_{S_{tv} \in \mathcal{S}_{tv}} \pi_{S_{tv}} = |F_t| \quad t \in T \quad (4.10)$$

$$\sum_{t \in T} \sum_{v \in V} \sum_{S_{tv} \in \mathcal{S}_{tv}} a_{iS_{tv}} \pi_{S_{tv}} = 1 \quad i \in C \quad (4.11)$$

$$\sum_{t \in T} \sum_{S_{tv} \in \mathcal{S}_{tv}} \pi_{S_{tv}} \leq 1 \quad v \in V \quad (4.12)$$

$$\sum_{v \in V} y_{jv} = 1 \quad j \in F \quad (4.13)$$

$$\sum_{j \in F_t} y_{jv} = \sum_{S_{tv} \in \mathcal{S}_{tv}} \pi_{S_{tv}} \quad v \in V, t \in T \quad (4.14)$$

$$\pi_{S_{tv}} \in \{0, 1\} \quad v \in V, t \in T, \\ S_{tv} \in \mathcal{S}_{tv} \quad (4.15)$$

$$y_{jv} \in \{0, 1\} \quad j \in F, v \in V \quad (4.16)$$

The objective function calculates the weighted distance traveled by facilities and clients. Constraints (4.10) assert that the total number of facilities moved for each type is equal to the total number of facilities of that type. Constraints (4.11) ensure that a client only appears in exactly one assignment. By constraints (4.12), at most one facility can be assigned to the same location. Similar to IP1, by setting $\pi_{S_{tv}}$ binary, the y_{jv} variables can be relaxed in the interval $[0, 1]$ since constraints (4.13) and (4.14) correspond to the assignment constraints that are totally unimodular.

Lemma 4.2.1 *The optimal objective value of the LP relaxation of IP2 (namely, LP2) is greater than or equal to the optimal objective value of the LP relaxation of IP1 (namely, LP1).*

Proof We first show that any feasible solution to LP2 can be transformed to a feasible solution of LP1 of equal cost. Let π and y be a feasible solution to LP2. Let

$$z_{tv} = \sum_{S_{tv} \in \mathcal{S}_{tv}} \pi_{S_{tv}} \quad t \in T, v \in V, \quad (4.17)$$

$$x_{iv} = \sum_{t \in T} \sum_{S_{tv} \in \mathcal{S}_{tv}} a_{iS_{tv}} \pi_{S_{tv}} \quad i \in C, v \in V \quad (4.18)$$

and y_{jv} indicates if the facility j is moved to location v in both formulations. After the transformation, constraints (4.2), (4.3), (4.4) and (4.5) are identical to constraints (4.11), (4.13), (4.14) and (4.12), respectively. From the transformation of

x_{iv} , we get

$$\begin{aligned}
x_{iv} &= \sum_{t \in T} \sum_{S_{tv} \in \mathcal{S}_{tv}} a_{iS_{tv}} \pi_{S_{tv}}, \\
&\leq \sum_{t \in T} \sum_{S_{tv} \in \mathcal{S}_{tv}} \pi_{S_{tv}}, \\
&= \sum_{t \in T} z_{tv},
\end{aligned}$$

since $a_{iS_{tv}}$ is a 0-1 coefficient, which implies that constraints (4.6) hold.

By definition, all feasible assignments $S_{tv} \in \mathcal{S}_{tv}$ satisfy $\sum_{i \in S_{tv}} q_i \leq Q_t$ for all $t \in T, v \in V$, which can also be stated as $\sum_{i \in C} q_i a_{iS_{tv}} \leq Q_t$. By summing up each side for $t \in T$ and $S_{tv} \in \mathcal{S}_{tv}$ and multiplying each side by $\pi_{S_{tv}}$, a nonnegative term, we get

$$\sum_{i \in C} \sum_{t \in T} \sum_{S_{tv} \in \mathcal{S}_{tv}} q_i a_{iS_{tv}} \pi_{S_{tv}} \leq \sum_{t \in T} Q_t \sum_{S_{tv} \in \mathcal{S}_{tv}} \pi_{S_{tv}}.$$

After replacing the corresponding terms with x_{iv} and z_{tv} , we have

$$\sum_{i \in C} q_i x_{iv} \leq \sum_{t \in T} Q_t z_{tv},$$

which is the set of constraints (4.7). Note that the lower and upper bound constraints for the variables are satisfied by the transformation. The second terms in the objective functions of both LP1 and LP2 are identical. Therefore, we focus on the first terms. In LP2, after replacing $d_{S_{tv}}$ with $\sum_{i \in S_{tv}} u_i d_{iv}$ in the first term, we

get

$$\begin{aligned}
\sum_{t \in T} \sum_{v \in V} \sum_{S_{tv} \in \mathcal{S}_{tv}} d_{S_{tv}} \pi_{S_{tv}} &= \sum_{t \in T} \sum_{v \in V} \sum_{S_{tv} \in \mathcal{S}_{tv}} \sum_{i \in S_{tv}} u_i d_{iv} \pi_{S_{tv}}, \\
&= \sum_{t \in T} \sum_{v \in V} \sum_{S_{tv} \in \mathcal{S}_{tv}} \sum_{i \in C} u_i d_{iv} a_{iS_{tv}} \pi_{S_{tv}}, \\
&= \sum_{i \in C} \sum_{v \in V} u_i d_{iv} x_{iv},
\end{aligned}$$

which is the first term in the objective function of LP1. Therefore, a feasible solution to LP2 can be transformed into a feasible solution to LP1 of equal cost.

Now we provide an example where the objective value of the optimal solution of LP1 is strictly less than the objective value of the optimal solution of LP2. Consider the example in Figure 4.2 with 3 nodes, 2 identical facilities, and 3 clients. Facilities 1 and 2 with a capacity of 5 are initially located at nodes 1 and 2, respectively. Clients 1, 2, and 3 with a demand of 1, 3, and 4 are at nodes 1, 2, and 3 respectively. Distances are given as $d_{11} = d_{22} = d_{33} = 0$, $d_{12} = d_{21} = d_{23} = d_{32} = 1$ and $d_{13} = d_{31} = 2$. Let $w_j = 1$ for all facilities and $u_i = 1$ for all clients. The optimal solution to LP1 is: $y_{11} = y_{22} = 1$, $x_{11} = x_{22} = 1$ and $x_{31} = x_{32} = 0.5$ with objective value 1.5. The optimal solution to LP2 gives us $\pi_{S_{11}}, \pi_{S_{22}} = 1$, where $S_{11} = \{1, 3\}$, $S_{22} = \{2\}$. This indicates that facilities stay put and clients 1 and 3 are assigned to facility 1 and client 2 is assigned to facility 2, which is in fact the optimal solution to IP1. In this example, the lower bound obtained from LP1 is 1.5 and the lower bound obtained from LP2 is 2. The optimality gap of LP1 is 33%, whereas the optimality gap of LP2 is 0%.

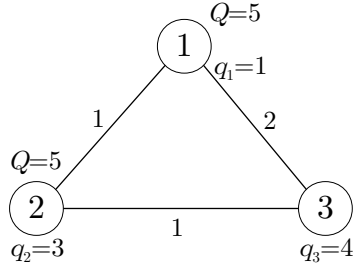


Figure 4.2: IP1, IP2 and LP2 have objective value of 2 (facilities and clients located at 1 and 2 stay put and client 3 is assigned to facility 1). However, LP1 has an objective value of 1.5, with a solution that splits the demand of client 3 between facilities 1 and 2.

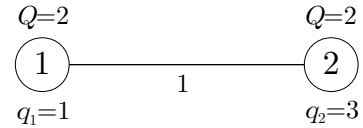


Figure 4.3: While LP2 is infeasible, LP1 is feasible with an objective value of $1/3$.

There is a more serious problem with the linear relaxation of IP1. It may be feasible when IP1 is infeasible. Consider the example in Figure 4.3 with 2 identical facilities and 2 clients. The distance between the two nodes is 1 and both facilities and clients have weight 1. Facilities 1 and 2 both have capacity 2, while client 1 has demand 1 and client 2 has demand 3. Clearly, this problem is infeasible. However, when we solve LP1, we obtain a feasible solution $y_{11} = y_{22} = 1$, $x_{11} = 1$, $x_{21} = 1/3$, $x_{22} = 2/3$ with objective value $1/3$, in which client 1 is assigned to facility 1 and client 2 is partially assigned to both facilities. Both facilities remain at their locations and the capacity constraints are satisfied. In contrast, LP2 is infeasible.

4.3 Column Generation and Branch-and-Price Procedure

In this section we describe a column generation procedure to solve LP2. We then apply a branch-and-price algorithm to IP2 (solving the linear relaxation using the column generation procedure), in which the variables of formulation (IP1)

are used for branching. We provide three branching alternatives and discuss the management of the columns.

4.3.1 Column Generation Procedure for LP2

Even though LP2 provides better bounds compared to LP1, there are exponentially many combinations of clients that can make up the set S_{tv} . Instead of solving LP2 with all S_{tv} for all $t \in T, v \in V$, we describe a column generation procedure that generates columns after we solve the restricted master problem (RMP), i.e., LP2 without the complete set of $\pi_{S_{tv}}$ columns. Let us consider the dual of LP2.

$$\text{(LP2}_D\text{) Maximize } \sum_{t \in T} |F_t| \alpha_t + \sum_{i \in C} \beta_i + \sum_{v \in V} \gamma_v + \sum_{j \in F} \delta_j \quad (4.19)$$

$$\text{subject to } \alpha_t + \sum_{i \in C} a_{iS_{tv}} \beta_i + \gamma_v - \omega_{tv} \leq d_{S_{tv}} \quad t \in T, v \in V,$$

$$S_{tv} \in \mathcal{S}_{tv} \quad (4.20)$$

$$\delta_j + \sum_{t \in T} \omega_{tv} \leq w_j d_{jv} \quad j \in F, v \in V \quad (4.21)$$

$$\gamma_v \leq 0 \quad v \in V \quad (4.22)$$

For primal optimality, we need dual feasibility. Note that constraints (4.21) are always satisfied since all of the y_{jv} variables are in the RPM. However, for each $t \in T, v \in V$, we need to make sure there is no assignment S_{tv} such that $\alpha_t + \sum_{i \in C} a_{iS_{tv}} \beta_i + \gamma_v - \omega_{tv} > d_{S_{tv}}$. To find such assignments, we solve the following

pricing problem, which is a 0-1 Knapsack Problem.

$$\text{(KP}(t, v)) \text{ Maximize} \quad \sum_{i \in C} (\beta_i - u_i d_{iv}) \xi_i \quad (4.23)$$

$$\text{subject to} \quad \sum_{i \in C} q_i \xi_i \leq R_{tv} \quad (4.24)$$

$$\xi_i \in \{0, 1\} \quad i \in C \quad (4.25)$$

When the pricing problem is solved at some node of the branch-and-price tree, the capacity of a facility may already be partially allocated. Therefore, we denote the remaining capacity of a type t facility at v by R_{tv} . At the root node of the tree $R_{tv} = Q_t$. In order to ensure a given solution is primal optimal (dual feasible), we have to solve $\text{KP}(t, v)$ for all types t and vertices v (that are not fixed to zero by branching). Let $S_{tv} = \{i \mid \xi_i = 1\}$. We check whether constraint (4.20) is satisfied. If the constraint is satisfied for all types t and vertices v , then we conclude that the solution is optimal. Otherwise, we add the column of $\pi_{S_{tv}}$ for every S_{tv} that violates (4.20) and resolve the RMP. The general outline of the Column Generation Procedure is as follows.

Column Generation Procedure

Step 1: Generate an initial set of feasible columns for the RMP.

Step 2: Solve the RMP with the existing columns and calculate the values of the optimal dual variables.

Step 3: By solving the pricing problems $\text{KP}(t, v)$, find columns such that (4.20) is

violated. If such columns exist, add them to the RMP and go to Step 2. Otherwise, terminate with the optimal solution.

4.3.2 Branching Scheme

The knapsack problems we solve for pricing depends on both the branching scheme we employ and the node of the branch-and-price tree. Branching on the variables of IP2 (i.e. $\pi_{S_{tv}}$) is not a viable option for the following reason. Consider branching on the variable $\pi_{S_{tv}}$, where S_{tv} is a feasible client assignment. For the branch where $\pi_{S_{tv}} = 0$, only the specific assignment S_{tv} is forbidden. Therefore, any other assignment in \mathcal{S}_{tv} must still be considered. In order to do that, each client in S_{tv} must be excluded from $KP(t, v)$ one by one. As the number of forbidden assignments increases, the number of knapsack problems to be solved also increases drastically. However, branching on the variables z_{tv} and x_{iv} does not have this problem and provides a much cleaner column generation process. We describe three branching strategies that use z_{tv} and x_{iv} variables after transforming them as in (4.17) and (4.18).

4.3.2.1 Binary Branching

In binary branching, we fix z_{tv} and x_{iv} variables to 1 in one branch, and to 0 in the other branch. We first start by branching on the z_{tv} variables since it is not possible to branch on x_{iv} without having branched on $z_{tv} = 1$ at one of the parent nodes for some t . Consider the branch where $z_{tv} = 1$, then some facility of type

t will move to v . Since no other type of facility t' can move to v , we set $z_{t'v} = 0$ for all $t' \in T \setminus \{t\}$. In addition, the constraint corresponding to v from the set of constraints (4.12) has to be set as an equality. That is, for v , the constraint is modified to $\sum_{t \in T} \sum_{S_{tv} \in \mathcal{S}_{tv}} \pi_{S_{tv}} = 1$ in the restricted master problem (RMP) of LP2. For the branch where $z_{tv} = 0$, vertex v is discarded for type t as a candidate for a facility destination and $\text{KP}(t, v)$ is not solved.

For the branch where $x_{iv} = 1$, client i is assigned to vertex v . Then all $x_{iv'}$ for $v' \in V \setminus \{v\}$ can be set to zero. For all types $t \in T$, we adjust the residual capacity to $R_{tv} - q_i$ while solving $\text{KP}(t, v)$ and exclude client i from $\text{KP}(t, v)$. For the branch where $x_{iv} = 0$, client i is simply excluded from $\text{KP}(t, v)$ for all $t \in T$.

Among all fractional z_{tv} , we branch on the most fractional one (i.e. closest to 0.5). If there does not exist a fractional z_{tv} , then among all i and v pairs, we branch on the most fractional x_{iv} given that z_{tv} is fixed to 1.

4.3.2.2 Partition Branching

Partition branching is similar to the branching strategies proposed for the generalized assignment problem by [Savelsbergh \[1997\]](#) and for the CPMSP by [Ceselli and Righini \[2005\]](#). Given a client $i \in C$, we divide the set of vertices V into two sets V^+ and V^0 such that $V^+ = \{v \mid x_{iv} > 0, v \in V\}$ and $V^0 = \{v \mid x_{iv} = 0, v \in V\}$. Then we further partition V^+ and V^0 into two sets such that $V^+ = V_1^+ \cup V_2^+$ and $V^0 = V_1^0 \cup V_2^0$. We set $V_1 = V_1^+ \cup V_1^0$ and $V_2 = V_2^+ \cup V_2^0$. A balanced partition can be achieved by sorting the vertices in V^+ in non-increasing order of x_{iv} and

assigning them alternately to V_1^+ and V_2^+ . We assign the vertices in V^0 to V_1^0 and V_2^0 in a similar fashion. We branch on the client i^* that satisfies $i^* = \arg \max\{|V^+|\}$, breaking ties arbitrarily. Finally, we set $x_{i^*v} = 0$ for all $v \in V_1$ in one branch and $x_{i^*v} = 0$ for all $v \in V_2$ in the other branch. In the column generation procedure, setting $x_{i^*v} = 0$ translates into removing client i^* from $\text{KP}(t, v)$ for all t .

4.3.2.3 Hybrid Branching

Ceselli and Righini [2005] reported that partition branching performs better than the binary branching for the CPMSP, which is a special case of the CMFLP where $|T| = 1$ and $w_j = 0$ for all $j \in F$. After preliminary computational experiments performed on instances for the CMFLP with $T = \{1, 2\}$, we have observed that using partition branching by itself is inferior to the binary branching in terms of average computational time and nodes explored. This may attest to the differences between the two problems. In hybrid branching, we use binary branching on z_{tv} variables. When there is no fractional z_{tv} , we employ partition branching for the x_{iv} variables. Though not necessary, branching on z_{tv} variables before the partition branching improves the computational time according to our tests.

4.3.3 Columns Management

Columns management is an integral part of any column generation procedure as it significantly affects the computational effort required to complete the procedure. There are three pillars to managing columns to which every column generation

procedure needs to attend. First, the initial set of columns to start the procedure. Second, the addition of new columns through the pricing problem or other approaches. Third, the management of the existing columns. In the literature, there are various schools of thought on columns management. As it is prohibitive to examine all possible approaches proposed in previous studies, we experimented on a few of the better practices in the literature with adjustments of our own.

4.3.3.1 Setting initial columns

At the root node of the branch-and-price tree, we generate an initial set of columns for the RMP by a greedy algorithm targeted towards obtaining feasible solutions in short time. We let the facilities stay in their original locations. Therefore, the algorithm only assigns clients to the facilities. Let R_j be the remaining capacity of facility $j \in F$. Initially $R_j = Q_t$, if $j \in F_t$. Also, initially let $F' = F$ and $C' = C$. The initial column generation algorithm is outlined as follows.

Initial Column Generation Algorithm

For each $j \in F'$, go through the following steps while $C' \neq \emptyset$ and $F' \neq \emptyset$.

Step 1: Let $i^* = \arg \min_{i \in C' | q_i \leq R_j} \left\{ \frac{u_i d_{ij}}{q_i} \right\}$.

Step 2: If $i^* = \emptyset$, then set $F' = F' \setminus \{j\}$; otherwise, assign i^* to facility j and set

$$R_j = R_j - q_{i^*} \text{ and } C' = C' \setminus \{i^*\}.$$

When the algorithm terminates, either $C' = \emptyset$ or $F' = \emptyset$. If $C' = \emptyset$, it means that all clients are assigned and we have a feasible solution. If $F' = \emptyset$, it means there

are clients left unassigned and there is no facility with enough remaining capacity to accommodate them. In this case, we run the following *assigned-unassigned client exchange* procedure.

Assigned-Unassigned Client Exchange Procedure

Step 1: For an unassigned client $i \in C'$, assign i to its closest facility $j \in F$ such that $q_i \leq R_j$. If no such facility exists, then let $F' = F$ and go to Step 2.

Step 2: If $F' = \emptyset$, then terminate. Otherwise, consider the facility $j \in F'$ that is closest to i ; find a client i' assigned to facility j such that i' satisfies the following criteria:

- $q_i > q_{i'}$
- $R_j - q_i + q_{i'} \geq 0$
- if more than one client satisfy the above criteria, pick the client with the larger $u_{i'} d_{i'j}$.

Step 3: If i' does not exist, set $F' = F' \setminus \{j\}$ and go to Step 2. Otherwise, exchange i with i' , i.e., assign i to j and i' to C' and go to Step 1.

Note that even after running this procedure, we may still not have a feasible solution. In that case, we add a separate dummy variable for each constraint with a very large objective function coefficient to have a starting feasible solution.

In addition to a starting feasible solution, we generate more columns by creating a feasible assignment S_{tv} for each type $t \in T$ and vertex $v \in V$ according to

the following procedure.

Generation of Additional Columns

Step 1: For each v , sort the list of clients with respect to $u_i d_{iv}$ in non-decreasing order. Let $\bar{d}_v = \sum_{i \in C} u_i d_{iv} / |C|$ be the average weighted distance of clients to v .

Step 2: For each type t , let $R_{tv} = Q_t$ be the remaining capacity.

- For each client i on the sorted list: Let $r \sim U[0, 1]$. If $r \leq e^{-u_i d_{iv} / \bar{d}_v}$, and $q_i \leq R_{tv}$, then add client i to the assignment and set $R_{tv} = R_{tv} - q_i$. Otherwise, process the next client.

We run the additional column generation procedure m times resulting in m feasible assignments for each type t and vertex v . Note that instead of totally random assignments, we use this procedure so that clients that are closer to a given vertex v have a higher chance of being in the feasible assignment S_{tv} . After the preliminary computational experiments, we set $m = 5$, as it caused the largest decrease in the average computational time. Compared to $m = 0$, that is, the case where no additional columns are added to the initial feasible solution, setting $m = 5$ decreases the computational time required to solve the root node two to three-fold in most of the instances.

The child node inherits all the active columns from the parent node. However, we ensure the columns corresponding to infeasible assignments based on the branching decision have sufficiently large objective function coefficients. That way,

these columns will be replaced by other columns that would yield a lower objective value. To ensure the child node has a starting feasible RMP, we add a separate dummy column for each constraint with a very large objective function coefficient. If column generation procedure terminates with columns corresponding to infeasible assignments and/or dummy columns in the optimal basis, we can conclude that the node is infeasible and proceed to prune the node.

4.3.3.2 Adding columns through pricing

While solving the exact $KP(t, v)$ in every RMP iteration is possible, finding columns that violate (4.20) does not necessarily mean that we have to solve the pricing problem exactly. Rather, we only have to solve the exact $KP(t, v)$ for all types t and vertices v to ensure that a given solution is optimal. Therefore, we prefer to use a greedy 2-approximation algorithm for the sake of computational time. The clients are sorted in non-increasing order of $(\beta_i - u_i d_{iv})/q_i$ and the knapsack is filled until no capacity is left. We check the set of constraints (4.20) for violations. We only solve the exact $KP(t, v)$ when the greedy algorithm fails to find violating columns. If the exact solution also fails to find violating columns, then we terminate with an optimal solution. However, if violating columns have been found after solving the exact $KP(t, v)$, then we add those to the RMP and switch to applying the greedy algorithm until it fails again.

4.3.3.3 Managing active columns

Even though there are exponentially many π_{Stv} variables, only $|T| + |C| + |V| + |T| \cdot |V|$ can be in the basis, hence a vast majority of them will be non-basic. The size of the problem grows every time we add a column, but the number of basic columns stay exactly the same. Clearly, the growth in the number of columns reflects badly on the computational time. To remedy this, we introduce a procedure that removes columns from the RMP. If a variable is non-basic for κ consecutive iterations, we remove that variable from the RMP. This procedure ensures that the size of the RMP stays in $O(\kappa(|T| + |C| + |V| + |T| \cdot |V|))$. Note that a removed column may be added again due to possible regeneration. This may increase the number of iterations and the total number of columns added to the RMP but the gain in computational time is well-justified based on the preliminary computational experiments.

4.4 Heuristics

We describe two heuristics for the CMFLP. The first is an LP rounding heuristic which is employed at all the nodes of the branch-and-price tree. The second is a local search heuristic.

4.4.1 LP Rounding Heuristic

By rounding the optimal fractional solution at any node of the branch-and-price tree, it is possible to quickly find good quality feasible solutions to the CMFLP

and generate primal bounds. After calculating the values of z_{tv} and x_{iv} variables from the optimal fractional solution to the corresponding LP2 as in (4.17) and (4.18), we run the following heuristic to obtain a feasible solution to the CMFLP.

Step 1: Sort the z_{tv} variables in non-increasing order. Then for each t , select the first $|F_t|$ vertices to a set named Z_t .

Step 2: Solve the facility assignment problems ($\text{FA}(Z_t, t)$) which sets the destination vertices for the facilities.

Step 3: For the client assignment problem, we run the following subroutine.

- Create a list of clients and their preferred vertices. Pair any client i in the list with the vertex v such that x_{iv} is closest to 1.
- Sort the list in non-increasing order of x_{iv} . Starting from the top of the list, assign each client to the facility that has its preferred vertex as the destination. Adjust its remaining capacity. If there is not enough remaining capacity, then go to the next client in the list.
- At the end of the list, if there are some clients left unassigned because there was not enough remaining capacity, assign them to the nearest facility with enough remaining capacity.

Step 4: Finally, run the following improvement heuristic.

- Evaluate all possible client shifts, i.e., removing the client from its current facility and assigning it to a different facility. Implement the

shift that would best improve the total cost. If no such shift is found, go to the next step.

- Evaluate all possible client swaps, i.e., exchanging clients that are assigned to different facilities. Implement the swap that would best improve the total cost. If there is no improving swap, then the heuristic is terminated.

The LP rounding heuristic is run every time a feasible LP solution is obtained in the branch-and-price tree. Note that the LP rounding heuristic is not guaranteed to terminate with an integer feasible solution. In fact, determining whether or not an instance to the CMFLP has a feasible solution is NP-Complete.

4.4.2 Local Search Heuristic

In [Halper et al. \[2015\]](#), the authors describe several heuristics for the MFLP based on the decomposition of the MFLP to facility and client assignment problems for a given set of facility destination vertices. Even though the facility and client assignment problems are different for the CMFLP, the general framework of the local search heuristics still applies. Here, instead of having a single facility assignment problem, we have $|T|$ facility assignment problems and instead of a polynomially solvable client assignment problem, we have the NP-Hard generalized assignment problem.

In n -OptSwapBI (where BI stands for best improvement), we are given a set of facility destination vertices $Z \subset V$. For each type t , a subset of k_t facility destina-

tions in Z_t are replaced by a subset of k_t destinations in $V \setminus Z_t$. Every possible combination of replacements across all types are considered such that $1 \leq \sum_{t \in T} k_t \leq n$. For each replacement, the corresponding facility assignment problems are solved optimally by the Hungarian Algorithm. For the facility assignment problem $\text{FA}(Z_t, t)$, the Hungarian algorithm requires $O(|Z_t|^3)$ from scratch. However, [Halper et al. \[2015\]](#) describe a procedure to update the facility assignments in $O(k_t|Z_t|^2)$, given the previous optimal assignments. We also employ this update procedure in our computations.

To solve the client assignment problem, we use the same greedy algorithm we have used to generate feasible solutions in the column generation procedure outlined in Section 4.3.3.1, albeit with one caveat. Instead of choosing i^* according to $i^* = \arg \min_{i \in C' | q_i \leq R_j} \left\{ \frac{u_i d_{ij}}{q_i} \right\}$ in Step 1, we use $i^* = \arg \min_{i \in C' | q_i \leq R_j} \{u_i d_{ij}\}$ in order to target solution quality rather than feasibility. In the case that the algorithm terminates with unassigned clients, we run the same assigned-unassigned client exchange procedure.

In [Halper et al. \[2015\]](#), the computational results indicate that setting $n > 1$ is not viable computationally, even for the MFLP where there is a single facility assignment problem and the client assignment problem is solvable in polynomial time. Hence, we focus on the case where $n = 1$. That is, we consider replacing each facility destination in Z_t with every other destination in $V \setminus Z_t$ for each type t by solving both facility and client assignment problems, and select the replacement that yields the most decrease in the objective value. Note that the neighborhood of 1-OptSwapBI for the CMFLP is populated by $\sum_{t \in T} |Z_t|(|V| - |Z_t|)$ possible replacements.

4.5 Computational Results

In order to assess the solution quality and computational efficiency of the branch-and-price algorithm and the underlying column generation procedure, we coded the branch-and-price algorithm to solve IP2 as described in Section 4.3.2. We used the hybrid branching scheme in the results reported since it performed the best during the preliminary computational experiments. We evaluate the nodes in the branch-and-price tree according to breadth-first search. We used CPLEX to solve IP1 as a benchmark to the branch-and-price algorithm. In this section we first provide results on the root node LP relaxations for IP1 and IP2, namely, LP1 and LP2 to compare the strength of the formulations. We also compare the solutions obtained from the LP rounding heuristic based on LP2, namely LP2RH, and the local search heuristic LSH with those obtained from the branch-and-price algorithm. We provide results regarding both the case with homogeneous facilities and the case with two types of facilities with respect to their capacity values.

4.5.1 Test Instances

The computational experiments are performed on 45 instances titled **p-med** (40 of them are adapted from Halper et al. [2015]). Originally, **p-med** instances were generated for the p-median problem and adapted to the MFLP by Halper et al. [2015]. We further adapted the instances to the CMFLP and generated more instances for structural consistency. The computational studies performed on these instances provide insights into the relationships between the solution quality and

the computational efficiency of the proposed algorithms, as well as the structural properties of the instances.

The first 30 instances (`p-med1` through `p-med30`) are grouped in fives with respect to the number of nodes and clients (for `p-med` instances, we have $|C| = |V|$). The first group of five instances (`p-med1` through `p-med5`) have 100 nodes with increasing number of facilities, the second group of five have 200 nodes, and so on. However, the last 10 instances (`p-med31` through `p-med40`) are not structured in groups of five. Rather, instances `p-med31` through `p-med34` have 700 nodes, instances `p-med35` through `p-med37` have 800 nodes, and instances `p-med38` through `p-med40` have 900 nodes. For consistency, we generated one 700 node instance (named `p-med34-1` using the shortest path distances, client and facility weights and client demands of instance `p-med34` only generating more facility locations to conform to the structure observed in the first 30 instances. In a similar fashion, two instances each were generated for 800 and 900 node instances (named `p-med37-1` and `p-med37-2`, and `p-med40-1` and `p-med40-2`).

The instances are adapted to the CMFLP by generating the demand q_i for all $i \in C$. We draw q_i randomly from a Gamma distribution with $\alpha = 5$ and $\beta = 2$. If q_i exceeds $\frac{0.8 \cdot E[q_i] \cdot |C|}{|F|}$, we set it to $\frac{0.8 \cdot E[q_i] \cdot |C|}{|F|}$ so that the demand can be served by a single facility with some slack. We experimented with homogeneous facilities where $|T| = 1$ and heterogeneous facilities where $|T| = 2$. For homogeneous facilities, the capacity Q is set to $\frac{\sum_{i \in C} q_i}{0.9 \cdot |F|}$.

When we have two type of facilities, the facilities are alternately assigned to F_1 and F_2 , and their capacities Q_1 and Q_2 are set to $\frac{0.65 \sum_{i \in C} q_i}{0.9 \cdot |F_1|}$ and $\frac{0.35 \sum_{i \in C} q_i}{0.9 \cdot |F_2|}$. We

provide some slack to the total capacity by scaling so that the problem is feasible with very high probability. In fact, we have not encountered an infeasible instance.

We observed a relationship between the ratio of the number of nodes (clients) to the number of facilities, $|V|/|F|$, and the quality of solutions obtained from both IP formulations, their LP relaxations, and the heuristics, just like [Ceselli and Righini \[2005\]](#) did for the capacitated p-median problem. For this reason, we grouped the test instances with respect to $|V|/|F|$, as $|V|/|F| \leq 10$ and $|V|/|F| > 10$. The ratio $|V|/|F|$ represents the expected average number of clients assigned per facility.

4.5.2 Computational Settings

CPLEX MIP solver is used to solve IP1 and LP1. We also solved IP1 after disabling the default CPLEX cuts, which we denote as IP1*, to assess the performance of IP1 in a plain branch-and-bound framework. LP2 is solved using the column generation procedure described in [Section 4.3](#), where the exact KP is solved by a dynamic program that runs in $O(|F_t|Q_t)$ and the master LP is solved using CPLEX. IP2 is solved using the branch-and-price algorithm proposed in [Section 4.3.2](#). Within the algorithm, LP2 is used to obtain lower bounds and the LP rounding heuristic is used to obtain upper bounds. Recall that we remove columns staying nonbasic for κ iterations. After preliminary analysis, we concluded that setting κ to $\lceil 0.15 \cdot |V| \rceil$ and $\lceil 0.1 \cdot |V| \rceil$ provides the most average decrease in run time for $|V| \leq 500$ and $|V| > 500$, respectively. We have implemented the branch-and-price algorithm using C++ where the RMP is solved with CPLEX. We used CPLEX version 12.5 coded

in C++ in all computational experiments and ran the instances on a computer with Intel Core i7-2600 CPU @ 3.40 GHz and 16 GB of RAM running 64-bit Windows 7.

Due to excessive computational times for large instances, we limited our runs to three hours (10800 seconds). If an instance of IP1 or IP2 was terminated at three hours, we report the objective value of the best integer solution found, namely the best upper bound, and the best lower bound found. The performance of the branch and price algorithm and the underlying column generation procedure is inferior in instances where $|V|/|F| > 10$. Among these instances, in almost half of the instances with $|T| = 1$ and for almost all instances with $|T| = 2$, the column generation procedure had to be terminated due to the time limit before finding the root node LP optimal solution. On the other hand, IP1 has found the optimal (or near optimal) solutions with relative ease for these instances. We therefore omit the tables with full results for instances where $|V|/|F| > 10$ and refer only to average results.

4.5.3 Homogeneous Facilities Case

We present the results in three tables. Tables 4.1, 4.2, and 4.3 contain the results for $|V|/|F| \leq 10$. In all tables, if a value cannot be calculated due to the time limit, we denote the corresponding cell with ‘-’. The running times exceeding three hours are also denoted with ‘-’.

4.5.3.1 Comparison of the LP Relaxations

Table 4.1 summarizes the computational results for LP1 and LP2 with respect to the best IP lower and upper bounds. The first column specifies the names of the instances. Generically, let $X(L)$ denote the best lower bound obtained from model X after 3 hours of computation, where X can be either IP1, IP1* or IP2. Similarly, $X(U)$ denotes the best feasible integer solution obtained from model X after 3 hours of computation. In the first group of columns, we report the best lower bound (BL) from either IP1(L), IP1*(L), or IP2(L) and the best upper bound (BU) from either IP1(U), IP1*(U), IP2(U) for each instance, along with the source of the bound. If the same bounds are found by IP1, IP*, and IP2, we specify the source as ‘ALL’. Note that if IP1, IP1* or IP2 terminated with the optimal solution, then BU=BL. For example, in Table 4.1, the source of the best known upper bound for the `p-med15` instance is IP1(U). That is, the integer feasible solution with the lowest objective function value for `p-med15` is obtained from IP1. On the other hand, the source of the best known lower bound for `p-med15` is IP2(L), meaning that it is found while solving IP2 by the branch-and-price algorithm. In the second group of columns labeled Gap (%), we provide the percentage gaps between various formulations. For all reported gaps denoted as $X-Y$, the gaps are calculated as $(X - Y)/X$. The column labeled BU-BL provides the gap between the best upper bound and the best lower bound, i.e. the best known optimality gap. The columns labeled BU-LP1 and BU-LP2 denote the gap between the best upper bound and the lower bounds obtained by solving LP1 and LP2, respectively. Similarly, the

columns labeled BL-LP1 and BL-LP2 denote the gap between the best lower bound and lower bounds of LP1 and LP2. The group of columns labeled ‘Running time (s)’ provides the CPU times in seconds. Finally, the size of the instances are given in the last group of columns.

In Table 4.1, where $|V|/|F| \leq 10$, we observe that as $|V|$ gets larger, especially for $|V| \geq 600$, IP2 starts to overtake IP1 at finding the best upper bound. Furthermore, the best lower bound is found by IP2 when $|V| \geq 300$. LP2 provides significantly smaller gaps than LP1 on the average with 1.80% vs. 5.62% compared to the best upper bound. In addition, we observe that the quality of LP2 compared to LP1 gets progressively better as $|V|/|F|$ gets smaller. Compared to the upper bound, LP2 has an average gap of 0.80% vs. 1.27% of LP1 for $|V|/|F| = 10$. The gaps become 2.09% vs. 4.92% for $|V|/|F| = 5$ and 2.62% vs. 11.17% for $|V|/|F| = 3$. We attribute this difference to the packing constraints (4.7) in LP1 which lead to more fractional variables as $|F|$ gets larger. These computational results confirm the theoretical finding that IP2 is a stronger formulation than IP1. In general, as expected, LP1 runs faster than LP2 with an average of 128.2 vs. 519.7 seconds. As a result, we observe an apparent trade-off between obtaining smaller gaps and having longer run times.

The significant difference between the quality of LP2 and LP1 bounds is nowhere to be found when $|V|/|F| > 10$, as the average gap between LP2 and the best upper bound is 0.54%, whereas the gap of LP1 is 0.56% for instances where LP2 terminated within the time limit, and 0.63% overall. However, the quality of the best upper bound is also poorer for $|V|/|F| \leq 10$, where the average BU-BL

gap is 1.43% as opposed to a 0.05% average gap for $|V|/|F| > 10$. Therefore, it does not necessarily mean that LP2 performs poorly for $|V|/|F| \leq 10$. In fact, the quality of LP2 seems to be somewhat robust with $|V|/|F|$. On the other hand, the running time of LP2 increases rapidly as $|V|/|F|$ gets larger. We believe the reason for the increase is the number of variables in LP2 being tied to the number of feasible assignments. Recall that for an assignment S_{tv} to be feasible, we need to have $\sum_{i \in S_{tv}} q_i \leq Q_t$. For larger $|V|/|F|$, Q_t is also large but the demand is drawn from the same distribution. That leads to more feasible combinations of clients, which in turn leads to more variables for LP2. Even with all the improvements to the column generation procedure, there are eight (out of seventeen) instances for which LP2 could not be solved in three hours. In summary, we can conclude that in terms of running time and the quality of the lower bound, LP2 and LP1 complement each other with increasing $|V|/|F|$.

4.5.3.2 Comparison of the Lower Bounds

In Table 4.2, where $|V|/|F| \leq 10$, the computational results for IP1(L), IP1*(L), and IP2(L) and the running times of IP1, IP1*, and IP2 are presented. The columns labeled BL-IP1(L), BL-IP1*(L) and BL-IP2(L) report the gap between the best lower bound and IP1(L), IP1*(L) and IP2(L). On average, IP2(L) is better than IP1(L) with an average value of 0.04% compared to 0.47%, which is somewhat expected given the quality of LP2 vs. LP1 when $|V|/|F| \leq 10$. In fact, CPLEX does a pretty good job closing the initial gap of LP1, which is 4.27% on average with

respect to the best lower bound. On the other hand, deprived of its state-of-the-art cuts, we observe that the poor quality of LP1 hinders the ability of IP1* at closing the gap, which is 3.53% on average but gets as large as 9.71%. In terms of running time, IP1, IP1* and IP2 all hit the three hour limit when $|V| \geq 300$ (except `p-med13` for IP1) with IP1 faring slightly better.

When $|V|/|F| > 10$, IP1(L) performs better than IP2(L), providing the best lower bound in all instances. IP2(L) has an average gap 0.31% compared to the best lower bound. Even IP1*(L) performs as well as IP1(L), which is somewhat expected since CPLEX cuts will be more useful when the packing problem is harder. We do not have the complete results for IP2 (because of the three hour time limit) for this subset of instances but the averages from the available results suggest that IP2 is much more robust with respect to $|V|/|F|$. However, the long running times of IP2 suggests IP1 should be the formulation of choice for $|V|/|F| > 10$, especially given that the running time of IP1 decreases significantly as $|V|/|F|$ gets larger.

4.5.3.3 Comparison of the Upper Bounds

In Table 4.3, where $|V|/|F| \leq 10$, the computational results for IP1(U), IP1*(U), IP2(U), the LP rounding heuristic from LP2 (LP2RH), and the local search heuristic (LSH) are presented. The columns labeled IP1(U)-BL, IP1*(U)-BL and IP2(U)-BL report the gap between IP1(U), IP1*(U) and IP2(U), and the best lower bound, i.e., the optimality gap. In the column LP2RH-BL, the gap between the best integer feasible solution found by LP2RH and the best lower bound is given. In

the next column labeled LSH-BL, we provide the gap between the feasible solution found by LSH and the best lower bound.

We observe that on average, IP2 performs better than IP1 in finding a good feasible solution. IP1 gives an average gap of 4.02% while IP2 yields 1.60% gap on the average with respect to the best lower bound. When we discard instance `p-med40-2` for which IP1 terminated with a very poor quality upper bound, IP2 still performs better compared to the lower bound with gaps 1.54% vs. 1.68%. Especially when $|V| \geq 600$, excluding instance `p-med40-2`, the average IP1(U)-BL gap is 3.12% vs. 2.56% of IP2(U)-BL, which signals that the relative quality of IP2 solutions get better in larger instances.

The LP rounding heuristic performs fairly well, given the hardness of this subset of instances and the poor quality of the local search heuristic. On average, the gap is 4.63%, which means by just solving LP2 at the root node and using the LP rounding heuristic, we get an integer solution which is on average at most 4.63% away from the optimal. However, the quality of the local search heuristic is extremely poor for $|V|/|F| \leq 10$ with an average gap of 10.38%. This is somewhat expected since the greedy procedure for the GAP tends to work better when the number of items assigned to a single bin gets larger. The running time of the LP rounding heuristic nearly equals the running time of LP2 since the steps after LP2 solution take negligible time. Therefore, we see that the LP rounding heuristic is quite fast for $|V|/|F| \leq 10$ and gets even faster as $|V|/|F|$ gets smaller. In contrast, the local search heuristic runs quite slowly since the neighborhood size is larger for $|V|/|F| \leq 10$. For $|V|/|F| \leq 10$, we observe that the speed of LP2 does not translate

into IP2 since the number of nodes explored also increases for $|V|/|F| \leq 10$. We see two reasons for the increased number of nodes. First one is simply the increased number of variables as $|F|$ gets larger. The second reason is more subtle. After fixing a z_{tv} variable, we naturally expect the lower bound obtained at that node to increase. However, as $|F|$ increases, the marginal increase in the lower bound caused by fixing a single facility decreases. This in turn makes the algorithm explore more nodes as we observe in the column labeled ‘Nodes Explored’, which provides the total number of nodes explored by the branch-and-price algorithm that solves IP2.

The running times of the local search heuristic are consistent with the local search neighborhood size and are much smaller for small $|F|$. In general, the performance of the local search heuristic gets better as $|V|/|F|$ gets larger. Especially when $|V|/|F| > 10$, the solution quality is much better with an average gap of 1.15% compared to the best lower bound. Local search heuristic enjoys the same benefits LP1 does; the packing problem is easier and the problem behaves more like MFLP. Note that for the same local search heuristic (1-OptSwapBI) applied to MFLP, where the client assignment problem is solved optimally in polynomial time, [Halper et al. \[2015\]](#) observes the gap between the local search solution and the optimal solution to be less than 0.20% on average. As a result, we can claim that LP2RH and LSH complement each other in terms of both solution quality and running times, which makes it a viable option to use LSH for large $|V|/|F|$ and LP2RH for small $|V|/|F|$.

Table 4.1: Comparison of the quality of LP1 and LP2 when $|V|/|F| \leq 10$ and $|T| = 1$.

Instance	Objective Value				Gap (%)					Runtime (s)		V	V / F
	BL	Source	BU	Source	BU-BL	BU-LP1	BU-LP2	BL-LP1	BL-LP2	LP1	LP2		
pmed2	5275.27	ALL	5275.27	ALL	0.00%	1.47%	0.60%	1.47%	0.60%	0.42	0.98	100	10
pmed3	6023.05	ALL	6023.05	ALL	0.00%	0.61%	0.27%	0.61%	0.27%	0.24	0.89	100	10
pmed4	5374.39	ALL	5374.39	ALL	0.00%	4.19%	0.78%	4.19%	0.78%	0.26	0.45	100	5
pmed5	3320.49	ALL	3320.49	ALL	0.00%	9.86%	2.22%	9.86%	2.22%	0.30	0.34	100	3
pmed8	6658.71	ALL	6658.71	ALL	0.00%	1.31%	0.70%	1.31%	0.70%	1.94	16.66	200	10
pmed9	5046.37	IP1(L)	5061.78	IP1(U)	0.30%	4.31%	1.21%	4.02%	0.90%	2.81	4.17	200	5
pmed10	3128.94	IP1(L)	3128.94	IP1(U)	0.00%	8.56%	0.92%	8.56%	0.92%	1.62	2.90	200	3
pmed13	5690.55	IP1(L)	5690.55	IP1(U)	0.00%	0.73%	0.32%	0.73%	0.32%	7.75	45.52	300	10
pmed14	4427.26	IP2(L)	4465.27	IP1*(U)	0.85%	4.15%	1.16%	3.32%	0.31%	7.77	10.76	300	5
pmed15	3416.81	IP2(L)	3459.56	IP1(U)	1.24%	9.65%	1.47%	8.52%	0.24%	5.50	9.92	300	3
pmed18	6010.19	IP2(L)	6038.28	IP1(U)	0.47%	1.04%	0.70%	0.58%	0.23%	19.95	93.34	400	10
pmed19	4674.05	IP2(L)	4749.88	IP1*(U)	1.60%	4.80%	2.03%	3.25%	0.44%	14.99	36.60	400	5
pmed20	3928.93	IP2(L)	3966.80	IP1(U)	0.95%	10.25%	1.13%	9.39%	0.18%	14.81	23.65	400	3
pmed23	6996.73	IP2(L)	7023.91	IP2(U)	0.39%	1.04%	0.59%	0.66%	0.20%	62.15	252.86	500	10
pmed24	5320.38	IP2(L)	5397.27	IP1(U)	1.42%	4.25%	1.62%	2.87%	0.20%	34.57	124.43	500	5
pmed25	3852.84	IP2(L)	3972.88	IP1*(U)	3.02%	12.11%	3.19%	9.38%	0.17%	27.83	85.02	500	3
pmed28	6367.63	IP2(L)	6390.13	IP1(U)	0.35%	0.90%	0.43%	0.55%	0.08%	101.12	386.88	600	10
pmed29	5286.17	IP2(L)	5405.70	IP2(U)	2.21%	5.34%	2.35%	3.20%	0.14%	64.94	193.69	600	5
pmed30	4031.32	IP2(L)	4187.17	IP2(U)	3.72%	13.41%	4.04%	10.06%	0.33%	40.11	91.39	600	3
pmed33	6981.21	IP2(L)	7007.81	IP1(U)	0.38%	0.87%	0.51%	0.49%	0.13%	227.11	793.56	700	10
pmed34	5075.42	IP2(L)	5254.71	IP2(U)	3.41%	6.07%	3.55%	2.75%	0.14%	163.35	222.80	700	5
pmed34-1	4079.74	IP2(L)	4241.66	IP2(U)	3.82%	12.10%	3.93%	8.61%	0.12%	59.85	129.40	700	3
pmed37	6536.65	IP2(L)	6646.71	IP1(U)	1.66%	2.21%	1.81%	0.57%	0.16%	384.68	1055.25	800	10
pmed37-1	5081.89	IP2(L)	5218.90	IP2(U)	2.63%	5.49%	2.71%	2.94%	0.09%	279.35	592.47	800	5
pmed37-2	4190.18	IP2(L)	4337.52	IP1(U)	3.40%	12.44%	3.48%	9.36%	0.08%	111.42	6332.21	800	3
pmed40	7397.00	IP2(L)	7543.70	IP1(U)	1.94%	2.53%	2.09%	0.60%	0.15%	1127.80	2115.01	900	10
pmed40-1	5790.58	IP2(L)	5984.67	IP2(U)	3.24%	5.64%	3.37%	2.47%	0.13%	478.20	1164.40	900	5
pmed40-2	4642.74	IP2(L)	4790.29	IP2(U)	3.08%	12.12%	3.19%	9.33%	0.11%	220.26	246.65	900	3
				Min	0.00%	0.61%	0.27%	0.49%	0.08%	0.24	0.34		
				Max	3.82%	13.41%	4.04%	10.06%	2.22%	1127.80	6332.21		
				Avg	1.43%	5.62%	1.80%	4.27%	0.37%	123.61	501.15		

Table 4.2: Comparison of the quality of lower bounds obtained from IP1, IP1* and IP2 when $|V|/|F| \leq 10$ and $|T| = 1$.

Instance	Gap (%)			Runtime (s)			V	V / F
	BL-IP1(L)	BL-IP1*(L)	BL-IP2(L)	IP1	IP1*	IP2		
pmed2	0.00%	0.00%	0.00%	9.45	34.45	13.26	100	10
pmed3	0.00%	0.00%	0.00%	1.05	1.15	10.13	100	10
pmed4	0.00%	0.00%	0.00%	32.67	973.69	41.24	100	5
pmed5	0.00%	4.01%	0.00%	112.29	-	3868.58	100	3
pmed8	0.00%	0.47%	0.00%	1227.54	-	7949.99	200	10
pmed9	0.00%	2.92%	0.32%	-	-	-	200	5
pmed10	0.00%	6.52%	0.62%	1591.44	-	-	200	3
pmed13	0.00%	0.32%	0.06%	1048.31	-	-	300	10
pmed14	0.39%	2.69%	0.00%	-	-	-	300	5
pmed15	0.44%	7.88%	0.00%	-	-	-	300	3
pmed18	0.07%	0.24%	0.00%	-	-	-	400	10
pmed19	0.84%	2.93%	0.00%	-	-	-	400	5
pmed20	0.42%	8.86%	0.00%	-	-	-	400	3
pmed23	0.29%	0.57%	0.00%	-	-	-	500	10
pmed24	0.66%	2.55%	0.00%	-	-	-	500	5
pmed25	0.43%	8.88%	0.00%	-	-	-	500	3
pmed28	0.15%	0.47%	0.00%	-	-	-	600	10
pmed29	0.71%	3.13%	0.00%	-	-	-	600	5
pmed30	0.79%	9.71%	0.00%	-	-	-	600	3
pmed33	0.25%	0.45%	0.00%	-	-	-	700	10
pmed34	0.89%	2.69%	0.00%	-	-	-	700	5
pmed34-1	1.47%	8.47%	0.00%	-	-	-	700	3
pmed37	0.36%	0.53%	0.00%	-	-	-	800	10
pmed37-1	1.00%	2.93%	0.00%	-	-	-	800	5
pmed37-2	1.28%	9.31%	0.00%	-	-	-	800	3
pmed40	0.36%	0.57%	0.00%	-	-	-	900	10
pmed40-1	1.00%	2.46%	0.00%	-	-	-	900	5
pmed40-2	1.40%	9.20%	0.00%	-	-	-	900	3
Min	0.00%	0.00%	0.00%	1.05	1.15	10.13		
Max	1.47%	9.71%	0.62%	-	-	-		
Avg	0.47%	3.53%	0.04%	8273.57	9822.32	9297.10		

Table 4.3: Comparison of the quality of upper bounds obtained from IP1, IP1*, IP2, LP2RH, and LSH when $|V|/|F| \leq 10$ and $|T| = 1$.

Instance	Gap (%)					Runtime (s)	Nodes	V	V / F
	IP1(U)-BL	IP1*(U)-BL	IP2(U)-BL	LP2RH-BL	LSH-BL	LSH	Explored		
pmed2	0.00%	0.00%	0.00%	0.90%	3.54%	0.32	33	100	10
pmed3	0.00%	0.00%	0.00%	0.24%	3.10%	0.29	7	100	10
pmed4	0.00%	0.00%	0.00%	3.80%	6.39%	1.48	366	100	5
pmed5	0.00%	0.00%	0.00%	7.10%	12.84%	4.43	49973	100	3
pmed8	0.00%	0.06%	0.00%	2.57%	5.32%	6.48	944	200	10
pmed9	0.30%	0.59%	0.30%	2.15%	10.41%	34.77	31539	200	5
pmed10	0.00%	0.40%	0.68%	6.04%	13.78%	68.33	38760	200	3
pmed13	0.00%	0.00%	0.01%	0.70%	2.80%	40.73	1125	300	10
pmed14	0.95%	0.85%	1.11%	4.92%	8.20%	191.19	14590	300	5
pmed15	1.24%	1.58%	1.93%	10.04%	16.68%	401.13	12410	300	3
pmed18	0.47%	0.51%	0.69%	4.50%	3.33%	188.12	2489	400	10
pmed19	2.06%	1.60%	1.80%	4.90%	9.61%	863.69	6742	400	5
pmed20	0.95%	1.93%	1.79%	7.28%	12.97%	2100.53	6317	400	3
pmed23	0.44%	0.64%	0.39%	2.55%	4.19%	559.10	1521	500	10
pmed24	1.42%	1.61%	1.79%	6.00%	8.45%	3126.61	4128	500	5
pmed25	3.13%	3.02%	3.06%	6.70%	15.73%	5201.88	3552	500	3
pmed28	0.35%	0.39%	0.51%	3.92%	5.89%	1184.15	1311	600	10
pmed29	2.87%	3.26%	2.21%	4.14%	8.77%	4679.60	3785	600	5
pmed30	5.83%	5.80%	3.72%	5.86%	16.23%	-	3003	600	3
pmed33	0.38%	0.48%	0.58%	3.83%	4.10%	2384.93	907	700	10
pmed34	3.99%	4.72%	3.41%	5.47%	8.73%	-	2855	700	5
pmed34-1	5.03%	5.62%	3.82%	5.62%	20.20%	-	2062	700	3
pmed37	1.66%	1.92%	1.72%	3.13%	5.06%	3905.30	620	800	10
pmed37-1	3.21%	4.39%	2.63%	4.01%	12.98%	-	1931	800	5
pmed37-2	3.40%	9.39%	4.18%	6.80%	22.16%	-	586	800	3
pmed40	1.94%	2.59%	2.12%	5.64%	4.32%	7157.25	519	900	10
pmed40-1	5.61%	5.08%	3.24%	5.47%	18.71%	-	1060	900	5
pmed40-2	67.21%	9.57%	3.08%	5.47%	26.04%	-	876	900	3
Min	0.00%	0.00%	0.00%	0.24%	2.80%	0.29	7		
Max	67.21%	9.57%	4.18%	10.04%	26.04%	-	49973		
Avg	4.02%	2.36%	1.60%	4.63%	10.38%	3846.47	6928.96		

4.5.4 Heterogeneous Facilities Case

We present the results in three tables (Tables 4.4 through 4.6) with the same layout and format used for homogeneous facilities. We have not implemented the Local Search heuristic for the heterogeneous facilities since it is significantly outperformed by every other method when $|V|/|F| \leq 10$. Furthermore, its neighborhood structure and size makes its performance highly predictable in terms of time and quality.

In Table 4.4, the BU-BL gap values show that the presence of heterogeneous facilities increases the difficulty of the problem as expected. For homogeneous facilities, the average BU-BL gap is 0.05% when $|V|/|F| > 10$. For heterogeneous facilities (where $|T| = 2$), the gap slightly increases to 0.09%, perhaps not indicative of a harder problem. However, when $|V|/|F| \leq 10$, the average BU-BL gaps are observed to be 1.43% and 2.86%, respectively for homogeneous and heterogeneous facilities. In general, the insights we gain from homogeneous facilities are still valid and more pronounced in heterogeneous facilities. When $|V|/|F| > 10$, LP2 terminated under three hours in only two out of seventeen instances, as opposed to nine when $|T| = 1$. On the other hand, the quality and the running time of LP2 seems to be affected less from the added dimension when $|V|/|F| \leq 10$.

In homogeneous facilities, we observe that IP2 starts to produce better upper bounds compared to IP1 when $|V| \geq 600$. Not only the trend holds up even stronger for heterogeneous facilities, but the quality of the upper bounds obtained from IP1 rapidly declines after $|V| \geq 700$. After disabling default CPLEX cuts, IP1* failed

to find a feasible solution in three hours for six out of nine instances for $|V| \geq 700$. The quality of the LP rounding heuristic slightly declines for heterogeneous facilities. However, compared to IP1, the LP rounding heuristic provides good quality solutions in reasonable time. Especially for $|V| \geq 700$ and $|V|/|F| \leq 10$, the LP rounding heuristic generally outperforms IP1. Similarly, the lower bound IP1(L) obtained from IP1 after three hours is worse than the lower bound obtained from LP2 for the same instances. Therefore, interestingly after solving the root node, the branch-and-price algorithm has better lower and upper bounds than IP1 has at its termination after three hours.

Table 4.4: Comparison of the quality of LP1 and LP2 when $|V|/|F| \leq 10$ and $|T| = 2$.

Instance	Objective Value				Gap (%)					Runtime (s)		V	V / F
	BL	Source	BU	Source	BU-BL	BU-LP1	BU-LP2	BL-LP1	BL-LP2	LP1	LP2		
pmed2	5064.83	ALL	5064.83	ALL	0.00%	1.25%	0.48%	1.25%	0.48%	0.24	7.47	100	10
pmed3	6162.93	ALL	6162.93	ALL	0.00%	1.99%	0.88%	1.99%	0.88%	0.19	1.98	100	10
pmed4	5172.33	ALL	5172.33	ALL	0.00%	3.16%	1.30%	3.16%	1.30%	0.17	0.73	100	5
pmed5	3302.98	ALL	3302.98	ALL	0.00%	9.10%	1.19%	9.10%	1.19%	0.22	0.53	100	3
pmed8	6484.06	IP1(L)	6484.06	IP1(U)	0.00%	2.18%	0.48%	2.18%	0.48%	2.62	32.15	200	10
pmed9	5043.84	IP2(L)	5115.69	IP1(U)	1.40%	6.02%	1.73%	4.68%	0.33%	2.40	6.24	200	5
pmed10	3199.40	IP1(L)	3240.95	IP1(U)	1.28%	11.36%	1.87%	10.21%	0.60%	2.62	9.92	200	3
pmed13	5673.47	IP1(L)	5673.47	IP1(U)	0.00%	1.09%	0.49%	1.09%	0.49%	10.90	87.39	300	10
pmed14	4372.98	IP2(L)	4414.65	IP1(U)	0.94%	4.40%	1.12%	3.49%	0.18%	6.43	17.36	300	5
pmed15	3364.28	IP2(L)	3447.41	IP1(U)	2.41%	10.63%	2.53%	8.42%	0.13%	6.38	13.31	300	3
pmed18	5997.19	IP2(L)	6038.59	IP1*(U)	0.69%	1.81%	0.80%	1.13%	0.12%	15.51	133.51	400	10
pmed19	4701.85	IP2(L)	4830.43	IP1*(U)	2.66%	6.28%	2.74%	3.72%	0.08%	16.68	34.05	400	5
pmed20	3973.66	IP2(L)	4162.68	IP1(U)	4.54%	14.31%	4.66%	10.23%	0.13%	15.07	30.55	400	3
pmed23	6997.44	IP2(L)	7039.70	IP1*(U)	0.60%	1.50%	0.67%	0.91%	0.07%	58.06	355.03	500	10
pmed24	5247.22	IP2(L)	5406.64	IP2(U)	2.95%	6.08%	3.02%	3.23%	0.07%	46.93	82.17	500	5
pmed25	3780.82	IP2(L)	3944.54	IP1(U)	4.15%	12.79%	4.21%	9.02%	0.06%	29.16	44.69	500	3
pmed28	6352.88	IP2(L)	6429.52	IP1(U)	1.19%	2.21%	1.26%	1.03%	0.07%	107.78	609.07	600	10
pmed29	5184.69	IP2(L)	5332.38	IP2(U)	2.77%	5.61%	2.84%	2.92%	0.07%	79.42	128.86	600	5
pmed30	3931.89	IP2(L)	4213.84	IP2(U)	6.69%	15.90%	6.77%	9.87%	0.08%	43.32	80.81	600	3
pmed33	6970.24	IP2(L)	7064.82	IP1*(U)	1.34%	2.26%	1.39%	0.93%	0.05%	260.57	861.42	700	10
pmed34	5056.57	IP2(L)	5267.82	IP2(U)	4.01%	6.76%	4.06%	2.87%	0.05%	204.86	202.46	700	5
pmed34-1	4079.69	IP2(L)	4358.50	IP2(U)	6.40%	15.31%	6.44%	9.52%	0.05%	129.03	124.13	700	3
pmed37	6507.84	IP2(L)	6655.00	IP2(U)	2.21%	3.14%	2.26%	0.95%	0.05%	500.87	969.15	800	10
pmed37-1	5092.52	IP2(L)	5352.55	IP2(U)	4.86%	8.55%	4.91%	3.89%	0.05%	446.82	341.45	800	5
pmed37-2	4134.17	IP2(L)	4558.39	IP2(U)	9.31%	18.41%	9.36%	10.04%	0.05%	159.37	187.87	800	3
pmed40	7351.84	IP2(L)	7510.01	IP2(U)	2.11%	2.98%	2.15%	0.89%	0.05%	1102.80	2620.10	900	10
pmed40-1	5742.88	IP2(L)	5918.55	IP2(U)	2.97%	6.12%	3.00%	3.25%	0.03%	575.02	524.82	900	5
pmed40-2	4731.60	IP2(L)	5192.53	IP2(U)	8.88%	17.77%	8.91%	9.76%	0.03%	313.58	238.67	900	3
				Min	0.00%	1.09%	0.48%	0.89%	0.03%	0.17	0.53		
				Max	9.31%	18.41%	9.36%	10.23%	1.30%	1102.80	2620.10		
				Avg	2.66%	7.11%	2.91%	4.63%	0.26%	147.75	276.64		

Table 4.5: Comparison of the quality of lower bounds obtained from IP1, IP1* and IP2 when $|V|/|F| \leq 10$ and $|T| = 2$.

Instance	Gap (%)			Runtime (s)			V	V / F
	BL-IP1(L)	BL-IP1*(L)	BL-IP2(L)	IP1	IP1*	IP2		
pmed2	0.00%	0.00%	0.00%	5.78	5.48	439.52	100	10
pmed3	0.00%	0.00%	0.00%	17.43	185.24	2113.54	100	10
pmed4	0.00%	0.00%	0.00%	26.18	205.45	277.40	100	5
pmed5	0.00%	0.00%	0.00%	194.19	-	1642.23	100	3
pmed8	0.00%	0.44%	0.31%	2240.12	-	-	200	10
pmed9	0.31%	3.19%	0.00%	-	-	-	200	5
pmed10	0.00%	8.54%	0.37%	-	-	-	200	3
pmed13	0.00%	0.38%	0.36%	8570.64	-	-	300	10
pmed14	0.60%	2.88%	0.00%	-	-	-	300	5
pmed15	0.75%	7.33%	0.00%	-	-	-	300	3
pmed18	0.40%	0.61%	0.00%	-	-	-	400	10
pmed19	1.28%	3.37%	0.00%	-	-	-	400	5
pmed20	1.54%	9.62%	0.00%	-	-	-	400	3
pmed23	0.48%	0.61%	0.00%	-	-	-	500	10
pmed24	1.33%	3.12%	0.00%	-	-	-	500	5
pmed25	1.66%	8.92%	0.00%	-	-	-	500	3
pmed28	0.58%	0.85%	0.00%	-	-	-	600	10
pmed29	1.24%	2.84%	0.00%	-	-	-	600	5
pmed30	1.51%	9.57%	0.00%	-	-	-	600	3
pmed33	0.63%	0.90%	0.00%	-	-	-	700	10
pmed34	1.41%	2.80%	0.00%	-	-	-	700	5
pmed34-1	2.23%	9.49%	0.00%	-	-	-	700	3
pmed37	0.71%	0.93%	0.00%	-	-	-	800	10
pmed37-1	2.11%	3.85%	0.00%	-	-	-	800	5
pmed37-2	3.06%	10.02%	0.00%	-	-	-	800	3
pmed40	0.81%	0.87%	0.00%	-	-	-	900	10
pmed40-1	1.81%	3.25%	0.00%	-	-	-	900	5
pmed40-2	3.16%	9.75%	0.00%	-	-	-	900	3
Min	0.00%	0.00%	0.00%	5.78	5.48	277.40		
Max	3.16%	10.02%	0.37%	-	-	-		
Avg	0.99%	3.72%	0.04%	8926.10	9716.26	9417.65		

Table 4.6: Comparison of the quality of upper bounds obtained from IP1, IP1*, IP2, and LP2RH when $|V|/|F| \leq 10$ and $|T| = 2$.

Instance	Gap (%)				Nodes Explored	V	V / F
	IP1(U)-BL	IP1*(U)-BL	IP2(U)-BL	LP2RH-BL			
pmed2	0.00%	0.00%	0.00%	1.78%	223	100	10
pmed3	0.00%	0.00%	0.00%	5.37%	2632	100	10
pmed4	0.00%	0.00%	0.00%	3.72%	1745	100	5
pmed5	0.00%	0.00%	0.00%	6.64%	18422	100	3
pmed8	0.00%	0.08%	0.21%	0.89%	932	200	10
pmed9	1.40%	2.23%	2.66%	6.32%	21716	200	5
pmed10	1.28%	1.64%	3.26%	7.89%	29006	200	3
pmed13	0.00%	0.12%	0.28%	1.51%	1032	300	10
pmed14	0.94%	1.88%	1.91%	4.85%	11439	300	5
pmed15	2.41%	2.60%	5.31%	13.47%	14975	300	3
pmed18	0.73%	0.69%	0.86%	2.17%	3221	400	10
pmed19	2.74%	2.66%	3.54%	10.15%	10408	400	5
pmed20	4.54%	7.66%	6.95%	12.05%	9592	400	3
pmed23	0.70%	0.60%	1.12%	3.08%	2484	500	10
pmed24	3.45%	3.05%	2.95%	7.19%	6493	500	5
pmed25	4.15%	5.92%	6.68%	13.09%	5447	500	3
pmed28	1.19%	1.25%	1.25%	5.71%	1334	600	10
pmed29	6.82%	6.58%	2.77%	3.95%	3781	600	5
pmed30	10.81%	18.81%	6.69%	10.71%	3964	600	3
pmed33	1.58%	1.34%	1.61%	1.65%	960	700	10
pmed34	4.98%	7.69%	4.01%	7.06%	3021	700	5
pmed34-1	71.41%	-	6.40%	8.48%	2529	700	3
pmed37	2.44%	-	2.21%	6.13%	601	800	10
pmed37-1	62.34%	-	4.86%	5.51%	1983	800	5
pmed37-2	11.66%	-	9.31%	11.31%	1746	800	3
pmed40	52.19%	50.73%	2.11%	5.14%	436	900	10
pmed40-1	60.86%	-	2.97%	4.42%	1375	900	5
pmed40-2	70.75%	-	8.88%	12.05%	1304	900	3
Min	0.00%	0.00%	0.00%	0.89%	223		
Max	71.41%	50.73%	9.31%	13.47%	29006		
Avg	13.55%	5.25%	3.17%	6.51%	5814.32		

Chapter 5: Concluding Remarks

Over the last two decades, the internet and personal mobile devices have become an essential part of daily life. It is now possible to collect and process data at an unprecedented level of speed and granularity. By performing descriptive and predictive analytics, practitioners are able to identify problems that, if solved optimally, will enhance productivity and profitability. Prescriptive analytics embarks upon developing methods for these problems, which are often too large to be handled by general purpose solvers thus requiring specialized approaches. In today's fast paced operational landscape, it is not enough to solve a problem optimally for a method to be useful, it needs to be computationally efficient as well. Therefore, in this dissertation, we develop methods, that balance solution quality with computational efficiency, for large-scale problems encountered in online advertising and logistics. We use optimization techniques that are well suited for these problems, e.g., decomposition and discretization, column generation and branch-and-price, and heuristic approaches. Each problem in this dissertation presents unique challenges and warrants a different approach, thus deserving a separate discussion of methodology, contributions, and insights, which we now present.

The Online Advertising Portfolio Optimization Problem: In Chap-

ter 2, we study the Online Advertising Portfolio Optimization Problem (OAPOP) faced by an advertiser on various online advertising platforms and exchanges. The advertiser manages portfolios of targeting items (keywords, cookies, websites, demographic dimensions, etc.) dispersed over a variety of advertising platforms and formats. The OAPOP combines ad campaigns across these platforms and formats under a consolidated portfolio and operates under a single advertising budget. Therefore, the number of targeting items in the portfolio may be in the tens of thousands for small businesses and tens of millions for large enterprises. Furthermore, the OAPOP is an operational problem and needs to be solved and resolved many times throughout the day, thus requiring fast solution approaches. By solving the OAPOP, the advertiser can determine how much to bid on each targeting item to maximize the return from the advertising budget. Further, the advertiser can understand the revenue-cost trade-off at different levels of ad spend.

We model the OAPOP as a Multiple Choice Knapsack Problem (MCKP) where each targeting item has multiple bid levels corresponding to expected cost and expected revenue estimates. We propose an efficient column generation algorithm where bid levels on the convex hull are generated as needed. We perform computational experiments on online advertising instances generated based on data collected from Google Adwords Keyword Planner. We demonstrate the column generation algorithm significantly outperforms the state-of-the-art linear time algorithm (proposed by Dyer [1984] and Zemel [1984]) when the advertising budget is less than 20% (the value is typically up to 5% for most portfolios) of the maximum amount possible to spend on the portfolio. The column generation algorithm scales well for

very large problems and in fact is able to solve enterprise scale problems (e.g., with 50 million targeting items and 2.55 billion variables) in a matter of minutes. This would make the column generation approach an integral part of the toolkit that the advertisers could use for doing multiple optimal bid cycles through the day as well as the foundational algorithm for performing what-if analysis for their ad portfolios. We further demonstrate that our column generation algorithm outperforms the linear time approach for standard problem instances in literature when the budget constraints are relatively tight.

We could further extend the OAPOP and cast it in the context of an exploration/exploitation framework. For many ad campaigns, the amount of historical click and revenue data available for each targeting item can be quite sparse. Hence, the advertiser would like to spend ad budgets while making a trade-off between expected revenue maximization for the targeting items with sufficient data and getting more traffic from those that have limited data.

The Bid Optimization Problem in Online Advertising: In 2013, the landscape of online advertising changed when Google introduced “Enhanced Campaigns”. In Enhanced Campaigns, the advertisers are able to modify their bids for targeting items (e.g., keywords, cookies, websites, etc.) based on ad query features (e.g., location, time, device, audience, etc.) using bid adjustments. The bid adjustments create opportunities for advertisers to better target desired user characteristics. While our approach proposed in Chapter 2 provides bid determination and budget allocation for portfolios not operating under enhanced campaigns, the portfolios subject to bid adjustments require special attention due to their target-

ing potential. However, the bid adjustments interact in a multiplicative manner in the bidding language thus leading to a computationally challenging problem. The bid adjustments are a recent development in online advertising. Therefore, there is a practical need for an efficient and effective approach to determine bid adjustments. In Chapter 3, we introduce the Bid Adjustment Problem in Online Advertising (BAPOA) that fully captures the practical setting of bid adjustments. In the BAPOA, the advertiser determines base bids and bid adjustments for an advertising portfolio to maximize expected revenue subject to an advertising budget.

The BAPOA can be modeled as a nonlinear mathematical program. However, the multiplicative nature of bid adjustments makes the problem very hard to solve optimally. We develop an approach where the mathematical programming formulation is decomposed into two subproblems. One to determine the base bids and one to determine the bid adjustments. We show that both subproblems can be modeled as Multiple Choice Knapsack Problems (MCKPs) when the domain of base bids and bid adjustments are discretized. Furthermore, we show how the solution to the linear programming relaxation of the MCKP can be used to obtain a feasible solution to the BAPOA. We iteratively create and solve subproblems, which we call the Iterative Adjustment Algorithm (IAA), to determine a high quality solution to the BAPOA. The IAA is a particularly attractive algorithm from a practical standpoint since the linear programming relaxation of the MCKP can be solved efficiently even for large problem sizes as we demonstrated in Chapter 2.

To evaluate the quality of the IAA and the benefits of using bid adjustments, we performed computational experiments on simulated data generated based on

sample data on Google Keyword Planner. We show the IAA provides near optimal solutions by comparing the revenue obtained from the IAA with an upper bound obtained from a MIP formulation in small instances. Our findings indicate the revenue benefit of using bid adjustments increases as the revenue-per-click variation across features increases and as the budget decreases. We observe the performance of the IAA is robust under varying problem sizes, budget amounts, and revenue-per-click variations. The running time of the IAA is nearly linear in the number of targeting items and the number of feature combinations, which indicates the algorithm can scale to handle very large instances of the BAPOA. In addition, we provide a procedure (that uses the IAA as a subroutine) where given a set of targeting items, the advertiser can construct campaigns and ad groups (by clustering targeting items based on bid adjustments), assign targeting items to these campaigns and ad groups, and determine base bids and bid adjustments that would maximize the benefit of using bid adjustments.

The introduction of bid adjustments to the bidding language is an exciting development. However, virtually no research currently exists on the subject despite its practical importance. Possible extensions to our work include the treatment of the problem in the context of an exploration/exploitation framework. Exploiting desired user characteristics would eventually lead to a sparsity of data thus hindering the ability of the advertiser to recognize shifts in user characteristic preferences. A healthy balance between exploration and exploitation would benefit the advertiser in the long run. Another venue of research could be a game theoretic approach to using bid adjustments where competitors (of the advertiser) use a similar (or

the same) tool to determine base bids and bid adjustments. In this environment, the targeting needs of the competitors might significantly effect the equilibrium behavior. For instance, every advertiser targeting the same user characteristic would increase competition (and the cost) and may render targeting ineffective due to an undesirable revenue-cost trade-off.

The Capacitated Mobile Facility Location Problem: In Chapter 4, we study the Capacitated Mobile Facility Location Problem (CMFLP). We compare two formulations for the CMFLP. The first formulation (IP1) is a layered graph formulation adapted from the MFLP formulation in [Halper et al. \[2015\]](#) to account for the capacity restrictions. The second formulation (IP2) is a set partitioning formulation. We show that the LP relaxation of IP2 (LP2) is stronger than the LP relaxation of IP1 (LP1) and propose an efficient column generation procedure to solve LP2, which is used within a branch-and-price algorithm to solve IP2. Within the branch-and-price procedure, we use a greedy 2-approximation algorithm to solve the pricing problem and only solve it exactly (via dynamic programming) when the heuristic algorithm fails to find a column to be added to the restricted master problem of LP2. We also keep track of variables that have been nonbasic for a certain number of iterations and remove them from the problem to maintain tractability. We discussed that branching on the variables of IP2 is not a viable option, we therefore consider branching on the variables of IP1. This strategy makes up for a much cleaner column generation process. We proposed and tested three branching strategies and observed the hybrid one to perform better.

We proposed and tested two heuristics for the CMFLP. The first is an LP

rounding heuristic that uses the fractional variables from the column generation procedure. The second one is a local search heuristic called 1-OptSwapBI, originally proposed for the MFLP, that uses the decomposition of IP1 into client and facility subproblems. In the MFLP, the client and facility subproblems are solvable in polynomial time. However, in the CMFLP, the client problem turns out to be the NP-Hard generalized assignment problem due to the capacity constraints.

The computational results underline the benefits and drawbacks of both formulations, and the heuristics. The increase in the total number of vertices naturally makes the problem harder to tackle. However, IP1 has more of a trouble handling the size of the problem than IP2, especially when the average number of clients assigned to a facility is small. In that case, the packing constraints in IP1 makes the problem significantly harder, which leads to LP2 dominating LP1 in terms of lower bound quality. The packing constraints cause difficulty for the local search heuristic as well. The local search heuristic performs worse as the average number of clients assigned to a facility decreases. On the other hand, the LP rounding heuristic both runs faster and provides good quality upper bounds under the same conditions. When we introduce a second type of facility, IP1 again has a more difficult time handling the added dimension compared to IP2. This suggests that introducing additional types of facilities will most likely make IP1 perform even worse. We suggest IP1 as the go-to formulation when the average number of clients assigned to a facility is large. However, as this number gets smaller and the number of vertices gets larger, IP2 becomes the better formulation. It is important to note that the performance of IP2 can be further improved by fine tuning the parameters and introducing other

column generation improvement techniques from the literature.

Bibliography

- V. Abhishek and K. Hosanagar. Optimal bidding in multi-item multislot sponsored search auctions. *Operations Research*, 61(4):855–873, 2013.
- B. Addis, G. Carello, and A. Ceselli. Exactly solving a two-level location problem with modular node capacities. *Networks*, 59(1):161–180, 2012.
- B. Addis, G. Carello, and A. Ceselli. Combining very large scale and {ILP} based neighborhoods for a two-level location problem. *European Journal of Operational Research*, 231(3):535–546, 2013.
- G. Aggarwal, A. Goel, and R. Motwani. Truthful auctions for pricing search keywords. In *Proceedings of the 7th ACM Conference on Electronic Commerce*, pages 1–7. ACM, 2006.
- S. Ahmadian, Z. Friggstad, and C. Swamy. Local-search based approximation algorithms for mobile facility location problems. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1607–1621, 2013.
- R. K. Ahuja, J. B. Orlin, S. Pallottino, M. P. Scaparra, and M. G. Scutellà. A multi-exchange heuristic for the single-source capacitated facility location problem. *Management Science*, 50(6):749–760, 2004.
- N. Anari, M. Fazli, M. Ghodsi, and M. Safari. Euclidean movement minimization. *Journal of Combinatorial Optimization*, 32(2):354–367, 2016.
- A. B. Arabani and R. Z. Farahani. Facility location dynamics: An overview of classifications and applications. *Computers & Industrial Engineering*, 62(1):408–420, 2012.
- A. Armon, I. Gamzu, and D. Segev. Mobile facility location: combinatorial filtering via weighted occupancy. *Journal of Combinatorial Optimization*, 28(2):358–375, 2012.
- M. Bateni, J. Feldman, V. Mirrokni, and S. C.-w. Wong. Multiplicative bidding in online advertising. In *Proceedings of the fifteenth ACM Conference on Economics and Computation*, pages 715–732. ACM, 2014.

- E. M. L. Beale and J. A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In *Proceedings of the Fifth International Conference on Operational Research*, pages 447–454, 1970.
- H. Beales. The value of behavioral targeting. *Network Advertising Initiative*, 2010.
- J. Berg, A. Greenwald, V. Naroditskiy, and E. Sodomka. A knapsack-based approach to bidding in ad auctions. In *Proceedings of the 2010 Conference on ECAI*, volume 215, pages 1013–1014. ECAI, 2010.
- A. Bingham, A. Bishop, P. Coffey, J. Winkler, J. Bradley, I. Dzuba, and I. Agurto. Factors affecting utilization of cervical cancer prevention services in low-resource settings. *Salud publica de Mexico*, 45:408–416, 2003.
- C. Borgs, J. Chayes, N. Immorlica, K. Jain, O. Etesami, and M. Mahdian. Dynamics of bid optimization in online advertisement auctions. In *Proceedings of the 16th International Conference on World Wide Web*, pages 531–540. ACM, 2007.
- D. G. Cattrysse and L. N. Van Wassenhove. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60(3): 260–272, 1992.
- A. Ceselli and G. Righini. A branch-and-price algorithm for the capacitated p-median problem. *Networks*, 45(3):125–142, 2005.
- C.-H. Chen and C.-J. Ting. Combining Lagrangian heuristic and ant colony system to solve the single source capacitated facility location problem. *Transportation Research Part E: Logistics and Transportation Review*, 44(6):1099–1122, 2008.
- M. J. Cortinhal and M. E. Captivo. Upper and lower bounds for the single source capacitated location problem. *European Journal of Operational Research*, 151(2): 333–351, 2003.
- E. Demaine, M. Hajiaghayi, H. Mahini, A. S. Sayedi-Roshkhar, S. Oveisgharan, and M. Zadimoghaddam. Minimizing movement. *ACM Transactions on Algorithms*, 5(3):30, 2009.
- K. Doerner, A. Focke, and W. J. Gutjahr. Multicriteria tour planning for mobile healthcare facilities in a developing country. *European Journal of Operational Research*, 179(3):1078–1096, 2007.
- M. E. Dyer. An $O(n)$ algorithm for the multiple-choice knapsack linear program. *Mathematical Programming*, 29(1):57–63, 1984.
- B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. Technical report, National Bureau of Economic Research, 2005.

- A. Farahat and M. C. Bailey. How effective is targeted advertising? In *Proceedings of the 21st international conference on World Wide Web*, pages 111–120. ACM, 2012.
- J. Feldman, S. Muthukrishnan, M. Pal, and C. Stein. Budget optimization in search-based advertising auctions. In *Proceedings of the 8th ACM Conference on Electronic Commerce*, pages 40–49. ACM, 2007.
- J. Feng, H. K. Bhargava, and D. M. Pennock. Implementing sponsored search in web search engines: Computational evaluation of alternative mechanisms. *INFORMS Journal on Computing*, 19(1):137–148, 2007.
- Z. Friggstad and M. R. Salavatipour. Minimizing movement in mobile facility location problems. *ACM Transactions on Algorithms*, 7(3):28, 2011.
- E. Geoffroy, A. D. Harries, K. Bissell, E. Schell, A. Bvumbwe, K. Tayler-Smith, and W. Kizito. Bringing care to the community: expanding access to health care in rural Malawi through mobile health clinics. *Public Health Action*, 4(4):252–258, 2014.
- A. Ghose and S. Yang. An empirical analysis of search engine advertising: Sponsored search in electronic markets. *Management Science*, 55(10):1605–1622, 2009.
- G. Guastaroba and M. G. Speranza. A heuristic for BILP problems: The single source capacitated facility location problem. *European Journal of Operational Research*, 238(2):438–450, 2014.
- M. H. Ha, L.-A. Bostel, N., and L.-M. Rousseau. An exact algorithm and a meta-heuristic for the multi-vehicle covering tour problem with a constraint on the number of vertices. *European Journal of Operational Research*, 226(2):211–220, 2013.
- R. Halper, S. Raghavan, and M. Sahin. Local search heuristics for the mobile facility location problem. *Computers & Operations Research*, 62:210–223, 2015.
- K. Holmberg, M. Rönnqvist, and D. Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113(3):544–559, 1999.
- G. Iyer, D. Soberman, and J. M. Villas-Boas. The targeting of advertising. *Marketing Science*, 24(3):461–476, 2005.
- H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004. ISBN 9783540402862.
- A. Klose. An LP-based heuristic for two-stage capacitated facility location problems. *The Journal of the Operational Research Society*, 50(2):pp. 157–166, 1999.

- A. Klose and S. Görtz. A branch-and-price algorithm for the capacitated facility location problem. *European Journal of Operational Research*, 179(3):1109–1125, 2007.
- H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
- E. Y.-H. Lin. A bibliographical survey on some well-known non-standard knapsack problems. *INFOR*, 36(4):274, 1998.
- L. A. N. Lorena and E. L. F. Senne. A column generation approach to capacitated p-median problems. *Computers & Operations Research*, 31(6):863–876, 2004.
- G. S. Lueker. Average-case analysis of off-line and on-line knapsack problems. *Journal of Algorithms*, 29(2):277–305, 1998.
- M. T. Melo, S. Nickel, and F. Saldanha da Gama. Dynamic multi-commodity capacitated facility location: a mathematical modeling framework for strategic supply chain planning. *Computers & Operations Research*, 33(1):181–208, 2006.
- S. Muthukrishnan, M. Pál, and Z. Svitkina. Stochastic models for budget optimization in search-based advertising. In *Internet and Network Economics*, pages 131–142. Springer, 2007.
- S. Nickel and F. Saldanha da Gama. *Location Science*, chapter Multi-Period Facility Location, pages 289–310. Springer, 2015.
- A. Pani. *Models for budget constrained auctions: An application to sponsored search & other auctions*. University of Maryland, College Park, 2010.
- D. Pisinger. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, 83(2):394–410, 1995.
- PwC Report. IAB internet advertising revenue report, april 2016. URL http://www.iab.com/wp-content/uploads/2016/04/IAB_Internet_Advertising_Revenue_Report_FY_2015-final.pdf.
- PwC Report. IAB internet advertising revenue report, april 2017. URL https://www.iab.com/wp-content/uploads/2016/04/IAB_Internet_Advertising_Revenue_Report_FY_2016.pdf.
- P. Rusmevichientong and D. P. Williamson. An adaptive algorithm for selecting profitable keywords for search-based advertising services. In *Proceedings of the 7th ACM Conference on Electronic Commerce, EC '06*, pages 260–269. ACM, 2006.
- M. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841, 1997.

- P. Sinha and A. A. Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27(3):503–515, 1979.
- F. Stefanello, O. C. B. de Araújo, and F. M. Müller. Matheuristics for the capacitated p-median problem. *International Transactions in Operational Research*, 22(1):149–167, 2015.
- J. E. Torres-Soto and H. Uster. Dynamic-demand capacitated facility location problems with and without relocation. *International Journal of Production Research*, 49(13):3979–4005, 2011.
- S. Tragantalerngsak, J. Holt, and M. Rönnqvist. An exact method for the two-echelon, single-source, capacitated facility location problem. *European Journal of Operational Research*, 123(3):473–489, 2000.
- H. R. Varian. Position auctions. *International Journal of Industrial Organization*, 25(6):1163–1178, 2007.
- V. Verter and S. D. Lapierre. Location of preventive health care facilities. *Annals of Operations Research*, 110(1-4):123–132, 2002.
- J. E. Weiss, M. R. Greenlick, and J. F. Jones. Determinants of medical care utilization: The impact of spatial factors. *Inquiry*, 8(4):50–57, 1971.
- J. Yan, N. Liu, G. Wang, W. Zhang, Y. Jiang, and Z. Chen. How much can behavioral targeting help online advertising? In *Proceedings of the 18th International Conference on World Wide Web*, pages 261–270. ACM, 2009.
- Z. Yang, F. Chu, and H. Chen. A cut-and-solve based algorithm for the single-source capacitated facility location problem. *European Journal of Operational Research*, 221(3):521–532, 2012.
- E. Zemel. An $O(n)$ algorithm for the linear multiple choice knapsack problem and related problems. *Information Processing Letters*, 18(3):123–128, 1984.
- Y. Zhou and V. Naroditskiy. Algorithm for stochastic multiple-choice knapsack problem and application to keywords bidding. In *Proceedings of the 17th International Conference on World Wide Web*, pages 1175–1176. ACM, 2008.
- Y. Zhou, D. Chakrabarty, and R. Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In *Internet and Network Economics*, pages 566–576. Springer, 2008.