Approximation Algorithms for Resource Allocation Optimization

by

Kewen Liao

A thesis submitted for the degree of
Doctor of Philosophy

School of Computer Science,
The University of Adelaide.

March, 2014

To my parents and grandparents,
for their endless support.

Contents

List of Figures

List of Tables

List of Acronyms

Acronym	*Meaning*
UFL	Uncapacitated Facility Location (problem)
KM	K-median (problem)
ILP	Integer linear program
$FTFL$	Fault-Tolerant Facility Location (problem)
$SCFL$	Soft Capacitated Facility Location (problem)
k-UFL	K-Uncapacitated Facility Location (problem)
$FTRA_\infty$	Unconstrained Fault-Tolerant Resource Allocation (problem)
$CFTRA_\infty$	Capacitated Unconstrained Fault-Tolerant Resource Allocation (problem)
RRA	Reliable Resource Allocation (problem)
$FTRA$	Constrained Fault-Tolerant Resource Allocation (problem)
k-$FTRA$	K-Constrained Fault-Tolerant Resource Allocation (problem)
QoS	Quality of service
FPT	Fixed parameter tractable
CDN	Content distribution/delivery network
PM	Physical machine
VM	Virtual machine
OR	Operations research
TCS	Theoretical computer science
CC	Computational complexity (theory)
VCO	Vertex cover optimization (problem)
VCD	Vertex cover decision (problem)
TM	Turing Machine
FSC	Finite state control
RAM	Random Access Machine
CU	Control unit
PC	Program counter
SAT	Boolean satisfiability (problem)

Acronym	*Meaning*
IS	Independent set (problem)
UVC	Unweighted vertex cover (problem)
LP	Linear program
CSC	Complementary slackness condition
$\mathcal{APX}$	Approximable
$\mathcal{PTAS}$	Polynomial time approximation scheme
$\mathcal{FPTAS}$	Full polynomial time approximation scheme
TSP	Traveling salesman problem
AP-reduction	Approximation preserving reduction
PCP	Probabilistically checkable proof (theorem)
VC	Weighted vertex cover (problem)
SC	Set cover (problem)
JV	An approximation algorithm by Jain and Vazirani [75] for UFL
MP	An approximation algorithm by Mettu and Plaxton [113] for UFL
MMSV	An approximation algorithm by Mahdian *et al.* [104] for UFL
JMS	An approximation algorithm by Jain *et al.* [74] for UFL
CRR	Clustered randomized rounding (algorithm) in [41]
CSGA	Cost scaling and greedy augmentation (procedures)
MYZ	An approximation algorithm by Mahdian *et al.* [106] for UFL
FLO	Facility Location with Outliers (problems)
$MLFL$	Multi-level Facility Location (problems)
OFL	Online Facility Location (problems)
DR	Dependent rounding (technique)
LC	Laminar clustering (technique)
CFL	Capacitated Facility Location (problem)
$HCFL$	Hard Capacitated Facility Location (problem)
IG	Integrity gap (of an integer linear program)
$CFLS$	Capacitated Facility Location (problem) with splittable demands
$CFLU$	Capacitated Facility Location (problem) with unsplittable demands
$UniFL$	Universal Facility Location (problem)

Acronym	*Meaning*
$HCFLU$	Hard Capacitated Facility Location (problem) with unsplittable demands
GAP	Generalized assignment problem
LR	Lagrangian relaxation (technique)
LMP	Lagrangian multiplier preserving (property)
$FTFA$	Fault-Tolerant Facility Allocation (problem)
$FTFP$	Fault-Tolerant Facility Placement (problem)
SG-1	A star-greedy algorithm for $FTRA_\infty$
PD-1	A primal-dual algorithm for $FTRA_\infty$
SG-2	An improved star-greedy algorithm for $FTRA_\infty$
PD-2	An improved primal-dual algorithm for $FTRA_\infty$
MRR	Minimum reliability requirement
VLSI	Very-large-scale integration
PD-3	A primal-dual algorithm for RRA
APD-3	An accelerated primal-dual algorithm for RRA
IDF	Inverse dual fitting (technique)
ULPR	A unified LP-rounding algorithm for $FTRA$
PD-4	A primal-dual algorithm for $FTRA$
APD-4	An accelerated primal-dual algorithm for $FTRA$
SOC	Sum of contributions
AGA	Acceleration of greedy augmentation (procedure)
PK	Procedures for solving k-$FTRA$
BS	Binary search (procedure)
GP	Greedy pairing (procedure)
RR	Randomized rounding (procedure)
e.g.	For example
i.e.	That is
etc.	And so on
w.r.t.	With respect to
w.l.o.g.	Without loss of generality
s.t.	Such that

Abstract

Nowadays, data storage, server replicas/mirrors, virtual machines, and various kinds of services can all be regarded as different types of resources. These resources play an important role in today's computer world because of the continuing advances in information technology. It is usual that similar resources are grouped together at the same site, and can then be allocated to geographically distributed clients. This is the resource allocation paradigm considered in this thesis. Optimizing solutions to a variety of problems arising from this paradigm remains a key challenge, since these problems are $\mathcal{NP}$-hard.

For all the resource allocation problems studied in this thesis, we are given a set of sites containing facilities as resources, a set of clients to access these facilities, an opening cost for each facility, and a connection cost for each allocation of a facility to a client. The general goal is to decide the number of facilities to open at each site and allocate the open facilities to clients so that the total cost incurred is minimized. This class of the problems extends the classical $\mathcal{NP}$-hard facility location problems with additional abilities to capture various practical resource allocation scenarios.

To cope with the $\mathcal{NP}$-hardness of the resource allocation problems, the thesis focuses on the design and analysis of approximation algorithms. The main techniques we adopt are linear programming based, such as primal-dual schema, linear program rounding, and reductions via linear programs. Our developed solutions have great potential for optimizing the performances of many contemporary distributed systems such as cloud computing, content delivery networks, Web caching, and Web services provisioning.

Declaration

I, Kewen Liao, certify that this work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. In addition, I certify that no part of this work will, in the future, be used in a submission for any other degree or diploma in any university or other tertiary institution without the prior approval of the University of Adelaide and where applicable, any partner institution responsible for the joint-award of this degree.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

The author acknowledges that copyright of published works contained within this thesis (as listed on Page x) resides with the copyright holder(s) of those works.

I also give permission for the digital version of my thesis to be made available on the web, via the University's digital research repository, the Library catalogue, and also through web search engines, unless permission has been granted by the University to restrict access for a period of time.

Signature

Date

Preface

During my PhD study at the University of Adelaide from 2009 to 2013, I have produced four conference papers and two journal articles related to this thesis (see my homepage at `http://cs.adelaide.edu.au/~kewen` for my bio with a complete list of publications). The thesis topic is theoretical in nature and based on the content presented in the following papers.

Conference Publications:

- Kewen Liao and Hong Shen. Unconstrained and constrained fault-tolerant resource allocation. In *Proceedings of the 17th annual international conference on computing and combinatorics (COCOON)*, pages 555–566, Dallas, Texas, USA, 14-16 August 2011. Springer-Verlag, Berlin

- Kewen Liao and Hong Shen. Fast fault-tolerant resource allocation. In *Proceedings of 12th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 231–236, Gwangju, Korea, October 20-22 2011. IEEE

- Kewen Liao and Hong Shen. Approximating the reliable resource allocation problem using inverse dual fitting. In *Proceedings of the Eighteenth Computing: The Australasian Theory Symposium (CATS)*, page Vol. 128, Melbourne, Australia, January-February 2012. ACS, Sydney

- Kewen Liao, Hong Shen, and Longkun Guo. Improved approximation algorithms for constrained fault-tolerant resource allocation. In *Fundamentals of Computation Theory (FCT) - 19th International Symposium*, pages 236–247, Liverpool, UK, August 19-21 2013. Springer-Verlag, Berlin (extended version invited to the special issue of Theoretical Computer Science)

Journal Publications:

- Kewen Liao and Hong Shen. Lp-based approximation algorithms for reliable resource allocation. *The Computer Journal*, 57(1):154–164, 2014

- Kewen Liao, Hong Shen, and Longkun Guo. Constrained fault-tolerant resource allocation. *Theoretical Computer Science*, submitted in December 2013, at `http://arxiv.org/abs/1208.3835`, currently under review

Acknowledgments

This PhD study was a real challenge for me, and overcoming this challenge would not have been possible without the support of many people.

First and foremost, I would like to thank my principal supervisor Prof. Hong Shen, for his great supervision over the past few years. I am especially grateful for his trust in me working on tough theoretical problems. Without his constant advice and guidance, this thesis would not be even completed.

I am truly indebted and thankful to my supervisor A/Prof. Michael Sheng. He was also the supervisor of my honors thesis. He is the one who introduced me to research, inspired and encouraged me to pursue a PhD during my honors year. Without him, this thesis might not even exist.

I am sincerely grateful to my supervisor Emeritus Prof. Zbigniew Michalewicz, for his kind help, support, and encouragement in every aspect of my research.

I owe a huge debt of gratitude to my parents and grandparents, for their unconditional love, support, and encouragement. My most heartfelt thanks go to my parents for their spiritual and financial support during my eight years of study in Australia.

I would like to thank some of my colleagues and friends: Changjian, Donglai, Jeff, Lei, Li, Lina, Longkun, Mike, Scott, Shihong, Sim, Xiang, Xiaoqiang, Yidong, Yihong, Yong, Yongrui, Denny, Ke, Leo, Nick, Su, White, Xiaoming, Yibing, Yuguo, and Zilang, for their accompany, inspiration, and influence during my PhD journey.

Finally, I thank the University of Adelaide for providing me scholarship and the School of Computer Science for financially supporting my conference travels. I would also like to express my gratitude to the anonymous examiners of this thesis.

Chapter 1

Introduction

In real life, many economic problems of great practical importance are about how to strategically choose the locations of sites to open facilities for meeting the demands of users, customers or clients[1]. Traditionally, in operations research, such problems are called *facility location* problems. The applications of these problems/models widely appear in supply chain management, where facilities can be manufacturing plants, processing units, distribution centers, and warehouses. More recently, in the fields of theoretical computer science and computer networks, facilities are often regarded as nodes, base stations, servers, databases, contents, and services. There are different classifications of facility location problems. If a facility can only serve up to a certain number of users, the problem is *capacitated*. Without this limitation, the problem is called *uncapacitated*. If the site and client locations are finite, the problem is *discrete*. Otherwise, one may select locations anywhere from a plane or higher dimension and the problem becomes *continuous*. In this thesis, we will restrict our attention to the discrete problems (see the book [116] for an extensive survey). Typically, these problems are combinatorial optimization problems, in which the objectives are to find an extreme solution (usually with a maximum or minimum cost) from a large feasible solution space. Unfortunately, many of the problems are $\mathcal{NP}$-hard and difficult to solve optimally (see Chapter 2 for computational complexity background). These include the following fundamental discrete *Uncapacitated Facility Location* and (uncapacitated) *k-median* problems.

Definition 1.0.1. In the Uncapaciated Facility Location (UFL) problem, we are given a set of sites $\mathcal{F}$ and a set of clients $\mathcal{C}$. The cost of opening a facility at site $i \in \mathcal{F}$ is f_i. For a client $j \in \mathcal{C}$, the cost of connecting a facility at i to

[1]We will use these terms interchangeably in the text.

j is c_{ij}. Every client has to use one open facility. The objective is to select a subset of sites to open facilities and assign clients to the open facilities, such that the total facility opening and connection/service cost is minimized.

Definition 1.0.2. In the k-median (KM) problem, we are given a set of sites $\mathcal{F}$ and a set of clients $\mathcal{C}$. For a client $j \in \mathcal{C}$, the cost of connecting a facility at $i \in \mathcal{F}$ to j is c_{ij}. Every client has to use one open facility. The objective is to open at most k facilities and assign clients to the open facilities, such that the total connection cost is minimized.

Note that the KM problem differs from UFL in that an upper bound of k facilities[2] need to be selected and there is no facility opening cost involved. Also, if $\mathcal{F} = \mathcal{C}$, both problems are *complete*[3] and can be viewed as different clustering problems. Moreover, although identical, we presented the above definitions in a slightly different way[4] from the conventional definitions in [116, 129], for the purpose of naturally leading to our focused research problems below. In Chapter 3, we will revisit the UFL and KM problems in more detail.

1.1 Research problems

Nowadays, data storage, server replicas/mirrors, virtual machines, and various kinds of services can all be regarded as different types of resources. It is usual that similar resources are grouped together at the same site, and can then be allocated to geographically distributed clients. In this thesis, we address various *resource allocation* problems built on UFL. These problems reveal additional practical issues such as fault-tolerance, reliability, facility service capacity, and constraints on the number of available facilities. For the ease of illustration, we capture the problems by a unified resource allocation model as shown in Figure 1.1.

Correspondingly, the model is encoded by the following unified integer linear program (ILP) formulation (see Chapter 2 for LP background).

[2]It is easy to see that this covers the case where exactly k facilities need to be selected.

[3]The complete problems are special cases of the bipartite problems (as in our definitions) in which there are two separate sets of sites and clients. They usually have weaker hardness results (see Chapter 3 for more details).

[4]In fact the difference is, conventionally $\mathcal{F}$ is denoted as a set of facilities rather than sites. However, it is identical to our definitions since every client has to use only one facility, and therefore at most one facility is needed to open at each site in both UFL and KM.

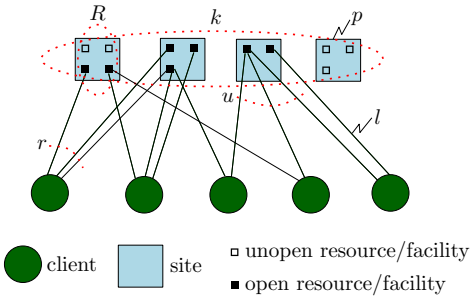

Figure 1.1: A unified resource allocation model with input parameters $\boldsymbol{p}$, $\boldsymbol{l}$, $\boldsymbol{r}$, k, $\boldsymbol{R}$, and $\boldsymbol{u}$ (to be explained in detail later) that capture various practical issues in resource allocation.

$$
\begin{array}{lll}
\text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij} x_{ij} & (1.11) \\
\text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} p_i l_{ij} x_{ij} \geq r_j & (1.12) \\
& \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 & (1.13) \\
& \sum_{i\in\mathcal{F}} y_i \leq k & (1.14) \\
& \forall i \in \mathcal{F} : y_i \leq R_i & (1.15) \\
& \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} x_{ij} \leq u_i y_i & (1.16) \\
& \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \in \mathbb{Z}^+ & (1.17) \\
& \forall i \in \mathcal{F} : y_i \in \mathbb{Z}^+ & (1.18)
\end{array}
\tag{1.1}
$$

Specifically, in the model, we have a set of sites $\mathcal{F}$ and a set of clients $\mathcal{C}$. At each site $i \in \mathcal{F}$, a number of facilities with f_i as costs can be opened to serve as resources. There is also a connection/service cost c_{ij} between each client $j \in \mathcal{C}$ and site i. This cost is usually proportional to the distance between i and j. The general goal (represented by the objective function (1.11)) is to optimally allocate/open a certain number of facilities from each i, such that every client j's requests are served while minimizing the sum of the facility opening and client connection costs. Notice that the two cost terms in the objective function can be scaled without altering the solution quality (see Chapter 3 for reason). This brings more applicability to the model.

In the unified model, we are also given a set of input parameters $\boldsymbol{p}$, $\boldsymbol{l}$, $\boldsymbol{r}$, k, $\boldsymbol{R}$, and $\boldsymbol{u}$. For each site i and client j, p_i is the reliability of the facilities at i, l_{ij} the reliability of the access link between i and j, r_j the reliability requirement or the required number of connections of j, R_i the maximum number of facilities allowed to open at i, and u_i the maximum number of clients that a facility at i can serve. In addition, k is a single parameter denoting the total number of facilities allowed to open across all sites. Besides these inputs, in

the formulation, the common integer solution we aim to find is $(\boldsymbol{x}, \boldsymbol{y})$[5] where the variable y_i denotes the number of facilities to open at i, and x_{ij} the number of connections to establish between i and j.

Subject to different input parameter values and constraints[6] (displayed as inequalities $(1.12) - (1.18)$), the unified model covers both some of the classical facility location problems and the resource allocation problems we concentrate on. All of the problems are listed and summarized below.

Classical facility location problems:

- Uncapacitated Facility Location (UFL): $\forall i, j : p_i, l_{ij}, r_j = 1$; without $(1.14) - (1.16)$.
- Fault-Tolerant Facility Location ($FTFL$): $\forall i, j : p_i, l_{ij}, R_i = 1$; $r_j \geq 1$; without (1.14), (1.16).
- Soft Capacitated Facility Location ($SCFL$): $\forall i, j : p_i, l_{ij}, r_j = 1$; $u_i \geq 1$; without (1.14), (1.15).
- k-median (clustering): $\forall i, j : \mathcal{F} = \mathcal{C}$; $f_i = 0$; $p_i, l_{ij}, r_j = 1$; $k \geq 1$; without (1.15), (1.16).
- k-UFL: $\forall i, j : p_i, l_{ij}, r_j = 1$; $k \geq 1$; without (1.15), (1.16).

Resource allocation problems:

- Unconstrained Fault-Tolerant Resource Allocation ($FTRA_\infty$):
  - $\forall i, j : p_i, l_{ij} = 1$; $r_j \geq 1$; without $(1.14) - (1.16)$.
- Capacitated $FTRA_\infty$ ($CFTRA_\infty$) : $\forall i, j : p_i, l_{ij} = 1$; $r_j, u_i \geq 1$; without (1.14), (1.15).
- Reliable Resource Allocation (RRA):
  - $\forall i, j : p_i, l_{ij} \in [0, 1]$; $r_j \in [0, +\infty)$; without $(1.14) - (1.16)$.
- Constrained Fault-Tolerant Resource Allocation ($FTRA$):

[5] In the facility location problems generated from our unified model, the range of $(\boldsymbol{x}, \boldsymbol{y})$ is actually restricted to be binary integers $\{0, 1\}$ (by the input parameters) instead of non-negative integers $\mathbb{Z}^+$.

[6] For smoothness of presentation, more detailed explanations about the parameters and constraints in different resource allocation problems are postponed to the next chapters.

– $\forall i, j : p_i, l_{ij} = 1$; $r_j, R_i \geq 1$; without (1.14), (1.16).

- *k-FTRA*: $\forall i, j : p_i, l_{ij} = 1$; $r_j, k, R_i \geq 1$; without (1.16).

The above problems are either *metric* or *non-metric* depending on whether the connection costs c_{ij}'s form a metric or not. The metric $\boldsymbol{c}$ is nonnegative, symmetric and satisfies the triangle inequality: that is, $\forall i \in \mathcal{F}, j \in \mathcal{C} : c_{ij} = c_{ji} \geq 0$, and $\forall i, j, k \in \mathcal{F} \cup \mathcal{C} : c_{ij} + c_{jk} \geq c_{ik}$. Another common interpretation of the triangle inequality used in our mathematical proofs is $\forall i, i' \in \mathcal{F}$ and $j, j' \in \mathcal{C} : c_{ij} + c_{i'j} + c_{i'j'} \geq c_{ij'}$. The metric case is important since in a network, the connection costs are often measured to be proportional to the shortest path routing distances. It is also not hard to see that the shortest paths satisfy the metric conditions. In addition, the problems are *uniform* if $\forall j, j' \in \mathcal{C} : r_j = r_{j'}$ and *nonuniform/general* otherwise. Unless elsewhere specified, we focus on the metric case of the problem. Furthermore, the resource allocation problems are $\mathcal{NP}$-hard. One reason is that the discrete ILP formulation is generally $\mathcal{NP}$-hard to solve. The other explanation is that both the non-metric and metric UFL problems (the most basic forms in our resource allocation model) have been proven to be $\mathcal{NP}$-hard via reductions from the fundamental set cover problem (see Chapter 3 for more details).

The classical facility location problems restrict at most one facility to open at each site, and connection requests of a client must be served by facilities from different sites. However, the resource allocation problems we consider allow multiple facilities to open at every site (the 'grouped' property of resources), and clients' connection requests to be served by facilities within the same site. These problems are more realistic since a client may access multiple resources at the same site in parallel. Among the resource allocation problems, $FTRA_\infty$ does not limit the number of resources to use at each site (so provider can always add them) while $FTRA$ and k-$FTRA$ constrain the amount of the available resources to allocate. RRA models the reliability in resource allocation which is an important quality of service (QoS) attribute. $CFTRA_\infty$ is a natural variant of $FTRA_\infty$ that takes into account the resource capacity.

1.2 Thesis aims and impact

This thesis aims to devise effective and efficient algorithms to solve our listed resource allocation problems from the theoretical point of view. To cope with the $\mathcal{NP}$-hardness of an optimization problem, typical approaches are exact

algorithms, approximation algorithms and meta-heuristic methods. An exact algorithm finds the optimal solution but usually takes exponential time with respect to the problem's input size[7]. On the other hand, an approximation algorithm (see Chapter 2 for more details) runs in polynomial time and settles for a near-optimal solution that is guaranteed not to be too far from the optimal. Unlike these two approaches, a meta-heuristic method does not have a guarantee on the solution found. However, it is able to explore a large space of candidate solutions and sometimes produce good empirical results. For the $\mathcal{NP}$-hard resource allocation problems, we mainly deal with the design and analysis of approximation algorithms. In particular, we measure the *effectiveness* of an algorithm by looking at how close the solution found is to the optimal, and the *efficiency* by the Big-O complexity in terms of the problem input sizes $n = |\mathcal{F}| + |\mathcal{C}|$ and $m = |\mathcal{F}| \times |\mathcal{C}|$. In a combinatorial optimization problem, these two measurements are often conflicting with each other, which means we usually need to seek for a tradeoff between them. For the resource allocation problems, the reasons why we focus on approximation algorithm are:

- The metric case of the problems admit constant approximation factors, implying that an approximation algorithm is likely to be effective and efficient. This is also because the approximation ratio is the worst case guarantee. In practice, the worst instance is rarely encountered and most likely the approximate solution is within 5% of the optimal (as shown in [73, 71] for UFL).

- The approximation algorithm brings mathematical rigor to the study of heuristics. It offers provable approximation ratio (effectiveness) and runtime (efficiency) on all or part of the problem instances [141].

- Algorithm design often focuses on idealized models first rather than the 'real world' applications. An approximation algorithm for a simpler model often gives us some idea on how to invent a heuristic that performs well in practice for an actual complex problem [141].

The impact of solving the resource allocation problems lies in both their broad application domains and great benefits. Some of the application domains are shared with facility location (see Chapter 3 for more details) since it is a

[7] A fixed parameter tractable (FPT) algorithm is also one type of the exact algorithm. After fixing an input parameter, the algorithm usually runs in polynomial in the input size and exponential or worse in the parameter. For a text book on FPT, see [49].

special case of resource allocation. The universality of resource allocation also makes it applicable to the fields of distributed computing, cloud computing, and computer networking. In the following, we only describe two concrete examples to see its contemporary applications in content distribution networks and cloud computing.

A content distribution network (CDN) offers reduced client access latency by replicating data at various nodes of a network. Each node usually consists of multiple surrogate servers that host data objects and share the duty of serving clients together [144]. With more intelligent replication schemes, clients are also able to achieve fault tolerance, parallel access, and reliability of data usage. Thus, it further improves the CDN's end-user experience. The resource allocation model provides such a scheme by treating nodes as sites and servers as facilities/resources. The objective is to minimize the total resource deployment and network access costs while satisfying the clients' data requirements. In particular, the model can achieve both intra-site and inter-site fault tolerance or resource reliability. This effectively prevents the whole site and individual facility failures. In addition, the capacitated model offloads the servers and improves the performance of CDN by taking into account the servers' workload limits. The resource constrained model increases the server utilization under harsh environments.

In cloud computing, cloud providers own data centers with data storage and virtual machines as resources. The geographically distributed clients can then easily lease these resources on-demand, without paying for the expensive physical machine bundles that host them. This new resource sharing paradigm inevitably brings up many optimization problems. Two important ones related to resource allocation are the replication of data storage across data centers to achieve high levels of fault tolerance [3], and the optimal placement and allocation of physical machines (PMs) and virtual machines (VMs) [112]. Similar to the replication in CDN, if we treat data centers as sites and data storage locations as facilities, the model provides an optimal replication scheme that achieves required levels of fault tolerance or reliability. Also, for PM/VM placement and allocation, if a PM/VM is regarded as a resource at a data center, the model then offers a cost optimal strategy for the cloud providers as indicated in [62, 32, 61]. Furthermore, the capacitated and constrained models can certainly power up the cloud system's capabilities.

The invention of novel, effective, and efficient algorithms for various resource allocation models would add new knowledge to the theoretical computer

science and operations research fields. This may also bring in more insights for solving the classical facility location problems.

1.3 Thesis results

In this thesis we present approximation algorithms and their analyses for the resource allocation problems.

For the $FTRA_\infty$ problem, we present three pseudo-polynomial time algorithms for its uniform case. The runtimes of these algorithms can be turned into strongly polynomial with some carefully designed acceleration heuristics. For brevity of presentation, we postpone the discussions of the runtime improvements to the chapters dedicated to the RRA and $FTRA$ problems. More specifically, we first present two star-greedy algorithms and analyze them via two equivalent prima-dual algorithms to achieve approximation factors of 1.861 and 1.61 respectively. Later, by applying cost scaling and greedy augmentation techniques we get an improved ratio of 1.52. This result significantly improves the 1.861-approximation in [145]. For the general case of the problem, we show a simple reduction to UFL. We also study the $CFTRA_\infty$ problem and obtain constant factors of 2.89 and 3.722 using Lagrangian relaxation.

We initiate the study of the RRA problem towards the provision of more robust fault-tolerance to the resource allocation paradigm. For this problem, two equivalent primal-dual algorithms are presented. The second, which runs in quasi-quadratic time, is a significant improvement of the first. For the approximation ratio analysis, RRA is a much harder problem than $FTRA_\infty$ and the main difficulty we overcome is to deal with fractional reliabilities. We apply the inverse dual fitting technique introduced in [145] as a central idea. Our analysis further elaborates and generalizes this generic technique, which naturally yields constant approximation factors of $2 + 2\sqrt{2}$ and 3.722. We obtain even better ratios through novel reductions to UFL and $FTRA_\infty$. The reductions demonstrate some useful and generic linear programming techniques.

For the $FTRA$ problem, we first develop a unified LP-rounding algorithm which can directly solve $FTRA$, $FTRA_\infty$ and $FTFL$ with the same approximation ratio of 4. Then, we show the reduction of $FTRA$ to $FTFL$ using an instance shrinking technique. This implies that these two problems share the same approximability in weakly polynomial time. Hence, from the $FTFL$ result of [31], a ratio of 1.7245 is achieved. For the uniform case of the problem, we first present a pseudo-polynomial time primal-dual algorithm. To

analyze the algorithm, we adopt a constraint-based analysis, by which we get a ratio of 1.61. Later, with two carefully designed acceleration heuristics, we obtain a strongly polynomial time algorithm having a ratio of 1.52 in runtime $O(n^4)$. The uniform k-$FTRA$ problem is also studied where we give the first constant-factor approximation algorithm with a ratio of 4. The algorithm relies on Lagrangian relaxation and a polynomial time greedy pairing procedure we develop for efficiently splitting sites into paired and unpaired facilities.

To sum up, we believe that our developed novel algorithmic ideas will be useful for solving other resource allocation variants and facility location problems.

1.4 Thesis structure

The rest of the thesis is organized as follows. In Chapter 2, we provide some necessary theory background related to our problems. This includes the basics of computational complexity, linear and integer linear programming, and approximation algorithms. Chapter 3 reviews the state of art techniques and results of some important discrete facility location problems with focus on UFL. In Chapter 4, we present our results on $FTRA_\infty$ and $CFTRA_\infty$. We initiate the study of the RRA problem in Chapter 5. Chapter 6 is dedicated to solve the constrained resource allocation problems $FTRA$ and k-$FTRA$. We conclude the thesis in Chapter 7 with a brief summary of our results and future research directions.

Chapter 2

Preliminaries

Optimization is the process of making the best of something, the action of rendering optimal and the state of being optimal. *Operations research* (OR) is an applied mathematics subject that uses advanced analytical methods to achieve optimization for complex business and industry problems. The probably most important tool in OR is *linear programming,* which is used for modeling optimization problems as linear programs and deriving their optimal solutions.

One important part of the field of *theoretical computer science* (TCS) influenced by OR[1] focuses on solving discrete combinatorial optimization problems. This is an important class of optimization problems which abstractly models thousands of real-life problems. Depending on different combinatorial structures, these problems can be classified as covering, packing, matching, coloring, scheduling and routing problems. Facility location problems can be viewed as covering problems because the clients' requirements need to be covered. Combinatorial optimization problems are usually modeled by integer linear programs. In general, an integer linear program's optimal solution cannot be found in polynomial time, since *integer linear programming* is $\mathcal{NP}$-hard. Therefore, for these problems, we often develop *approximation algorithms*, settling for near-optimal solutions. Another important part of TCS is *computational complexity*[2]. Computational complexity concentrates on classifying and

[1] In fact, some problems studied in TCS and OR are overlapped. However, TCS usually has a different treatment/analysis and pays more attention to the abstract or mathematical aspects of computing.

[2] Understanding computational complexity is extremely important since it not only tells what kind of algorithm can be developed for a problem, but also explains whether a problem is exactly solvable or approximatable within a fix amount of resources (e.g., memory, time, etc.).

comparing the difficulty/complexity of *computational problems*. A computational problem is an abstract mathematical problem that models many real-world problems for computers to solve. A combinatorial optimization problem is also a computational problem.

In this chapter, we will briefly review the background knowledge of the two parts of TCS relevant to our thesis topic. We start from the fundamental computational complexity theory. Then, we look at the basics of the tools — linear programming and integer linear programming. Lastly, we introduce the concept of approximation algorithm and discuss its design methods. For simplicity, throughout this chapter we use the following (weighted) *vertex cover* problem as a running example.

Definition 2.0.1. Let $G = (V, E)$ be an undirected graph with weights $w_i \geq 0$ associated with each vertex $i \in V$, the vertex cover problem asks for a vertex subset $V' \subseteq V$ such that every edge $e \in E$ has at least one endpoint in V' (definition of a vertex cover) and $\sum_{i \in V'} w_i$ is minimized.

Note that this problem is an optimization problem. We can assume the weights to be non-negative integers since rationals can be transformed to integers. Also, if $\forall i \in V : w_i = 1$, the problem becomes unweighted and the objective changes to find a minimum cardinality vertex cover.

2.1 Computational complexity

2.1.1 Computational problems

Computational complexity (CC) theory is established on computational problems. In TCS, such a *problem* is an infinite collection of problem (input) *instances*. Each instance has a set of *solutions*. Two typical types of the problems are *optimization problems* and *decision problems*.

Optimization problems. An optimization problem consists of a set of input instances, a set of *constraints* which defines feasibility conditions of a solution, an *objective function* that assigns a value to a feasible solution of any instance, and a *goal* of either maximizing or minimizing the objective function. For example, an instance of the above vertex cover optimization (VCO) problem consists of a graph G and its weighs vector $\boldsymbol{w}$, denoted by a tuple $\langle G, \boldsymbol{w} \rangle$. A constraint is that all the edges must be covered by the selected vertex set V' so that V' is a feasible solution. The objective function is the total weight of the

cover $\sum_{i \in V'} w_i$ and the goal is to minimize this function. For a given instance, the objective function gives an *objective value*, and the goal corresponds to the *optimal solution*. Many optimization problems of real practical interest are modeled by linear programs.

Decision problems. An optimization problem can be cast as a decision problem by imposing a bound on the objective value. For instance, the vertex cover decision (VCD) problem can be stated as follows.

Definition 2.1.1. Given an undirected graph $G = (V, E)$ with weights $w_i \geq 0$ associated with each vertex $i \in V$ and an integer parameter W, is there a vertex cover with $\sum_{i \in V'} w_i \leq W$?

The answer to a decision problem is simply 'yes' or 'no', or binary '1' or '0'. A decision problem is defined by a set of instances, in which there are subsets of 'yes' instances and 'no' instances. Take the VCD problem as an example, an instance can be represented by $\langle G, \boldsymbol{w}, W \rangle$, so the set of instances are different weighted graphs with different parameters W. A graph with cover weight no more than W is an 'yes' instance and otherwise it is a 'no' instance. CC focuses on the study of decision problems since they are easier to deal with when classifying problem complexity. Also, as shown later, the theory of $\mathcal{NP}$-completeness directly applies to decision problems rather than optimization problems. This is natural since an optimization problem is not easier than its corresponding decision problem. Hence, if we want to show an optimization problem is 'hard', we only need to address the hardness of its decision problem. For example, we can solve the VCD problem by solving VCO. This is because once a minimum cover V' is found for an instance, we can easily compute the minimum $W' = \sum_{i \in V'} w_i$ and compare W' with W to determine the answer to VCO. Thus, the CC results established on decision problems also have complexity implications for optimization problems.

2.1.2 Languages, machine models, and algorithms

In CC, there are formal definitions for decision problems, machine/computation models and algorithms. A decision problem P can be view as a formal *language* L defined over a finite alphabet Σ. Typically, $\Sigma = \{0, 1\}$. Then, Σ^* is a set of finite binary strings based on Σ and $L \subseteq \Sigma^*$. This language framework is reasonable since input instances of P are normally encoded as binary strings in Σ^*. In addition, language L represents the set of 'yes' instances of P. Thus,

it is conventional to regard a decision problem as L and if an instance $x \in L$, x is an 'yes' instance, otherwise $x \in \Sigma^* \backslash L$ is an 'no' instance.

The formal language framework is established at a lower level for machine models to solve decision problems. The most important model is *Turing Machine* (TM), proposed by Turing [137]. TM is a theoretical model of modern computers. It formally defines some algorithmic concepts since if a problem can be solved by an algorithm, there exists a TM that solves the problem. A general TM model [16] has two components: a finite state control (FSC) and an infinite tape. FSC has a number of states controlling the computation logic. Two special states are 'accept' and 'reject' which give answer to a problem and halt the machine. Note that a TM may not halt forever. In the computability theory of TCS, TM is used to give proofs for the problems that do not have algorithms. The infinite tape is the place to hold and process the initial input string belonging to Σ^*. It has a read/write head that moves to either left or right and performs simple operations such as overwriting a value on the tape according to a set of transition rules. The transition rules are mathematically defined on a TM to mimic an algorithm to solve a decision problem: given an instance $x \in \Sigma^*$, is $x \in L$? If the rules are *deterministic*, that is, the machine's next FSC state and head movement subject to the tape content are fixed, it is a deterministic TM. Otherwise, the TM is *non-deterministic*. The number of operations a TM performs gives a measure of the algorithm's *running time.* The other important model is *Random Access Machine* (RAM) [48]. In comparison to TM, RAM more closely models the running time and space usage of a common computer. A RAM has a control unit (CU), program counter (PC) and a set of registers. PC and registers control the program logic which is simpler than the tape in TM. CU executes a statement at a time and provides a better control over running time and space. However, for a problem like analyzing the runtime of multiplying two numbers, it is more meaningful to use the measure of TM, since RAM regards multiplication as a single elementary operation. In this thesis, for the algorithm's runtime analysis, we follow the convention of using the RAM model. Notice that an algorithm in one model could be easily turned into an algorithm in another model.

At a high level, an algorithms is a procedure or sequence of instructions designed to solve a specific problem. For a computational problem, the *size* of an instance x (denoted by $|x|$) is often measured by the number of bits necessary to encode x. Take the VCO problem as an example, the size is $O\left(|V|^2\right)$ since a graph can be encoded by a binary adjacency matrix. For an algorithm to

solve a problem, its running time is often expressed in terms of the Big-O notation $O(.)$[3]. The Big-O measure essentially provides a runtime upper bound. Further, we say an algorithm for solving an instance x runs in *polynomial time* if there exists a polynomial $p(.)$ such that the running time of the algorithm is $O(p(|x|))$. For instance, the polynomial function might be $p(n) = n^k$ for some fixed $k \in \mathbb{N}$ where $\mathbb{N}$ denotes the set of nonnegative natural numbers. TCS generally regards an algorithm as efficient if it runs in polynomial time. This is in opposite to an inefficient algorithms that runs in exponential time ($O\left(2^{|x|}\right)$ for example). In the thesis, we distinguish among pseudo-, weakly and strongly polynomial time algorithms. In general, a pseudo-polynomial time algorithm runs in polynomial in the numeric input parameters (e.g., $O(W)$ for the VCO problem). A weakly polynomial time algorithm runs in logarithmic of the parameters ($O(\log W)$). A strongly polynomial time algorithm's runtime does not depend on the parameters ($O\left(|V|^2\right)$).

2.1.3 $\mathcal{P}$ and $\mathcal{NP}$

With the basic concepts of problems, languages, machine models, and algorithms, we are now ready to see several important complexity classes for decision problems. Later, in section 2.3, we shall see some other complexity classes for optimization problems. A complexity class contains a set of problems/languages with similar difficulties for appropriate algorithms to solve. Two basic complexity classes are $\mathcal{P}$ and $\mathcal{NP}$. $\mathcal{P}$ is a set of decision problems having (exact) polynomial time algorithms. Equivalently, we say a decision problem L is in $\mathcal{P}$ iff there is a deterministic TM that decides L and halts in polynomial number of steps. Before introducing the class $\mathcal{NP}$, we need the notions of *certifier* and *certificate.* A two-argument algorithm $C(.,.)$ is a certifier for a problem L if and only if for every $s \in L$ there exists a string s' such that $C(s, s') =$ 'yes'. The string s' is the certificate of s. In addition, C is efficient if it runs in polynomial time. The certificate s' is short if its size is bounded by a polynomial in the size of the input string s. For example, a short certificate of any VCD instance $\langle G, \boldsymbol{w}, W\rangle$ is a set of vertices V''. An efficient certifier simply checks against the input instance to see whether $\sum_{i\in V''} w_i \leq W$ and V'' forms a valid cover. We can then define the class $\mathcal{NP}$ as a set of decision problems having efficient certifiers and short certificates. Equivalently, a decision problem L is in $\mathcal{NP}$ iff there is a non-deterministic TM that decides L

[3]Consult the fundamental algorithm book [88] for the definitions of different runtime measures including Big-O.

and halts in polynomial number of steps. About the set relationship of $\mathcal{P}$ and $\mathcal{NP}$, it is not hard to see $\mathcal{P} \subseteq \mathcal{NP}$ since a certifier for a $\mathcal{P}$ problem can be easily constructed from a polynomial time algorithm. Nevertheless, determining whether $\mathcal{P} = \mathcal{NP}$ or not is one of the central open problem in computer science and mathematics. It is listed as one of the Millennium Prize Problems.

There is a subclass of problems which represents the hardest problems in $\mathcal{NP}$. These are the $\mathcal{NP}$-complete problems. Before defining this subclass, we need the concept of *polynomial time reduction* between two problems. Given problems L_1 and L_2, we say L_1 is polynomial time reducible to L_2 if there exists a polynomial time reduction algorithm $R(.)$ such that $\forall x \in \Sigma_1^* : x \in L_1$ iff $R(x) \in L_2$. Thus, R polynomially transforms an instance x to another instance $R(x) \in \Sigma_2^*$ with the same answer. This definition implies four things: i) we can solve L_1 by combining R and an algorithm for L_2; ii) if we have an efficient algorithm for L_2 then we have an efficient algorithm for L_1; iii) if there is no efficient algorithm for L_1, then there is also no efficient algorithm for L_2; iv) *transitivity* of reduction: if L_1 reduces to L_2 and L_2 reduces to L_3, then L_1 also reduces to L_3. With this reduction concept, we first define the class of the $\mathcal{NP}$-hard problems. A problem L is said to be $\mathcal{NP}$-hard, if all problems in $\mathcal{NP}$ are polynomial time reducible to L. Then, $\mathcal{NP}$-complete is a subset of $\mathcal{NP}$ problems that is also $\mathcal{NP}$-hard (the problems in the intersection of the classes $\mathcal{NP}$ and $\mathcal{NP}$-hard). Further, by the transitivity of reduction, it is not hard to see if L is in $\mathcal{NP}$ and there exists an $\mathcal{NP}$-complete problem L' which in polynomial time reduces to L, L is also $\mathcal{NP}$-complete. Therefore, in order to prove a problem is $\mathcal{NP}$-complete, we can show a reduction from another known $\mathcal{NP}$-complete problem. Consequently, there must be a prototypical $\mathcal{NP}$-complete problem. The problem is called the *satisfiability* (SAT) problem. SAT asks for the existence of a truth assignment to variables in a conjunctive normal form formula. Cook [44] first proved the $\mathcal{NP}$-completeness of this problem. Afterwards, many other problems have been proven to be $\mathcal{NP}$-complete via reductions from SAT. Two of the problems are independent set (IS) and unweighted vertex cover (UVC). IS and UVC are also polynomial time reducible to each other. Since UVC is a special case of VCD, together with the certifier shown earlier, we can conclude that VCD is $\mathcal{NP}$-complete. Finally, we can directly show a related optimization problem of a decision problem is $\mathcal{NP}$-hard. As described in Section 2.1.1 for vertex cover problems, VCD can be solved by solving VCO. It implies that VCD reduces to VCO. Also, we just stated that VCD is $\mathcal{NP}$-complete, hence VCO is $\mathcal{NP}$-hard.

2.2 LP and ILP

In this section, we shall look at the tools of linear programming and integer linear programming[4] for modeling and solving optimization problems. Recall that an optimization problem usually consists of a set of input instances, a set of constraints, an objective function and a goal (minimization or maximization). Assuming there is a minimization problem. It can be conveniently modeled by a general *linear program* (LP) below.

$$\begin{array}{lll} \text{minimize} & \sum_{j=1}^{n} c_j x_j & (2.11) \\ \text{subject to} & \forall i = 1, \ldots, m : \sum_{j=1}^{n} a_{ij} x_j \geq b_i & (2.12) \\ & \forall j = 1, \ldots, n : x_j \geq 0 & (2.13) \end{array} \tag{2.1}$$

The LP has an equivalent matrix form:

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{array} \tag{2.2}$$

where $\boldsymbol{c}$[5], $\boldsymbol{b}$ are column vectors of real numbers with dimensions n and m respectively and $\boldsymbol{A}$ is an $m \times n$ matrix. These inputs denote different input instances of the problem. The nonnegative column n-vector $\boldsymbol{x}$ denotes the solution/variable we wish to find subject to two *linear inequality constraints*. The constraint $\boldsymbol{x} \geq \boldsymbol{0}$ is call the *nonnegative constraint* on variables. $\boldsymbol{x}$ is a *feasible solution* if it satisfies all of the constraints and an *infeasible solution* if it violates at least one constraint. The *linear objective function* is represented by $\boldsymbol{c}^T\boldsymbol{x}$, which attains an *objective value* for every fixed $\boldsymbol{x}$. A feasible $\boldsymbol{x}$ with the minimum objective value (among all feasible solutions) is an *optimal solution*. The corresponding $\boldsymbol{c}^T\boldsymbol{x}$ is the *optimal objective value*. Notice that the objective function, objective value and optimal objective value are often termed as the *cost function*, *cost* and *optimal cost* respectively. An LP without any feasible solutions is called an *infeasible LP*, otherwise it is a *feasible LP*. For instance, an LP could have conflicting constraints to make it infeasible. An LP may also be *unbounded*, if its optimal solution does not have a finite objective value.

The above general LP is actually in a *standard form*. It can sufficiently capture other variations of the LP such as the objective function is maximization

[4]Notice that linear optimization tools [124] are particularly useful for solving the combinatorial optimization problems studied in this thesis. Other optimization tools may involve convex optimization [25] and non-linear optimization [23].

[5]$\boldsymbol{c}^T$ is the transpose of $\boldsymbol{c}$ which is a row vector.

instead of minimization, allowing negative variables, having inequality constraints with '$\leq$' instead of '$\geq$', and having equality constraints. For instance, in LP (2.1), maximizing $\sum_{j=1}^{n} c_j x_j$ is equivalent to minimizing $-\sum_{j=1}^{n} c_j x_j$; a negative variable x_j can be replaced by $x'_j - x''_j$ in the objective function and constraints where x'_j and x''_j are both non-negative variables; $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ can be substituted by $-\sum_{j=1}^{n} a_{ij} x_j \geq -b_i$; $\sum_{j=1}^{n} a_{ij} x_j = b_i$ can be expressed by $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ and $-\sum_{j=1}^{n} a_{ij} x_j \geq -b_i$. We can also convert the LP to a useful *slack form* such that all constraints are equalities except the non-negative constraints that are inequalities. For example, the constraint $\forall i : \sum_{j=1}^{n} a_{ij} x_j \geq b_i$ can be rewritten as a constraint $\forall i : \sum_{j=1}^{n} a_{ij} x_j - b_i = d_i$ and a nonnegative constraint $\forall i : d_i \geq 0$ on the added *slack variable* d_i.

The famous *simplex method* is one of the most popular and practical ways for solving LPs. This method essentially works on the slack (matrix) form of an LP via elementary row operations. Geometrically, the method first constructs a *basic solution* at a vertex of the feasible convex polytope of an LP and then moves to the other vertices/basic solutions with the increasing objective values. The process halts at one of the three conditions: a *basic optimal solution* is reached, the LP is unbounded, and the LP is infeasible. Although simplex behaves extremely well in practice, it suffers an exponential worst case runtime. The first polynomial time approach is the *ellipsoid method*. It was first introduced for solving convex optimization problems and then turned out to be able to solve LPs in polynomial time. On the negative side, the method has worse empirical performances than simplex for solving most of the LPs. Another group of algorithms following the work of [79] are based on the *interior point method*. Instead of searching around the vertices as simplex does, this method looks into the interior of the feasible polytope through projection. It has a better worst case polynomial runtime than ellipsoid and a similar practical efficiency as compared to simplex. Note that both ellipsoid and interior point methods run in weakly polynomial time. For recent developments of the LP algorithms based on the above methods, we recommend the books [24, 133].

One useful feature of LP is the concept of *duality*. Under this concept, we call LP (2.1) a *primal* LP. We observe that in this LP, the objective function (2.11) can be lower bounded by adding up the scaled constraints of (2.12). Let the scalers/multipliers be $\forall i : y_i \geq 0$, constraint (2.12) then becomes $\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_j y_i \geq \sum_{i=1}^{m} b_i y_i$. For a lower bound of $\sum_{i=1}^{m} b_i y_i$, we want to ensure $\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_j y_i$. For this guarantee, we need $\forall j : c_j \geq$

$\sum_{i=1}^{m} a_{ij}y_i$; and for the best lower bound, $\sum_{i=1}^{m} b_i y_i$ has to be maximized. These observations together gives the *dual* of the primal LP:

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{m} b_i y_i \\ \text{subject to} & \forall j = 1, \ldots, n : \sum_{i=1}^{m} a_{ij} y_i \leq c_j \\ & \forall i = 1, \ldots, m : y_i \geq 0. \end{array} \tag{2.3}$$

It is not hard to see that the dual of the dual LP (2.3) is the primal LP (2.1). There are other very useful properties established for primal and dual LPs. The first is the *weak duality* theory:

Theorem 2.2.1. *For every feasible primal solution $\boldsymbol{x}$ to LP* (2.1) *and dual solution $\boldsymbol{y}$ to LP* (2.3)*, we have $\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i$.*

In above, when deriving the dual LP, we have actually described a simple proof for the above theorem. The theorem implies an important fact that $\boldsymbol{b}^T\boldsymbol{y} \leq \min_{\boldsymbol{x}} \boldsymbol{c}^T\boldsymbol{x}$, i.e. any dual objective value low bounds the optimal primal objective value. The other interesting fact is that for a pair of feasible primal and dual LPs, their optimal objective values are the same. This is the *strong duality* theory:

Theorem 2.2.2. *For a pair of feasible primal and dual LPs* (2.1) *and* (2.3) *with optimal solutions $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$ respectively, we have $\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*$.*

By extending the weak and strong duality theories, we have the following *complementary slackness conditions* (CSCs):

Theorem 2.2.3. *Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be the feasible solutions to LPs* (2.1) *and* (2.3) *respectively. Then, $\boldsymbol{x}$ and $\boldsymbol{y}$ are both optimal if and only if both of the following primal and dual CSCs are satisfied:*

Primal CSCs: $\forall j = 1, \ldots, n : x_j \left(c_j - \sum_{i=1}^{m} a_{ij} y_i\right) = 0.$

Dual CSCs: $\forall i = 1, \ldots, m : y_i \left(\sum_{j=1}^{n} a_{ij} x_j - b_i\right) = 0.$

The CSCs are particularly useful in designing exact algorithms for solving LPs. As shown later in the thesis, they also play an important role in the design of approximation algorithms. So far, the established properties are for feasible LPs. The following theorem summarizes the feasibility relationship between primal and dual LPs.

Theorem 2.2.4. *For a pair of primal and dual LPs, one of the following must hold:*

(a) Primal LP is unbounded and dual LP is infeasible.

(b) Dual LP is unbounded and primal LP is infeasible.

(c) Both primal and dual LPs are feasible.

(d) Both primal and dual LPs are infeasible.

Proof. In this theorem, (a) and (b) imply that if at least one LP is unbounded, the other LP in the pair must be infeasible. To illustrate (a), if $\sum_{j=1}^{n} c_j x_j \to -\infty$ in LP (2.1), by weak duality, $\sum_{i=1}^{m} b_i y_i \leq \sum_{j=1}^{n} c_j x_j$ which is impossible. So the corresponding dual LP cannot have a feasible solution. Similarly, (b) holds as well. Obviously, (c) and (d) the other two complementary cases. □

If the nonnegative constraints on $\boldsymbol{x}$ is restricted to nonnegative or binary integers such that $\forall j : x_j \in \mathbb{Z}^+$ or $x_j \in \{0, 1\}$ respectively, LP (2.1) then becomes an *integer linear program* (ILP). Unlike LP, an ILP's feasible solution set is non-convex. Recall that the weighted vertex cover optimization (VCO) problem asks for a vertex subset $V' \subseteq V$ in graph $G = (V, E)$ such that every edge $e \in E$ has at least one endpoint in V' and the total weight is minimized. This problem can be modeled by the following ILP:

$$\begin{array}{lll} \text{minimize} & \sum_{i \in V} w_i x_i & (2.41) \\ \text{subject to} & \forall e = (i, j) \in E : x_i + x_j \geq 1 & (2.42) \\ & \forall i \in V : x_i \in \{0, 1\} & (2.43) \end{array} \tag{2.4}$$

in which solution x_i takes either 1 or 0 corresponding to vertex i is selected or not respectively; constraint (2.42) ensures a valid cover and (2.41) represents the total weight function to minimize. If we relax constraint (2.43) of the ILP to $\forall i \in V : x_i \geq 0$, we get its *LP-relaxation.*

For a given instance $\langle G = (V, E), \boldsymbol{w} \rangle$ of the LP-relaxation, the corresponding optimal objective value yields an lower bound of the optimal objective value of the ILP. The maximum difference (supremum) of these two optimal values over all instances is called the *integrity gap* of the ILP. Different approaches exist to exactly solve an ILP. The naive one is to enumerate all of the solutions. In the case of the VCO problem, we can try all 2^V possibilities of choosing vertices. Other more advanced methods are *cutting-plane* and *branch and bound*. Cutting-plane iteratively refines the feasible region of an ILP by imposing linear inequalities. Geometrically, these inequalities are cuts to ensure the solution to be integral. Branch and bound first solves the LP-relaxation of an ILP to get a solution value for each variable. Some of the variables might not have integer solution values, so the method recursively branch on these variables one by one by creating new LPs with nonnegative integer constraints. The bounding happens when the solution on a branch is

inferior, infeasible or integer. The whole method terminates when no branching is needed. Cutting-plane was shown to be slow in practice if used alone. However, it can be combined with branch and bound to run faster. There are also some special ILPs that are polynomially solvable. Most of such ILPs model the network flow/routing problems [9]. For a survey of the methods for solving ILP, we suggest the book [118]. Recall that for a general ILP, it is $\mathcal{NP}$-hard to solve. In addition, the above methods for ILP require worse case exponential runtime. For polynomial runtime, we need to be satisfied with an approximate solution. This brings up the field of approximation algorithms. We will see later the theories established for LP and ILP are very useful in the design of approximation algorithm.

2.3 Approximation algorithms

In this section, we introduce the theme of the thesis: approximation algorithms for optimization problems. Previously, we have seen complexity classes $\mathcal{P}$ and $\mathcal{NP}$ for decision problems. Accordingly, there are $\mathcal{P}$-optimization ($\mathcal{PO}$) and $\mathcal{NP}$-optimization ($\mathcal{NPO}$) problems. For instance, the polynomial time solvable shortest path problem belongs to the class $\mathcal{PO}$ and vertex cover belongs to $\mathcal{NPO}$. We have also seen how optimization problems are modeled by LPs and ILPs and exactly solved by different methods. For the $\mathcal{NPO}$ problems, these methods usually require exponential time and therefore they have not actually tackled the problems' inherent hardness. On the other hand, an approximation algorithm runs in polynomial time and as a tradeoff it provides a near-optimal solution quality. The approach of approximation also naturally explains and classifies the problems in $\mathcal{NPO}$. The solution quality of an approximation algorithm for a problem is measured by an *approximation ratio/factor/bound*. For each input instance I of a problem P and an algorithm A for P, let $A(I)$ and $OPT(I)$[6] denote a feasible solution (objective) value produced from A and the optimal solution value respectively. We say A is ρ-approximation for P if, for every $I \in P$, $A(I) \leq \rho OPT(I)$ and $\rho \geq 1$ (for a minimization problem P) or $A(I) \geq \rho OPT(I)$ and $0 < \rho \leq 1$ (for a maximization problem P). Equivalently speaking, A is an approximation algorithm for P with an approximation ratio of ρ. We remark that ρ is the worst case ratio of the algorithm. It is also called the performance/approximation guarantee. If ρ is a constant, then A is a constant factor algorithm, otherwise the value of ρ

[6] In the thesis, these are often denoted as A and OPT for brevity.

may depend on other parameters such as the input size. With the notion of approximation ratio, we are ready to classify problems with different difficulties/approximability in $\mathcal{NPO}$. Let n be the size of the input to a problem P, P is in the class *Approximable* ($\mathcal{APX}$) if there exists a constant factor approximation algorithm for P that runs in time polynomial in n. The vertex cover problem belongs to this class as well as some hard problems' metric cases. We say an algorithm is a *polynomial time approximation scheme* ($\mathcal{PTAS}$) if for any fixed $\epsilon > 0$ it achieves an approximation ratio of $(1+\epsilon)$ (for minimization problem) or $(1-\epsilon)$ (for maximization problem) in time polynomial in n. Then P is in the class $\mathcal{PTAS}$ if such an algorithm exists for P. A $\mathcal{PTAS}$ is a *full polynomial time approximation scheme* ($\mathcal{FPTAS}$) if the algorithm's runtime is polynomial in n and $\frac{1}{\epsilon}$ instead, so the $\mathcal{PTAS}$ runtime like $O\left(n^{\frac{1}{\epsilon}}\right)$ does not belong to $\mathcal{FPTAS}$. For these complexity classes, the relationship $\mathcal{FPTAS} \subseteq \mathcal{PTAS} \subseteq \mathcal{APX} \subseteq \mathcal{NPO}$ clearly holds. In addition, different cases of a problem may reside in different classes. Take the famous traveling salesman problem (TSP) as an example. On the positive side, its euclidean case admits $\mathcal{PTAS}$ and metric case belongs to $\mathcal{APX}$. On the negative side, one can not hope for any non-trivial approximation for its general case. Similar to the polynomial time reduction for decision problems, we have the concept of *approximation-preserving reduction* (AP-reduction) for optimization problems in $\mathcal{NPO}$. An AP-reduction[7] is defined such that if a problem P_1 AP-reduces to a problem P_2 and P_2 is approximable up to a factor of ρ, then P_1 is approximable up to a factor of $O(\rho)$. So far, we have looked at the approximability results of $\mathcal{NPO}$, there are also interesting inapproximability results based on the probabilistically checkable proof (PCP) theorem [12]. For example, a class called MAX-$\mathcal{SNP}$ [120] was defined for determining which problems have a $\mathcal{PTAS}$ and which do not. Formally, for any MAX-$\mathcal{SNP}$-hard problem, there does not exist a $\mathcal{PTAS}$ unless $\mathcal{P} = \mathcal{NP}$. Moreover, it is unlikely to obtain any nontrivial approximation guarantee for some very difficult problems such as maximum clique and maximum independent set. For more details on the computational complexity of optimization problems (from the perspective of approximation), we refer readers to the pointers [59, 15, 11].

After getting some knowledge about an optimization problem's hardness, we are ready to embark the process of designing an approximation algorithm

[7]Note that there are several competing definitions for the AP-reduction. The other important one is: if a problem P_1 AP-reduces to a problem P_2 and P_2 is approximable up to a factor of $1+\rho$, then P_1 is approximable up to a factor of $1+O(\rho)$.

for the problem. At a high level, the design of such algorithm is not much different from the design of exact algorithms for the problems in $\mathcal{PO}$. However, the (combinatorial) structure of an $\mathcal{NPO}$ problem is usually more elaborate. The algorithmic techniques and the analysis needed for unraveling and exploiting the structure is more sophisticated. To obtain an approximation ratio for a problem, one way is to find an AP-reduction from another problem with a known approximation factor. In fact, such kind of reduction is not always obvious. The more common approach is to compare the cost of the solution produced by an algorithm to the cost of the optimal solution OPT [138]. However, computing OPT is generally $\mathcal{NP}$-hard. To get around this issue, we can either work with OPT but do not compute it, or seek for polynomial time computable lower/upper bounds of OPT. These different approximation ideas lead to the following design methods of approximation algorithms. Broadly speaking, they consist of: combinatorial methods such as *greedy* and *local search* algorithms; linear programming based methods such as *primal-dual* and *LP-rounding*; randomized methods (as opposed to deterministic methods) to improve the (randomized) approximation factor and runtime such as *randomized local search* and *randomized LP-rounding*; hybrid methods formed by combining some of the previous methods and other techniques[8]. We recommend the books [141, 138, 82, 70] for a comprehensive survey on approximation algorithms for different optimization problems.

The thesis focuses on developing deterministic approximation algorithms for the resource allocation problems. As a warmup of the next chapters, the rest of this section gives readers some flavor of the key methods we adopt. For each method, we provide a brief discussion first and then apply it to the weighted vertex cover (VC) problem[9]. Notice that for the unweighted case of VC, similar results as the following presented can be obtained with simpler approximation algorithms.

2.3.1 Greedy and local search algorithms

As a combinatorial approach, a greedy algorithm constructs a solution step by step. At each step, the algorithm usually forms part of the solution that is locally optimal according to an objective function. The algorithm terminates

[8]There may also be approximation algorithms combined with some design techniques for the exact algorithms. Such techniques are advanced data structures, recursion, dynamic programing, network flows, fixed parameter tractable, etc.

[9]From now on, unless elsewhere specified, the problems we are going to deal with are optimization problems. Hence, here we use the abbreviation 'VC' instead of 'VCO'.

once a complete solution is formed. Local search is another combinatorial approach. It starts with a feasible (complete) solution and then iteratively moves to another feasible solution (a neighbor) with an improved objective value. The move is some small local changes and it stops at a local optimum when no changes can be made to further improve the objective value. Local search is often regarded as a greedy approach as its process is also guided by an objective. To strictly separate the two, one can consider that a greedy algorithm is building a solution while local search is improving on an existing solution. Further, greedy usually has a much shorter runtime than local search. Both of these approaches actually belong to a broader class — *heuristics*. For this reason, a local search method is sometimes called a 'greedy heuristic' or 'greedy augmentation' rather than 'greedy algorithm'. A heuristic is often problem-specific, and based on relatively simple and intuitive ideas. Heuristics are often developed with the hope of a good solution when a problem is too complex to solve by the standard optimization techniques within a reasonable timeframe. Other general heuristic methods such as simulated annealing and genetic algorithms are essentially variants of randomized local search.

In the thesis, along the combinatorial approach we consider greedy approximation algorithms. For more details about local search, readers can refer to Chapter 3. The greedy approximation algorithms are not much different from the greedy exact algorithms, except that in the analysis they ask for approximation ratios instead of optimality proofs. Compared with the other approximation algorithms, greedy methods are usually easy to design and implement, but often do not work towards to a meaningful approximation factor. In the following, we shall look at three greedy algorithms for the VC problem. The first two are based on naive greedy ideas which produce unbounded ratios. Recall that in an undirected graph $G = (V, E)$ with vertex weights $i \in V : w_i \geq 0$, VC asks for a vertex subset $V' \subseteq V$ such that every edge $e \in E$ has at least one endpoint in V' and $\sum_{i \in V'} w_i$ is minimized. The first greedy algorithm A_1 works as follows: in each iteration, the algorithm picks the vertex currently with the smallest weight and then removes edges incident to this vertex. This iterative process terminates once all edges are removed. The second algorithm A_2 works differently by iteratively choosing the vertex that covers the most number of uncovered edges first until all edges have been covered. To see their ratios are unbounded, we illustrate special problem instances as examples. These instances provide useful lower bounds of the algorithms' approximation factors. Let the graph $G = (V, E)$ be a star as shown in Figure 2.1. A star is a tree with one internal node (like node v in

the figure) and a number of leaves. If there is a star instance with $w_v = 1 + \epsilon$ in which ϵ is a small positive constant, and $\forall i \in V \backslash v : \ w_i = 1$, algorithm A_1 will pick all the vertices in $V \backslash v$, incurring a cost of $|V| - 1$. On the other hand, the optimal solution is obviously only selecting v with cost $1 + \epsilon$, so we can conclude that the ratio ρ_{A_1} of algorithm A_1 satisfies $\rho_{A_1} \geq \frac{|V|-1}{1+\epsilon}$ which is unbounded (in polynomial scale) as the size of V grows. Similarly, for algorithm A_2, we consider another star instance with $w_v = \omega$ where ω is a large positive constant, and $\forall i \in V \backslash v : \ w_i = 1$. In this case, A_2 instead chooses v to cover all edges first, but the optimal solution is to choose all the leaves. As a consequence, the ratio holds that $\rho_{A_2} \geq \frac{\omega}{|V|-1}$ which is also unbounded as ω can be much larger than $|V|$. The greedy objective of A_1 is to select the cheapest node first while the objective of A_2 is to cover the most number of edges first. From the naive algorithms A_1 and A_2, one may think of a new objective that always chooses the vertex with the smallest average weight, i.e. the smallest ratio of its weight to the number of the uncovered edges it covers. This is the heuristic of the third greedy algorithm A_3. The algorithm can be implemented in quadratic time $O\left(|V|^2\right)$ by maintaining linked edge counts for every vertex. The approximation factor is analyzed to be $O\left(\log |V|\right)$ through analyzing the structure of the optimal cover in every iteration. In fact, the VC problem is a special case of the fundamental weighted set cover problem and the algorithm A_3 is based on the original work [43] for this problem. We will revisit the set cover problem several times in the thesis. In Chapter 3 and 4, we shall see the applications of greedy algorithms.

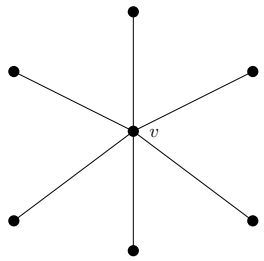

Figure 2.1: A star graph

2.3.2 LP rounding algorithms

We have looked at the basics of LP and ILP. Next, we introduce the linear programming based methods LP-rounding and primal-dual, which are important in the design of approximation algorithms. These methods utilize generic LP bounds to establish approximation ratios. In this section, we concentrate

on LP-rounding. Given an ILP of a minimization problem[10], it is relaxed to an LP (LP-relaxation) and solved by an existing tool (preferably a polynomial time algorithm) to obtain an optimal fractional solution. The fractional solution is then rounded to an integer solution via a *rounding process*. To obtain an approximation ratio, the cost of the rounded solution is compared to the cost of the optimal fractional solution which is a lower bound of the cost of the optimal integer solution. The approximation ratio is always lower bounded by the integrity gap of the ILP. A rounding process usually consists of several stages. Each stage transforms a fractional solution to another (partial) fractional solution. Eventually, a complete integer solution is reached. For an ILP with binary variables, the easiest way is to directly round up the variables with large fractional values to 1, while rounding down the other variables with relatively small values to 0. In this way, the sets of variables to round up and down have to be chosen carefully for the feasibility of the integer solution. In the following, we describe a simple LP-roundng algorithm for the VC problem. The algorithm achieves a constant approximation ratio of 2, which greatly outperforms the previously seen greedy algorithms with non-constant factors.

Recall that the VC problem is modeled by the following ILP:

$$\begin{array}{lll} \text{minimize} & \sum_{i \in V} w_i x_i & (2.51) \\ \text{subject to} & \forall e = (i,\, j) \in E :\, x_i + x_j \geq 1 & (2.52) \\ & \forall i \in V :\, x_i \in \{0, 1\} & (2.53) \end{array} \tag{2.5}$$

in which solution x_i takes either 1 or 0 corresponding to vertex i is selected or not respectively; constraint (2.52) ensures at least one end of every edge is selected, so all edges are covered; (2.51) represents the objective function to minimize. If we relax the nonnegative constraint (2.53), we get its LP-relaxation below.

$$\begin{array}{lll} \text{minimize} & \sum_{i \in V} w_i x_i & (2.61) \\ \text{subject to} & \forall e = (i,\, j) \in E :\, x_i + x_j \geq 1 & (2.62) \\ & \forall i \in V :\, x_i \geq 0 & (2.63) \end{array} \tag{2.6}$$

Let $\boldsymbol{x}^*$ denote the optimal fractional solution found after solving this LP. In the rounding step, we round the variables with optimal values no less than $\frac{1}{2}$ to 1 while rounding others to 0. In other words, we eventually select the vertex set $V' = \left\{ i \in V \,|\, x_i^* \geq \frac{1}{2} \right\}$. In the analysis, firstly we show the feasibility of V', i.e. V' is a valid cover. From constraint (2.62), it can be deduced that

[10]For a maximization problem, the basic LP-rounding steps are identical.

$\forall e = (i, j) \in E$ at least one of x_i^* and x_j^* is no less than $\frac{1}{2}$. Hence, V' contains vertices covering all edges. Secondly, let SOL and OPT_{LP}[11] denote the objective values $\sum_{i \in V'} w_i$ and $\sum_{i \in V} w_i x_i^*$ respectively, we are going to find a factor ρ such that $SOL \leq \rho OPT_{LP}$. This factor is the approximation ratio because OPT_{LP} lower bounds the optimal objective value of the ILP. From the definition of V', we get $\sum_{i \in V'} w_i x_i^* \geq \frac{1}{2} \sum_{i \in V'} w_i = \frac{1}{2} SOL$. Also, it is obvious that $\sum_{i \in V'} w_i x_i^* \leq \sum_{i \in V} w_i x_i^* = OPT_{LP}$. Therefore, $SOL \leq 2OPT_{LP}$ and the approximation ratio ρ is 2.

The above rounding idea is based on the work in [68]. Some advanced rounding algorithms are actually much more complicated. They usually involve a mixture of ideas like constructing structured intermediate solutions, maintaining complementary slackness conditions, and adding randomness to the rounding process. In Chapter 3 and 6, we will see some of these techniques applying to the facility location and resource allocation problems.

2.3.3 Primal-dual algorithms

Primal-dual algorithms were traditionally used for exactly solving the $\mathcal{P}$ problems with integer optimal solutions to their LP formulations. Such problems include matching, network flow, and shortest paths. The algorithms are usually driven by the powerful complementary slackness conditions (CSCs) as the conditions tell that the optimal solution to an LP satisfy all its CSCs. Hence, for the optimality of a solution, one can either maintain primal feasibility and CSCs and seek dual feasibility or maintain dual feasibility and CSCs and seek primal feasibility.

In the case of the $\mathcal{NP}$-hard problems, a primal-dual approximation algorithm/schema is somewhat more subtle and sophisticated than LP-rounding. It strongly relies on the LP duality theories we have seen before to establish approximation guarantees. Given an ILP of a minimization problem, we can usually immediately obtain its LP-relaxation (primal LP) and a corresponding dual LP. From the weak duality theory, the cost of any feasible dual solution gives a lower bound of the cost of any primal solution (including the optimal solution). Therefore, for an approximation ratio, we can compare the cost of the obtained integer solution (from an algorithm) with the cost of any feasible dual solution instead, avoiding solving the primal LP. This is perhaps the biggest advantage of primal-dual over LP-rounding as solving a large LP incurs

[11] It might be better to denote them as $SOL(I)$ and $OPT_{LP}(I)$ for any input instance I. For briefness, in the thesis we mostly use the short notation.

a high running time and often leads to an unpractical algorithm. Similar to an exact primal-dual algorithm, the mechanism of a primal-dual approximation algorithm is also driven by CSCs for generating a near optimal integer solution. Notice that the primal CSCs (respectively the dual CSCs) are satisfied is equivalent to the constraints of dual LP (respectively the primal LP) are tight[12].

A primal-dual algorithm often starts with primal and dual solutions that are both zero, so dual solution is feasible but primal infeasible. Afterwards, it maintains a feasible dual solution, and iteratively constructs a feasible integer primal solution inferred by the dual solution. More precisely, in each iteration, the values of a set of dual variables are increased until a dual constraint becomes tight. This dual constraint corresponds to a primal variable, so we add this variable to the primal solution (e.g. set this variable to 1 or increase it by an integral amount). The algorithm terminates with the constructed primal and dual solutions that are both feasible and related to each other. Consequently, we may establish a bound between the cost of the primal solution and the cost of the dual. This bound is the approximation ratio by weak duality. The other interpretation of this primal-dual process is that the primal cost is gradually paid by the dual cost and one can find some intuitive meanings (such as the payment, value, price, and contribution) for the dual variables according to the dual constraints. Sometimes, dual may make primal increase too much or dual may increase too much itself (even become infeasible). In this case, the algorithm usually incorporates an extra key step to find a minimal primal or dual solution. This step is also guided by CSCs or equivalently the constraints of the primal and dual LPs. There are excellent surveys [142, 139] on the approximation algorithms using primal-dual schema. In the following, we show a primal-dual algorithm for the VC problem based on the work of [19].

Similar to the LP-rounding algorithm for VC, the primal-dual algorithm also achieves an approximation ratio of 2. However, it runs in linear time which is much more practical. In Chapter 3, 4, 5, and 6, we will see extensive use of the versatile primal-dual method for the facility location and resource allocation problems in achieving both effective and efficient solutions.

The VC problem is modeled by ILP (2.5). Its primal LP is LP (2.6). The dual LP below can be easily obtained from the primal:

[12]A constraint is tight if it holds with equality.

$$\begin{array}{lll} \text{maximize} & \sum_{e\in E} y_e & (2.71) \\ \text{subject to} & \forall i \in V : \sum_{e\in\delta(i)} y_e \leq w_i & (2.72) \\ & \forall e \in E : y_e \geq 0 & (2.73) \end{array} \tag{2.7}$$

in which y_e is the dual variable associated with each edge e, (2.72) is the dual constraint on y_e, (2.71) is the cost function of the dual and $\delta(i)$ denotes the set of edges incident on vertex i. Following from our described schema, we can design a primal-dual approximation algorithm (Algorithm 2.1). In the algorithm, each vertex $i \in V$ is paid to be chosen by the edges in $\delta(i)$. The payment of edge e is y_e, and i is selected if all edges in $\delta(i)$ fully pay its weight w_i (the dual constraint is tight). Once i is picked, edges in $\delta(i)$ will be removed from E since they are already covered. At this moment, the corresponding y_e's will not be raised anymore to violate the existing tight dual constraints. The algorithm terminates and outputs the solution $\boldsymbol{x}$ once all edges are covered. Note that the set V' corresponding to $\boldsymbol{x}$ is only needed in the analysis.

Algorithm 2.1 A Primal-Dual Algorithm for VC

Input: undirected graph $G = (V, E)$ and vertex weights $\boldsymbol{w}$.
Output: the binary integer solution $\boldsymbol{x}$.
Initialization: set primal variable $\boldsymbol{x} \leftarrow \boldsymbol{0}$, dual variable $\boldsymbol{y} \leftarrow \boldsymbol{0}$ and the vertex set to pick $V' \leftarrow \emptyset$.

While $E \neq \emptyset$:

1. pick an edge $e \in E$ arbitrarily and increase value of y_e until some dual constraint goes tight (possibly one or two constraints), i.e. $\exists i \in V : \sum_{e\in\delta(i)} y_e = w_i$.
2. Let S be the set of vertices (possibly one or two vertices) corresponding to the tight constraints, $\forall i \in S$: set $x_i \leftarrow 1$, $E \leftarrow E \backslash \delta(i)$ and include i into V'.

A primal-dual approximation algorithm is often similarly expressed as above where the change of dual is continuous (e.g., gradually increase a dual value until some dual constraints become tight). By describing in this way, the primal-dual process can be more easily visualized and its ratio analysis becomes more intuitive. Nevertheless, for the ease of implementation and runtime analysis, the above algorithm is discretized as follows (Algorithm 2.2).

Algorithm 2.2 Discretization of the Primal-Dual Algorithm

Input: $G = (V, E)$ and $\boldsymbol{w}$. **Output**: $\boldsymbol{x}$.
Initialization: $\boldsymbol{x} \leftarrow \mathbf{0}$, $\boldsymbol{y} \leftarrow \mathbf{0}$ and $V' \leftarrow \emptyset$.
for all $e = (i, j) \in E$ **do**
  let $p_i \leftarrow \sum_{e \in \delta(i)} y_e$ and $p_j \leftarrow \sum_{e \in \delta(j)} y_e$
  $\Delta \leftarrow \min(w_i - p_i, w_j - p_j)$
  $y_e \leftarrow y_e + \Delta$, $p_i \leftarrow p_i + \Delta$ and $p_j \leftarrow p_j + \Delta$
  if $p_i = w_i$ **then**
    $x_i \leftarrow 1$ and $V' \leftarrow V' \cup \{i\}$
  if $p_j = w_j$ **then**
    $x_j \leftarrow 1$ and $V' \leftarrow V' \cup \{j\}$

The above algorithm is combinatorial and essentially identical to Algorithm 2.1[13]. For discretization, the algorithm utilizes an increment variable Δ. For simplicity, it also maintains a variable $\boldsymbol{p}$ that records the total payments to the vertices. In terms of runtime analysis, it is easy to see the algorithm runs in linear time. For the approximation ratio analysis, we are first sure that the produced primal and dual solutions are feasible. Then, we compare the cost of the primal solution $\sum_{i \in V'} w_i$ to the cost of the dual $\sum_{e \in E} y_e$. For the vertices in V', the algorithm maintains the tight dual constraints, i.e., $\forall i \in V'$: $w_i = p_i$, so we have $\sum_{i \in V'} w_i = \sum_{i \in V'} p_i$. In addition, $\sum_{i \in V'} p_i \leq \sum_{i \in V} p_i = \sum_{i \in V} \sum_{e \in \delta(i)} y_e$, and it is clear that $\sum_{i \in V} \sum_{e \in \delta(i)} y_e = 2 \sum_{e \in E} y_e$. Therefore, we get $\sum_{i \in V'} w_i \leq 2 \sum_{e \in E} y_e$, and the algorithm attains an approximation factor of 2 by weak duality.

[13]Sometimes a primal-dual algorithm is essentially the same as a greedy algorithm. However, in terms of ratio analysis, the equivalent primal-dual algorithm might be more powerful.

Chapter 3

Discrete Facility Location

Given a finite set of facility and client locations, a (discrete) facility location problem concerns about selecting a subset of facility locations to build facilities to serve clients subject to an objective and several constraints. There are many different types of facility location problems. An important one is the minsum facility location, i.e., the objective is to minimize the sum of facility opening costs, and connection/service costs between clients and facilities. The constraints of a facility location problem are usually imposed on both facilities and clients, such as a maximum number of facilities to open (a global constraint), and a minimum number of facilities each client requires to use (a local constraint). Facility location problems have been extensively studied since 1960s. Early works are mostly from the field of operations research. They are due to Stollsteimer [130], Kuehn and Hamburger [86], Manne [110] and Balinski [17]. Early surveys are from Krarup and Pruzan [84] and Mirchandani and Francis [116].

Perhaps the most widely studied facility location problem is the *Uncapaciated Facility Location* (UFL) problem[1] introduced in Chapter 1. It was first intensively studied as an operations research problem in the literatures [130, 17, 51, 84, 46, 116]. These early works focused on exactly solving the problem. Some time later, after the the theories of $\mathcal{NP}$-completeness [44] and approximation algorithm [58, 78] have been established, UFL gradually became one of the central topics in theoretical computer science. Surprisingly, no constant factor approximation algorithm was known for this problem until 1997. Since then, there has been a lot of research on the approximation algorithms for UFL and its variants. Recent surveys that summarize the research works include [126, 127, 140, 141]. In addition to these surveys, this chapter

[1] It is also called the simple plant location problem in operations research.

briefly reviews the state of art techniques, results and applications of several important facility location problems with emphasis on the metric UFL. The content of this chapter also serves as the related work for the resource allocation problems studied in this thesis.

3.1 Uncapacitated Facility Location

Despite the fact that UFL is probably the simplest facility location problem, it is $\mathcal{NP}$-hard and provides a showcase of different approximation techniques. In the UFL problem[2], we are given a set $\mathcal{F}$ of n_f facility locations and a set $\mathcal{C}$ of n_c clients. For convenience, let the input sizes be $n = n_f + n_c$ and $m = n_f \times n_c$. The cost of opening a facility at location $i \in \mathcal{F}$ is f_i. For a client $j \in \mathcal{C}$, the cost of connecting a facility at i to j is c_{ij}. Every client is required to connect to at least one facility. The objective is to open a subset of facilities and assign clients to open facilities such that the total facility opening and connection cost is minimized. UFL has broad application domains, ranging from management science [45, 47, 80, 111], networks, services design [53, 121, 101, 55, 143, 146, 4] to computational biology [50] and computer vision [87] applications. As shown later, there are also more complicated UFL variants. They are designed for solving specific practical issues arising from different applications.

3.1.1 LP formulations

UFL has the following standard integer linear program (ILP) originally due to Balinski [17].

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij} x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \geq 1 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \in \{0,1\} \\ & \forall i \in \mathcal{F} : y_i \in \{0,1\} \end{array} \tag{3.1}$$

In the formulation, the first constraint ensures that each client is assigned to some facility, and the second constraint ensures that a client can only connect to open facilities. Notice that each facility can be assigned to an unlimited

[2]For convenience, we state the classical UFL definition here which is actually identical to the definition in Chapter 1.

number of clients, and only one assignment to a facility is actually required for each client[3] for the goal of cost minimization.

To explore the full potential of this (minsum) formulation, we observe that it is mathematically equivalent[4] to the following maximization variant. This variant was traditionally proposed for modeling the applications of supply chain management [46] and locating bank accounts [45].

$$\begin{array}{ll} \text{maximize} & \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} p_{ij}x_{ij} - \sum_{i\in\mathcal{F}} f_i y_i \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \geq 1 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \in \{0,1\} \\ & \forall i \in \mathcal{F} : y_i \in \{0,1\} \end{array} \tag{3.2}$$

In the supply chain context, f_i is the cost incurred to open a facility at location $i \in \mathcal{F}$ while p_{ij} is the known profit by satisfying a demand of client j from i. Typically, $p_{ij} = d_j\,(p_j - q_i - t_{ij})$ where d_j is the amount of the demand, p_j is the unit (demand) selling price to client j, q_i is the unit production cost at facility i, and t_{ij} is the unit transportation cost. The tricky bit here is even if we omit the price term p_j and set $p_{ij} = d_j\,(-q_i - t_{ij})$, the optimal solution of the ILP remains the same. This is because, when adding back the price term, one can verify that the objective is only changed by a constant $\sum_{j\in\mathcal{C}} d_j p_j$. As a consequence, this ILP's objective function consists of all negative terms to maximize. We can then safely set the nonnegative $c_{ij} = -p_{ij} = d_j\,(q_i + t_{ij})$[5] and change maximization to minimization to obtain the conventional ILP (3.1). Although these two models are equivalent from the perspective of optimization, they are different from the perspective of approximation[6] . The first approximation algorithm for the maximization variant was proposed by Cornuejols *et al.* [45]. The authors showed a simple greedy algorithm with a factor of 0.632 even for the non-metric p_{ij}'s. Later, this ratio was improved to 0.828 by Ageev [6] using the idea of randomized LP rounding. The standard UFL formulation is more versatile since the objective function of ILP (3.1) can be arbitrarily scaled without affecting the (approximation) solution quality. More precisely, the objective could become $\sum_{i\in\mathcal{F}} \alpha_i f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} \beta_{ij} c_{ij} x_{ij}$. This is equivalent to solve another UFL problem with an objective function

[3]This means the solution of the ILP stays the same even if the first constraint is changed to $\forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} = 1$.

[4]Two formulations have the same optimal objective value.

[5]This cost can simply be regarded as a negative profit, so the profit decreases as the cost increases.

[6]For approximation, the maximization variant looks for a lower bound of the optimal cost while the minimization problem looks for an upper bound.

$\sum_{i\in\mathcal{F}} f'_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c'_{ij}x_{ij}$ where $\forall i \in \mathcal{F}, j \in \mathcal{C}: f'_i = \alpha_i f_i$, $c'_{ij} = \beta_{ij}c_{ij}$. We call this the (α, β)-UFL problem which covers the case of UFL with arbitrary demands [77], i.e., the objective is $\sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} d_j c_{ij} x_{ij}$ where d_j is the amount of the demand of a client j. In the field of theoretical computer science (TCS), Shmoys [129] mentioned that the standard (minimization) ILP is much more natural for the setting of network design. This led to the later more intensive study of the problems built on the standard UFL. From now on in the thesis, we will mostly consider the minsum facility location and resource allocation problems with single demands, i.e., $\forall j \in \mathcal{C}: d_j = 1$.

3.1.2 Hardness results

Several hardness results for UFL appeared in [64, 140, 82]. Before looking at the approximation algorithms for UFL, in the following, we present some of the important hardness results for the problem. For the background knowledge of computational complexity, reader may refer back to Chapter 2.

Before discussing the hardness results, we first note the following definitions of the decision problems of set cover (SC) and UFL.

Definition 3.1.1. Given a set cover instance $(\mathcal{U}, \mathcal{S})$ and an integer parameter $k \leq |\mathcal{S}|$ where $\mathcal{U}$ is a universe of elements and $\mathcal{S}$ is a collection containing the subsets of $\mathcal{U}$. The union of these subsets is $\mathcal{U}$. The problem asks whether there is a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ such that the union of the subsets in $\mathcal{S}'$ is still $\mathcal{U}$ (a set cover) and $|\mathcal{S}'| \leq k$.

Definition 3.1.2. Given a UFL instance $(\mathcal{F}, \mathcal{C}, \boldsymbol{f}, \boldsymbol{c})$ and an integer parameter c^*. The problem asks whether there is a set of facilities $\mathcal{F}' \subseteq \mathcal{F}$ and an assignment of clients to facilities in $\mathcal{F}'$ such that the total cost is at most c^*.

The first definition above is to find the minimum cardinality set cover if SC is viewed as an optimization problem. It is equivalent to find the minimum weight set cover (to be discussed in Chapter 5) with unit weights of 1. This problem is strongly $\mathcal{NP}$-complete through a reduction from the independent set problem. In the second definition, $\boldsymbol{f}$, $\boldsymbol{c}$, and the parameter c^* can all be fractional. However, they can be easily transformed to integers through scaling, so here we treat them as integers.

Theorem 3.1.3. *[82] The metric UFL is strongly $\mathcal{NP}$-hard.*

The strong $\mathcal{NP}$-hardness in the above theorem implies that unless $\mathcal{P} = \mathcal{NP}$, there does not even exist an exact pseudo-polynomial time algorithm for

the metric UFL. This is a stronger statement than the $\mathcal{NP}$-hardness of a problem where an exact pseudo polynomial time algorithm might exist, e.g. the knapsack problem that is weakly $\mathcal{NP}$-hard. To prove the theorem, the idea is to reduce a SC decision problem to a metric UFL decision problem in polynomial time. The key of the reduction is to preserve the metricity of $\boldsymbol{c}$ when constructing an UFL instance $(\mathcal{F}, \mathcal{C}, \boldsymbol{f}, \boldsymbol{c})$ from a SC instance $(\mathcal{U}, \mathcal{S})$, and show that the resulting UFL instance has a cost $c^* = |\mathcal{C}| + k$ (an 'yes' instance of UFL) if and only if the SC instance has a set cover of size k (an 'yes' instance of SC).

Theorem 3.1.4. *[64] The metric UFL is MAX-$\mathcal{SNP}$-hard.*

Recall that for any MAX-$\mathcal{SNP}$-hard (optimization) problem, there does not exist a $\mathcal{PTAS}$ unless $\mathcal{P} = \mathcal{NP}$. It is known that in [120], the B-vertex cover problem has been shown to be MAX-$\mathcal{SNP}$-hard. B-vertex cover differs from vertex cover in which the degree of each node is bounded by a constant B. To prove the metric UFL is also MAX-$\mathcal{SNP}$-hard, we show an approximation-preserving reduction from B-vertex cover to metric UFL, that is showing a $(1+\rho)$ approximation algorithm for the metric UFL problem implies a $(1+O(\rho))$ approximation algorithm for the B-vertex cover problem. In the reduction, the construction of a UFL instance is similar to the construction for proving the previous theorem. More importantly, for the constructed UFL instance, one needs to assume the existence of a $(1+\rho)$ approximation algorithm, and show that the solution produced from this algorithm can be mapped to an approximate solution to the B-vertex cover with ratio $(1+\rho')$ where $\rho' = (1+B)\rho = O(\rho)$.

Theorem 3.1.5. *[82] The non-metric UFL cannot be approximated with a ratio smaller than $O(\log n)$ unless $\mathcal{NP} \subseteq DTIME[n^{O(\log\log n)}]$ or $\mathcal{P} = \mathcal{NP}$.*

Theorem 3.1.6. *[64, 140] The metric UFL cannot be approximated with a ratio smaller than 1.463 unless $\mathcal{NP} \subseteq DTIME[n^{O(\log\log n)}]$ or $\mathcal{P} = \mathcal{NP}$.*

The preceding two theorems are the inapproximatability results. The first is for the non-metric UFL. This result is due to the reason that SC can be viewed as a special case of the non-metric UFL. In addition, from [52] or [122], the best factor one can hope for SC is a logarithmic approximation factor under the complexity assumption $\mathcal{NP} \nsubseteq DTIME[n^{O(\log\log n)}]$ or $\mathcal{P} = \mathcal{NP}$ respectively. The former assumption essentially means not all problems in $\mathcal{NP}$ can be solved in superpolynomial (deterministic Turing machine) time.

Hochbaum [69] and Lin & Vitter [100] separately gave algorithms achieving this best possible $O(\log n)$ factor. The former author considered the UFL problem instance as an equivalent minimum weight set cover instance and then performed the greedy algorithm of [43]. The latter authors used a novel LP-rounding technique called filtering. This technique is general enough to be applied to many combinatorial optimization problems.

The second theorem states that a lower bound approximation ratio of 1.463 can be obtained for the metric UFL. The basic idea is to transform an instance of SC to an instance of UFL and utilize an extended hardness result for SC (Lemma 3.1 of [64]). More precisely, we can run an approximation algorithm for UFL repeatedly to construct partial solutions for the set cover instance. By analyzing each iteration, if the algorithm can ever finds a cover that violates the hardness result for SC, it implies the converse of the complexity assumption. Otherwise, a lower bound can be cleverly calculated to be 1.463. In fact, for a special case of the metric UFL where all connection costs are within the interval [1, 3], a simple greedy augmentation algorithm [64] achieved this lower bound.

Theorem 3.1.7. *[64] For the metric UFL, the integrity gap of the standard LP is at least 1.463.*

To analyze a lower bound for the integrity gap, one has to demonstrate at least a metric UFL instance such that the ratio between its optimal integer solution and optimal fractional solution is 1.463. In fact, Guha and Khuller [64] constructed a special family of UFL instances and analyzed that the gap for this family is at least 1.463. Therefore, the overall integrity gap of the LP is at least 1.463, implying none of the LP-rounding strategies can achieve a ratio smaller than 1.463.

For the complete metric UFL problem where $\mathcal{F} = \mathcal{C}$, a lower bound of the approximation guarantee was shown to be 1.278 (smaller than 1.463) through a reduction from the dominating set problem [64]. For a special case of UFL in which there is no opening cost, this problem is clearly polynomial time solvable. The hardness results presented above carry over to the problems of which UFL is a special case, including the resource allocation problems we have introduced before and some other facility location variants to be introduced later. Throughout the rest of this chapter, for completeness, we will also see a bi-factor hardness result for UFL and improved hardness results for some other facility location problems.

3.1.3 Approximation algorithms

In this section, we summarize approximation techniques and results for the metric UFL. The first constant-factor approximation algorithm for the metric UFL was proposed by Shmoys *et al.* [129] in 1997. Since then, there have been a lot of efforts (see Table 3.1[7]) to approach the approximation lower bound 1.463, and the best result is 1.488 obtained in 2011. This problem provides a showcase of different kinds of approximation algorithms. In the following, we separately review greedy and primal-dual, LP-rounding, and local search and hybrid algorithms[8] for the metric UFL. The techniques used in these algorithms and their analyses are quite technical and involved. For each type of the algorithms, we only highlight key techniques and results. Readers should look at the references in the table for more details. Notice that the basics of linear programming and approximation algorithms are described in the relevant sections of Chapter 2.

Reference	Ratio	Algorithm/ Time
Jain and Vazirani (1999) [75, 77]	3	primal-dual/ $O\left(n^2 \log n\right)$
Mettu and Plaxton (2000) [113, 114]	3	greedy/ $O\left(n^2\right)$
Mahdian *et al.* (2001) [104, 73]	1.861	greedy or primal-dual/ $O\left(n^2 \log n\right)$
Jain *et al.* (2002) [74, 73]	1.61	greedy or primal-dual/ $O\left(n^3\right)$
Shmoys *et al.* [129]	3.16	LP-rounding
Chudak and Shmoys (1998) [41, 42]	1.736	LP-rounding
Sviridenko (2002) [131]	1.582	LP-rounding
Byrka *et al.* (2010) [29]	1.575	LP-rounding
Korupolu *et al.* (1998) [83, 102]	$5+\epsilon$	Local search/ $O\left(n^4 \log \frac{n}{\epsilon}\right)$
Charikar and Guha (1999) [34, 35]	$3+\epsilon$	Local search/ $\tilde{O}\left(\frac{n^2}{\epsilon}\right)$
Arya *et al.* (2001) [13, 14]	$3+\epsilon$	Local search/ $O\left(n^4 \log \frac{n}{\epsilon}\right)$
Guha and Khuller (1998) [64, 65]	2.408	Hybrid
Mahdian *et al.* (2002) [106, 108]	1.52	Hybrid/ $O\left(n^3\right)$
Byrka (2007) [27, 28]	1.5	Hybrid
Li (2011) [89, 90]	1.488	Hybrid

Table 3.1: UFL approximation results

[7] A theoretical result often appears both in a conference paper and a journal article (in an extended form). In the above table, for the results having two related publications, we record the year in which the result first appeared. For the algorithms require solving LP, we do not list their running times. This is because the time spent by an LP solver usually dominates the overall runtime.

[8] A hybrid approximation algorithm consists of several standalone algorithms of different kinds. Some of the hybrid algorithms for UFL contain a local search (also called greedy augmentation) step. Therefore, we discuss these two types of algorithms together.

Greedy and primal-dual algorithms In the table, we can find that the greedy and primal-dual algorithms have clear runtime advantage over the other algorithm types such as LP-rounding and local search. Jain and Vazirani [75] first came up with a 3-approximation algorithm (known as the JV algorithm) that runs in time $O(n^2 \log n)$. The breakthrough of their work is the first-time adoption of the powerful *primal-dual schema*, which is based on the *complementary slackness conditions* (CSCs) of the following primal and dual LPs of UFL.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij} x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \geq 1 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij},\, y_i \geq 0 \end{array} \tag{3.3}$$

$$\begin{array}{ll} \text{maximize} & \sum_{j\in\mathcal{C}} \alpha_j \\ \text{subject to} & \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} \beta_{ij} \leq f_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j - \beta_{ij} \leq c_{ij} \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j,\, \beta_{ij} \geq 0 \end{array} \tag{3.4}$$

The primal and dual CSCs are shown below, assuming the primal solution $(\boldsymbol{x}, \boldsymbol{y})$ and dual solution $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ are both optimal.

Primal CSCs:
$\forall i \in \mathcal{F}, j \in \mathcal{C}$: If $x_{ij} > 0$ then $\alpha_j - \beta_{ij} = c_{ij}$.
$\forall i \in \mathcal{F}$: If $y_i > 0$ then $\sum_{j\in\mathcal{C}} \beta_{ij} = f_i$.
Dual CSCs:
$\forall j \in \mathcal{C}$: If $\alpha_j > 0$ then $\sum_{i\in\mathcal{F}} x_{ij} = 1$.
$\forall i \in \mathcal{F}, j \in \mathcal{C}$: If $\beta_{ij} > 0$ then $y_i - x_{ij} = 0$.

It is $\mathcal{NP}$-hard to find such optimal solutions. However, if there exists a pair of feasible solutions $(\boldsymbol{x}, \boldsymbol{y})$ and $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ that satisfy the following relaxed primal CSCs and the above dual CSCs, it is not hard to see the cost of $(\boldsymbol{x}, \boldsymbol{y})$ is 3-approximation. This is achieved by doing addition and substitution with these CSCs to bound the primal cost by the dual cost. The approximation ratio is then implied from the weak duality theory.

Relaxed primal CSCs:
(r) $\forall i \in \mathcal{F}, j \in \mathcal{C}$: If $x_{ij} = 1$ then $\frac{1}{3} c_{ij} \leq \alpha_j - \beta_{ij} \leq c_{ij}$,
$\forall i \in \mathcal{F}$: If $y_i = 1$ then $\sum_{j\in\mathcal{C}} \beta_{ij} = f_i$.

The primal-dual algorithm JV is driven by the above CSCs to produce such satisfying $(\boldsymbol{x}, \boldsymbol{y})$ and $(\boldsymbol{\alpha}, \boldsymbol{\beta})$. For the ease of description and understanding,

there are some intuitive meanings[9] for the dual variables and constraints: i) α_j is the total price paid by a client j to offset c_{ij} and possibly f_i (for any facility i) in order to connect to i; ii) β_{ij} is the net contribution of a client j to a facility i such that $\beta_{ij} = \alpha_j - c_{ij}$; iii) if $\sum_{j \in \mathcal{C}} \beta_{ij} = f_i$ holds, we say the facility i is fully paid to be temporarily opened by the contributions of the clients. With these notions, the algorithm then proceeds in two phases. The first phase is the usual primal-dual process that updates the primal and dual variables until the primal CSCs are satisfied. After this phase, the dual CSCs are not ensured and the total cost of the dual solution $\sum_{j \in \mathcal{C}} \alpha_j$ pays less than the cost of the primal. As a consequence, the total primal cost cannot be upper bounded by the dual cost for an approximation ratio. This motivates the second phase in which a minimal primal solution is constructed that satisfies both the dual CSCs and the relaxed primal CSCs. In fact, this phase ensures the key CSC (r) which eventually leads to the approximation ratio. This condition is achieved via building a *support graph* G and perform *clustering*[10] on G. The graph is defined as $G = (\mathcal{F} \cup \mathcal{C}, E)$ where $\forall i \in \mathcal{F}, j \in \mathcal{C}$: we include the edge (i, j) in E if $x_{ij} > 0$ after the first phase. The clustering process then iteratively partitions G into disjoint clusters each centered at a different facility. These facility centers are permanently opened to serve clients. Afterwards, each client reconnects to one of the closest open facility instead of the temporarily opened facilities it is connected with in the first phase. The reconnection is necessary to ensure the dual CSCs. It may also cause a client j's connection cost to be upper bounded by $3\alpha_j$ due to the triangle inequality in the metric condition. This connection cost bound eventually results in the condition (r).

Following Jain and Vazirani's work, Mettu and Plaxton [113] gave a faster greedy algorithm (known as the MP algorithm) with the same performance ratio. The greedy choices in their algorithm make the primal-dual process of [75] explicit with slight modifications. Specifically, the algorithm first creates a *ball* around each facility in $\mathcal{F}$. Each ball is centered at a facility with a defined radius that captures the contribution of a client for opening the facility. A ball with a smaller radius usually indicates the facility in the ball has a cheaper cost or there are more clients close to this facility. The algorithm then iteratively partitions/clusters the set $\mathcal{F} \cup \mathcal{C}$ into disjoint balls. It greedily chooses the ball with the smallest radius first until all balls have been considered. In the

[9] In Chapter 4, 5, and 6, dual variables in the primal-dual algorithms for the resource allocation problems also adapt to these meanings.

[10] As shown later, the ideas of support graph and clustering are also frequently used in LP-rounding algorithms.

process, it also ensures geometrically the next ball chosen does not intersect with any of the previously selected balls. At the end, facility centers of the picked balls are opened and each client can connect to its closest open facility. Since the publication of the MP algorithm, Thorup [134, 136] showed that the runtime bound $O(n^2)$ of the algorithm is optimal for any constant factor approximation. Built on MP, Thorup also considered UFL in a sparse graph with n nodes and $m = O(n)$ edges. He presented an algorithm with factor $3 + o(1)$ in $\tilde{O}(m)$[11] time for this new setting. Later, Archer *et al.* [10] showed that MP can be viewed as an equivalent primal-dual algorithm.

Inspired from the greedy algorithm for the minimum set cover problem [43], Mahdian *et al.* [104] created another breakthrough along the combinatorial approach. Their algorithm is not much different from the Hochbaum's algorithm [69] for the non-metric UFL. It only additionally sets the opening cost of any chosen facility to zero. Nevertheless, in the algorithm's analysis and implementation, they recast the algorithm as a primal-dual algorithm (known as the MMSV algorithm) similar to Jain and Vazirani's [75], and combined the *dual fitting* technique and the novel idea of *factor revealing LP* to achieve a ratio of 1.861 in time $O(n^2 \log n)$. The recasting is essential for getting a constant factor, since it enables the use of the metric condition in the analysis. MMSV is adapted from the primal-dual schema to produce a feasible primal solution $(\boldsymbol{x}, \boldsymbol{y})$ and a infeasible dual solution $(\boldsymbol{\alpha}, \boldsymbol{\beta})$, such that the respective costs $cost(\boldsymbol{x}, \boldsymbol{y}) \leq cost(\boldsymbol{\alpha}, \boldsymbol{\beta})$. Denote the optimal solution as OPT, the dual fitting analysis shows that if the dual solution is shrunken by a suitable factor ρ (to be found later), it becomes feasible and ρ is the approximation factor. This is because once the fitted dual is feasible, we have $\frac{cost(\boldsymbol{\alpha}, \boldsymbol{\beta})}{\rho} \leq OPT$ by weak duality. Hence, $cost(\boldsymbol{x}, \boldsymbol{y}) \leq \rho OPT$. To analyze ρ, we aim to find a minimum factor $\rho = \rho_{min}$ that suffices the feasibility of the dual solution. For a tight ρ_{min}, a series of factor revealing LPs is constructed to encode the UFL problem instances and the corresponding solutions generated by the algorithm. The constraints of the LPs capture the execution of the algorithm and ρ_{min} is set to be the supremum/maximum of the objective values. In general, it is not easy to directly analyze this supremum. The common way is to use a computer-aided analysis to guess a value first and then provide a complicated formal proof for this value. Along this line, Jain *et al.* [74] modified the primal-dual algorithm of [104] to take into account the optimization of the total connection cost via reconnecting clients. Their new algorithm (known

[11] In the thesis, $\tilde{O}$ is the notation for quasi time which means the logarithmic factors in the runtime are suppressed.

as the JMS algorithm) can be easily recast back to another greedy algorithm, again showing the equivalence of greedy and primal-dual. With the same ideas of dual fitting and factor revealing LPs in the analysis of JMS, an improved factor of 1.61 in time $O\left(n^3\right)$ was obtained. Following Jain *et al.*'s result, Thorup [135] made a small change to yield an amortized analysis on JMS. This improves the algorithm's deterministic runtime to $O\left(n^2 \log n\right)$ while only losing 0.1 in the approximation factor. For the sparse graph setting, he also presented an algorithm based on JMS which runs in $\tilde{O}\left(m\right)$ time with a factor of 1.62.

LP-rounding algorithms Although the LP-rounding method requires weakly polynomial time to solve LP, in practice, once the size of an LP is not too large, the simplex method often offers a reasonable runtime. LP-rounding is actually one of the most generally applicable methods for the design of approximation algorithms. The first constant-factor approximation algorithm for the metric UFL was proposed by Shmoys *et al.* [129] using LP rounding. The algorithm achieved a ratio of 3.16 based on the *filtering* [100] and *clustered rounding* [99] techniques introduced by Lin and Vitter. Both of the techniques are fundamental since all existing LP-rounding algorithms for the metric UFL essentially adopt these as core ideas. In the following, we describe these techniques in more detail. First, let $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ and $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ denote the optimal primal and dual solutions of LPs (3.3) and (3.4) respectively, and $OPT_{LP} = F^* + C^*$ denote the optimal cost of the LP where $F^* = \sum_{i \in \mathcal{F}} f_i y_i^*$ (the optimal facility cost) and $C^* = \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} c_{ij} x_{ij}^*$ (the optimal connection cost). By strong duality, $OPT_{LP} = \sum_{j \in \mathcal{C}} \alpha_j^*$. Based on the value of $(\boldsymbol{x}^*, \boldsymbol{y}^*)$, a support graph $G = (\mathcal{F} \cup \mathcal{C}, E)$ of UFL can be constructed where $\forall i \in \mathcal{F}, j \in \mathcal{C}$: we include the edge (i, j) in E if $x_{ij}^* > 0$. The filtering process then converts $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ to a feasible fractional solution $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ with a nice closeness property. This property ensures that the connection cost c_{ij} associated with $\tilde{x_{ij}} > 0$ is not too big. The process essentially transforms G into a new graph G' with edge set E' containing all edges with $\tilde{x_{ij}} > 0$. More specifically, the filtering in [129] initially sets $\forall i \in \mathcal{F}, j \in \mathcal{C} : \tilde{x_{ij}} = x_{ij}^*$. Afterwards, for every client j, if any of its associated edges (i, j) in G has c_{ij} larger than a defined threshold g_j, filtering then sets $\tilde{x_{ij}} = 0$ while scaling the other $\tilde{x_{ij}}$'s and $\tilde{y_i}$'s to maintain feasibility. As a consequence, the edges in G' are g-close because $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ is feasible, and $\tilde{x_{ij}} > 0$ implies $c_{ij} \leq g_j$. Later, the clustered rounding technique in [129] is carried out on G' for rounding $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ to the desired integer solution $(\boldsymbol{x}, \boldsymbol{y})$. The technique has two steps: clustering and rounding. In the clustering step, the neighborhood of a client j in G' is defined as $N\left(j\right) = \{i \in \mathcal{F} : \tilde{x_{ij}} > 0\}$. G'

is iteratively partitioned into several disjoint clusters as follows: in iteration k, a client j in G' with the minimum g_j value is elected as a cluster center. The cluster P_k to be created consists of the center j, the neighborhood facilities $N(j)$, and all the other clients j' such that $N(j) \cap N(j') \neq \emptyset$. After P_k is constructed, set $G' = G' \backslash P_k$ before the iteration $k+1$. The clustering process terminates when all clients in G' are removed. It is easy to see that all the constructed clusters are disjoint (see Figure 3.1). The next step is rounding. In each cluster, the facility i with the smallest cost is opened by setting $y_i = 1$. Finally, every client connects to one of its closest open facilities via updating x_{ij}'s. In the analysis, it is not hard to see $\sum_{i \in \mathcal{F}} f_i y_i \leq \sum_{i \in \mathcal{F}} f_i \tilde{y}_i$ since the clusters are disjoint and only the cheapest facilities are opened. Moreover, for any client j, its connection cost can be bounded by $3g_j$ using triangle inequality and the neighborhood structure of the clusters. With a particularly defined g_j and a *randomized filtering* process, Shmoys *et al.* [129] eventually obtained the factor of 3.16.

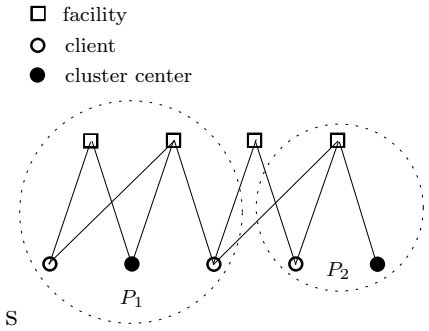

Figure 3.1: Clusters constructed from G'

Following Shmoys *et al.*'s algorithm for UFL, Chudak and Shmoys [41] observed that the optimal primal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is already α-close, i.e. $\forall i \in \mathcal{F}, j \in \mathcal{C} : x^*_{ij} > 0$ implies that $c_{ij} \leq \alpha^*_j$. This observation is deduced from the primal CSC $\forall i \in \mathcal{F}, j \in \mathcal{C}$: If $x^*_{ij} > 0$ then $\alpha^*_j - \beta^*_{ij} = c_{ij}$, and the nonnegative constraint $\beta_{ij} \geq 0$. Hence, by directly applying clustered rounding on G, we get $\sum_{i \in \mathcal{F}} f_i y_i \leq \sum_{i \in \mathcal{F}} f_i y^*_i \leq OPT$ and $\sum_{i \in \mathcal{F}} c_{ij} x_{ij} \leq 3\alpha^*_j = 3OPT$. This implies the clustered rounding technique alone (without filtering) is a simple 4-approximation algorithm. In fact, the main contribution of their work is the design and analysis of a *clustered randomized rounding* (CRR) algorithm with an improved ratio from randomization. Compared to the clustered rounding, CRR made three changes. Firstly, CRR in polynomial time transforms any input instance of UFL with its optimal fractional solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ to an equivalent instance with solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ for rounding. The transformed

solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}})$ is still α-close. It is also optimal because $\sum_{i\in\mathcal{F}} f_i\bar{y_i} = F^*$ and $\sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}\bar{x_{ij}} = C^*$. In addition, it possesses the property of *completeness*, i.e. $\forall i \in \mathcal{F}, j \in \mathcal{C}$: If $\bar{x_{ij}} > 0$ then $\bar{x_{ij}} = \bar{y_i}$. This useful property can greatly simplify the design and analysis of CRR. Secondly, in the clustering step, for every client j, let $\bar{C_j}$ denote its average fractional connection cost, that is, $\bar{C_j} = \sum_{j\in\mathcal{C}} c_{ij}\bar{x_{ij}}$. In each iteration, a client j with the minimum $g_j + \bar{C_j}$ value (instead of g_j) is elected as a cluster center. Here, $g_j = \alpha_j^*$. This modification takes into account the average connection costs for lowering down the total connection cost after rounding. Thirdly, randomness is added to the rounding step to further reduce the connection cost. Specifically, within a cluster P, each facility $i \in P$ is opened with probability $\bar{y_i}$. This ensures there is at least one open facility in each cluster because $\sum_{i\in P} \bar{y_i} = 1$. For each facility i' not included in any cluster, it is opened independently with probability $\bar{y_i'}$. In the end, every client connects to one of its closest open facilities. In the analysis, it is obvious that the total expected opening cost is exactly F^*. For the total expected connection cost, Chudak and Shmoys [41] analyzed the upper bound to be $C^* + \frac{2}{e}\sum_{j\in\mathcal{C}} \alpha_j^*$ using several probability lemmas and the triangle inequality. Adding these two cost bounds together, an approximation ratio of 1.736 is obtained.

Other improved LP-rounding algorithms are due to Sviridenko [131] and Byrka *et al.* [29]. Although both of the algorithms modify on CRR, the impact of the modification and the related improved analysis are quite involved in each case. Sviridenko's algorithm extends the filtering technique by combining it with the property of completeness. In addition, it chooses a filtering parameter from some defined *probability distribution* to effectively reduce the connection cost. The clustering procedure is essentially the same as CRR but it is performed on the filtered solution instead. The other contribution of the algorithm is the application of the *pipage rounding* technique [7] in the rounding step. The technique is a deterministic rounding process that takes advantage of the convexity/concavity of some cost function. In each iteration of the process, at least one fractional solution (e.g. $\bar{y_i}$) is rounded to an integer. The process terminates once there are enough integrally opened facilities which satisfy some desirable properties for bounding the solution cost. Based on these properties and the selected distribution for the filtering parameter, Sviridenko presented quite technical analysis to achieve a performance guarantee of 1.582. Byrka *et al.* [28, 29] in their algorithm used essentially the same filtering technique as Sviridenko's. However, the process of the filtering seems different and has a different impact on the ratio analysis. It first scales

all facility opening fractions (y_i^*'s) by a constant γ. Next, it splits facilities and updates connection fractions (x_{ij}^*'s) to maintain the property of completeness. Byrka *et al.* named their novel analysis of the filtering process as a *sparsening technique*. The technique considers regular and irregular UFL instances. For the irregular instances, it intuitively divides facilities serving a client j into close facilities $\mathcal{F}_j^C$ and distant facilities $\mathcal{F}_j^D$. The connections from the clients to their respective distant facilities are filtered out to reduce the total connection cost. Afterwards, the clustering (on the close facilities) and rounding steps are entirely in spirit of CRR, so one can simply view this algorithm as CRR with filtering/sparsening. In the analysis, bounding the total expected facility cost is easy since all facility opening fractions have been scaled by a constant γ. For tightly bounding the expected connection cost of a client j, Byrka *et al.* cleverly used the average distances to the facility sets $\mathcal{F}_j^C$, $\mathcal{F}_j^D$ and $\mathcal{F}_j^C \cup \mathcal{F}_j^D$ respectively in the analysis of connecting j under randomization. This further reflects the usefulness of the sparsening technique. Finally, taking $\gamma = 1.575$ [29] balances the total expected facility and connection costs and results in a single factor 1.575-approximation.

Notice that all of the above randomized LP-rounding algorithms can be derandomized using the *method of conditional expectation* (refer to [42] for more details) to yield an exact approximation ratio (rather than a randomized approximation ratio). Basically, this method always unravels an expectation into two conditional expectations. The smaller of these two expectations is chosen to be recursively unraveled further until a deterministic process is fully presented.

Local search and hybrid algorithms In practice, local search is a much more common approach in dealing with hard combinatorial optimization problems (see book [2] for a treatment of this subject). On one hand, local search algorithms/heuristics are generally easy to implement, and often yield good solutions at run time. In particular, these algorithms often perform very well in practice when enhanced with various ways of randomization. On the other hand, unlike primal-dual and LP-rounding, local search does not generate obvious comparable information. In most case, it is difficult to prove a strong performance guarantee for the solution found by local search [127]. In any local search algorithm, there is a search space of feasible solutions with a *global optimum*. Two feasible solutions in the space are defined as *neighbors* if one can be obtained/moved from the other by some specified set of local search operations. Defining such operations essentially defines a *neighborhood structure*.

Any *local optimum* in this structure is achieved when any specified move cannot lead to a better solution. A typical local search algorithm starts with a feasible solution and then iteratively moves to its neighbor with an improved solution cost until a local optimum is reached. In TCS, researchers are interested in constructing good neighborhood structures and analyzing their *locality gaps* [13, 14]. The locality gap of a neighborhood is the maximum ratio between its local optimum cost and the global optimum cost over all input instances. As shown in Table 3.1, the locality gap of a local search algorithm serves as an approximation ratio of the algorithm with an additional factor ϵ. The factor is used in the algorithm to control the improvement speed of the solution cost at each local search step. As a result, the number of local search steps can be polynomially bounded which delivers an approximation algorithm with a slightly weaker ratio. A trade-off of the solution quality and the runtime of the algorithm is also established depending on ϵ.

For the UFL problem, the study of a local search heuristic actually dates back to the early 1960s due to Kuehn and Hamburger [86]. After a long time, Korupolu *et al.* [83] analyzed this well-know heuristic and proved a locality gap of 5. The algorithm starts with an arbitrary feasible solution and then iteratively improves the solution towards to a local optimum. A feasible solution can be represented by a set of open facilities $I \subseteq \mathcal{F}$. This is because once I is fixed, every client simply connects to its closest open facility in I. The neighborhood in the algorithm is defined by add, delete and swap operations on facilities. Formally, let I be the current solution in the local search process, its neighborhood $N(I)$ is defined as the union of the sets $\{I + \{i'\} \mid i' \in \mathcal{F}\}$, $\{I - \{i\} \mid i \in I\}$ and $\{I - \{i\} + \{i'\} \mid i \in I,\, i' \in \mathcal{F}\}$. The algorithm has a runtime of $O\left(n^4 \log \frac{n}{\epsilon}\right)$ and once I becomes a local optimal, we have $cost(S) \leq 5cost(O)$ where O denotes the global optimal. Some time later, for the same algorithm, Arya *et al.* [13] presented an improved analysis to achieve a locality gap of 3. Moreover, Charikar and Guha [34] considered a slightly more sophisticated neighborhood and presented an easier analysis to reach the locality gap of 3. The neighborhood does not have the single swap move, instead it includes the operation of deleting one or more facilities. Formally, the neighborhood $N'(I)$ is defined as the union of the sets $\{I + \{i'\} \mid i' \in \mathcal{F}\}$ and $\{I - I' \mid I' \subseteq I\}$. This local search algorithm is about $O(n^2)$ faster than the previous heuristic due to randomization (which

produces a result with high probability[12]) and a linear time implementation of a gain function. For each facility $i' \in \mathcal{F}$, the value of the function $gain\,(i')$ from the current solution I is the total cost saving incurred by the operations in $N'\,(I)$. For instance, for the add operation, $gain\,(i') = C - C' - f_{i'}$ where C and C' denote the connection costs before and after the operation respectively. It is also possible that $gain\,(i') \leq 0$ but the local search process only continues when $\exists i' :\ gain\,(i') > 0$.

In Chapter 2, we have introduced the concept of ρ-approximation. For a minsum facility location problem, there are usually two costs (the total facility cost $\sum_{i\in\mathcal{F}} f_i y_i$ and connection cost $\sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij} x_{ij}$) in its objective. This drives the following definition of *bi-factor approximation*.

Definition 3.1.8. An algorithm is bi-factor $(\rho_f,\ \rho_c)$ or single factor $\max\,(\rho_f,\ \rho_c)$-approximation for a facility location problem P, iff for every instance $\mathcal{I}$ of P and any feasible solution SOL of $\mathcal{I}$ with total facility cost F_{SOL} and connection cost C_{SOL}, the total cost produced from the algorithm is at most $\rho_f F_{SOL} + \rho_c C_{SOL}$.

In the above definition, SOL can be either the optimal integer solution denoted as OPT or the optimal fractional solution OPT_f. With this definition, one can design an algorithm with different tradeoffs of facility cost and connection cost. Moreover, several algorithms with different bi-factor results can be combined into a hybrid algorithm (possibly with other tricks) to achieve an overall better single factor approximation (as the ultimate goal). In the following, we describe the algorithms with bi-factor results and the tricks of *cost scaling* and *greedy augmentation* to adjust or balance the achieved factors. We then show how algorithms with different bi-factor results can be combined for a better approximation.

Shmoy's first LP-rounding algorithm is $\left(\frac{1}{\alpha},\ 3g\,(\alpha)\right)$-approximation where $g\,(\alpha)$ is a filtering function depending on a parameter α. The ratio of Chudak and Shmoys's CRR algorithm is actually $\left(1,\ \frac{2}{e}\right)$ (e is the base of natural logarithm). For the primal-dual JMS algorithm, Jain *et al.* displayed an approximation tradeoff curve of ρ_f and ρ_c based on the computed solutions from the factor revealing LP. They also proved the existence of a bi-factor (1, 2) on the curve. Moreover, by adapting the hardness proof of Theorem 3.1.6, they have shown the following important bi-factor hardness result for UFL.

[12] A randomized algorithm is said to work with high probability if the chance of the failure event is at most an inverse polynomial n^{-c} where n describes the input size and c is a positive constant.

Theorem 3.1.9. *[73] The metric UFL cannot be approximated by an algorithm with a bifactor* (ρ_f, ρ_c) *such that* $\rho_c < 1 + 2e^{-\rho_f}$ *unless* $\mathcal{NP} \subseteq DTIME[n^{O(\log\log n)}]$.

Mahdian *et al.* [106] further analyzed JMS and proved a (1.11, 1.78) bi-factor on the approximation curve. Byrka *et al.* [27, 29] established another approximation curve described by the bi-factor $(\gamma, 1 + 2e^{-\gamma})$ for the scaling parameter $\gamma \geq 1.6774$. This implies the algorithm is (1.6774, 1.3738)-approximation. More importantly, according to the bifactor hardness result, this is the first algorithm that touches the approximation lower bound. For the two different local search algorithms [83, 34] described before, both of them are actually $(2 + \epsilon, 3 + \epsilon)$-approximation.

Now we describe the trick of cost scaling [34, 35] and show how it impacts the bi-factor approximations of UFL. In fact, cost scaling is a preprocessing procedure applicable to all minsum facility location problems with separate facility and client costs. It is captured by the following lemma[13].

Lemma 3.1.10. *For any instance* I *of a facility location problem, suppose that there is an algorithm* A *which produces facility and connection costs* F_A *and* C_A *such that* $\gamma F_A + \beta C_A \leq \rho_f F_{SOL} + \rho_c C_{SOL}$, *where* SOL *is any feasible solution to* I. *After scaling all facility costs of* I *by* λ, *running* A *on the scaled instance and then scale the costs back to the original, the costs obtained* F *and* C *satisfies* $\lambda\gamma F + \beta C \leq \lambda\rho_f F_{SOL} + \rho_c C_{SOL}$.

Proof. After scaling, SOL is also any feasible solution to the scaled instance with costs λF_{SOL} and C_{SOL}. Running A on the scaled instance then yields $\gamma F'_A + \beta C'_A \leq \lambda\rho_f F_{SOL} + \rho_c C_{SOL}$ where F'_A and C'_A are the produced costs with respect to the scaled instance. After scaling back, the obtained costs are clearly $F = \frac{F'_A}{\lambda}$ and $C = C'_A$, then lemma follows. □

In the above lemma, γ, β, and λ are positive constants. Algorithm A is (ρ_f, ρ_c)-approximation when $\gamma, \beta \geq 1$. The overall algorithm (Algorithm A with scaling) becomes $(\lambda\rho_f, \rho_c)$-approximation when $\lambda \geq \frac{1}{\gamma}$ and $\beta \geq 1$. As an application of cost scaling, let the algorithm A be the primal-dual JV algorithm [77] which produces the costs $3F_A + C_A \leq 3F_{SOL} + 3C_{SOL}$. In this case, $\gamma = 3$, $\beta = 1$, and $\rho_f = \rho_c = 3$. After applying the lemma, we

[13]Note that we show a more general lemma of cost scaling here. This lemma may also be presented in a different form if the costs F_A and C_A have separate bounds. This is usually the case of a local search algorithm.

get $3\lambda F_A + C_A \leq 3\lambda F_{SOL} + 3C_{SOL}$. By setting $\lambda = \frac{1}{3}$, we have a $(1, 3)$-approximation algorithm. Furthermore, for the local search algorithms, their ratio can be improved to $2.414 + \epsilon$ by cost scaling.

Greedy augmentation [64, 34] is essentially a postprocessing local search procedure only with the add operation. The difference in the operation is, for each facility $i' \in \mathcal{F}$, the value of the gain function is defined as $gain\,(i') = \frac{C - C' - f_{i'}}{f'_i}$. The procedure then augments a feasible solution of UFL as follows: while there exists facilities with positive gains, among these facilities, the facility i with the largest $gain\,(i)$ is opened, and every client reconnects to its nearest facility. The effect of this procedure is shown in the following lemma. The original proof of this lemma is due to [64, 34].

Lemma 3.1.11. *For any instance I of UFL, suppose that there is an initial feasible solution with facility cost F and connection cost C. Let SOL be any feasible solution to I with costs F_{SOL} and C_{SOL}, after greedy augmentation, the total cost is at most $F + F_{SOL} \max\left\{0, \ln\left(\frac{C - C_{SOL}}{F_{SOL}}\right)\right\} + F_{SOL} + C_{SOL}$.*

With a slightly different form of the above lemma, Guha and Khuller (1998) [64] considered a stronger LP relaxation of UFL and applied greedy augmentation to the algorithm of Shmoys *et al.* [129] (with a bi-factor $\left(\frac{1}{\alpha}, 3g\,(\alpha)\right)$) to reduce the ratio from 3 to 2.408. So far, we have seen that cost scaling can adjust the produced facility cost and greedy augmentation reduces the connection cost through opening more facilities. For a facility location problem, it is conventional that cost scaling and greedy augmentation (CSGA) are applied one after another. As shown in the theorem below, these tricks can be combined and then analyzed together for UFL.

Theorem 3.1.12. *For any instance I of UFL, suppose that an algorithm A produces facility cost F_A and connection cost C_A such that $\gamma F_A + \beta C_A \leq \rho_f F_{SOL} + \rho_c C_{SOL}$, where SOL is any feasible solution to I. After cost scaling with ratio λ and then greedy augmentation, the overall algorithm is $\left(\frac{\rho_f}{\gamma} + \ln\left(\frac{\lambda\gamma}{\beta}\right), 1 + \frac{\rho_c - \beta}{\lambda\gamma}\right)$-approximation.*

Proof. After cost scaling, from Lemma 3.1.10, it produces a solution of total cost T with facility cost F and connection cost C such that $\lambda\gamma F + \beta C \leq \lambda\rho_f F_{SOL} + \rho_c C_{SOL}$. In Lemma 3.1.11 with these T, F and C, consider two disjoint cases: $C < F_{SOL} + C_{SOL}$ and $C \geq F_{SOL} + C_{SOL}$. The first case yields:

$$
\begin{aligned}
T = F + C &= \frac{\lambda\gamma F + \beta C}{\lambda\gamma} + \left(1 - \frac{\beta}{\lambda\gamma}\right) C \\
&< \frac{\lambda\rho_f F_{SOL} + \rho_c C_{SOL}}{\lambda\gamma} + \left(1 - \frac{\beta}{\lambda\gamma}\right)(F_{SOL} + C_{SOL}) \\
&= \left(\frac{\rho_f}{\gamma} + 1 - \frac{\beta}{\lambda\gamma}\right) F_{SOL} + \left(1 + \frac{\rho_c - \beta}{\lambda\gamma}\right) C_{SOL}
\end{aligned}
$$

The second case implies $\ln\left(\frac{C - C_{SOL}}{F_{SOL}}\right) \geq 0$, together this with Lemma 3.1.11 and 3.1.10 yields:

$$
T \leq F + F_{SOL} + C_{SOL} + F_{SOL} \ln\left(\frac{\lambda\rho_f F_{SOL} + (\rho_c - \beta) C_{SOL} - \lambda\gamma F}{\beta F_{SOL}}\right)
$$

The derivative of the above right hand side value rhs with respect to F is:

$$
rhs' = 1 - \frac{\lambda\gamma F_{SOL}}{\lambda\rho_f F_{SOL} + (\rho_c - \beta) C_{SOL} - \lambda\gamma F}
$$

rhs is maximized when $rhs' = 0$, that is $F = \left(\frac{\rho_f}{\gamma} - 1\right) F_{SOL} + \left(\frac{\rho_c - \beta}{\lambda\gamma}\right) C_{SOL}$. Together with the assumption of this case, we have $T \leq \left(\frac{\rho_f}{\gamma} + \ln\left(\frac{\lambda\gamma}{\beta}\right)\right) F_{SOL} + \left(1 + \frac{\rho_c - \beta}{\lambda\gamma}\right) C_{SOL}$. By comparing this expression with the one in the first case, the theorem then follows because $1 - \frac{\beta}{\lambda\gamma} \leq \ln\left(\frac{\lambda\gamma}{\beta}\right)$. □

Mahdian *et al.* [106] in their MYZ algorithm applied a simpler form of the above CSGA result to the JMS algorithm's bifactor result (1.11, 1.78). In this case, we have $\gamma = \beta = 1$, $\rho_f = 1.11$, and $\rho_c = 1.78$. The value of λ will be fixed later. After applying the theorem, an algorithm with factor $\left(1.11 + \ln\lambda,\ 1 + \frac{0.78}{\lambda}\right)$ is obtained. For the best single factor, setting $\lambda = 1.504$ makes $1.11 + \ln\lambda = 1 + \frac{0.78}{\lambda} \approx 1.52$. For runtime, JMS and greedy augmentation both take $O(n^3)$. Therefore, the overall MYZ algorithm is 1.52-approximation in $O(n^3)$. Mahdian *et al.* subsequently in their journal paper [108] discretized the CSGA stage of MYZ into a constant number of rounds to get an improved runtime analysis and a novel factor revealing based ratio analysis. Together with the result of Thorup [135] for JMS, Mahdian *et al.* claimed that their MYZ algorithm runs in quasi-linear time even for the sparse graph setting.

Lastly, we show how the best single factor results are obtained through combining the algorithms with different bi-factor results. The combination involves randomization. Byrka *et al.* [27, 29] used the factor (1.6774, 1.3738)

of their algorithm (when the scaling factor $\gamma = 1.6774$) and combined it with the (1.11, 1.78) factor of the JMS algorithm. Let $A1(\gamma)$ and $A2$ denote these two algorithms respectively. If $A1(1.6774)$ is run with probability p and $A2$ with probability $1-p$, an expected approximation ratio of $(1.11 + 0.5674p,\ 1.78 - 0.4062p)$ is obtained. By setting $p = 0.6882$ to make the bi-factor terms equal, the ratio of 1.5 can be achieved. To derandomize the algorithm, one can simply always choose the better solution of the two produced from $A1(1.6774)$ and $A2$. Later, Li's more refined analysis in [89] obtained the current best ratio of 1.488. Basically, he used a game theory approach inspired by Byrka *et al.* [29] to analyze an explicit distribution for γ and a value of p. In the game, there are two players. One player is the algorithm designer who plays a distribution of γ and a value p to minimize the approximation ratio. The other player is the adversary who plays an instance characteristic function to maximize the ratio. The value of the game is 1.488 with the analyzed γ, p, and the characteristic function. The final combined algorithm is as follows: With probability about 0.2, run $A2$; with probability about 0.5 run $A1(1.5)$; with the remaining 0.3 probability, run $A1(\gamma)$ where γ is selected uniformly from the range (1.5, 2).

3.2 Other facility location problems

In this section, we shall briefly look at some other facility location variants which model different application scenarios. Some of the problems are closely related to the resource allocation problems introduced in Chapter 1. These are the metric *Fault-Tolerant Facility Location*, *Capacitated Facility Location,* and *k Uncapacitated Facility Location* problems to be discussed in more detail in the following subsections. Other our interested problems are described below. Most of the solutions to these metric facility location variants are inspired from the algorithms we have seen for the metric UFL.

Facility Location with Outliers (FLO) Charikar *et al.* [37] introduced the notion of *outliers*. It captures the situation that there are sometimes a few very distant clients in a facility location model (e.g. UFL or k-median) which may increase the cost of the solution dramatically. These distant clients are the expensive outliers and they do not have to be serviced. Built on UFL, there are two types of FLO problems. The first is called *Facility Location with*

Penalties[14]. In this problem, a client j can be chosen not to connect to any facility by paying a penalty p_j instead. If we let r_j be an indicator variable (which takes the value 0 or 1) for deciding whether j is connected or not, the objective function to minimize is $\sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x_{ij} + \sum_{j\in\mathcal{C}} p_j r_j$. Charikar *et al.* [37] adapted the JV algorithm for UFL to solve this problem with 3-approximation. To date, the best result is due to Li *et al.* [92] who achieved a factor of 1.52 by extending the 1.488-approximation algorithm for UFL. The other FLO problem is *Robust Facility Location*. In this problem, a number l is given as the number of allowable outliers, so only $n_c - l$ clients are required to connect to open facilities while minimizing the total facility and connection cost. Charikar *et al.* [37] used the technique of parametric pruning and combined it with the JV algorithm for UFL to obtain 3-approximation. Mahdian *et al.* [104] replaced the JV algorithm with their MMSV algorithm to get an improved ratio of 2.

Multi-level Facility Location ($MLFL$) $MLFL$ models a hierarchical supply chain network with several levels of facilities like plants, warehouse, and distribution centers. Let k denote the number of levels. Every client in the network needs to be serviced by k facilities in order with each from a different level. Formally, we are given k sets of facilities and a set of clients, every client j has to be serviced by a path that connects j with one facility from each facility set. The objective is to open a subset of facilities in every facility set to serve all j's such that the sum of facility opening and path connection costs is minimized. UFL is the same as $MLFL$ with $k = 1$. For the case $k = 2$, Shmoys *et al.* [129] used filtering and rounding to give the first constant factor approximation algorithm with a ratio of 3.16. For the general k, Aardal *et al.* [1] used randomized rounding to get an approximation factor of 3. Ageev *et al.* [5] showed a recursive reduction which reduces a k-level problem to a $k-1$ level problem. Later, Zhang [151] combined randomized rounding and dual-fitting to get the current best ratio of 1.77 for $k = 2$. Recently, Byrka and Rybicki [30] presented a general algorithm with improved ratios for the cases $k > 2$. They have also shown that the ratio of their algorithm converges to 3 as $k \to +\infty$. Several hardness results for $MLFL$ with $k \geq 2$ were presented in [85]. The hardness approximation factors proved therein are 1.539 for $k = 2$ and 1.61 for the general k. These results indicate that the general $MLFL$ is computationally harder than UFL.

[14]This problem is also called the prize-collecting facility location.

Online Facility Location (OFL) The problems we have considered so far are offline problems in which the entire input is available from the start. In an online problem, part of the input arrives gradually online and the corresponding *online algorithm* keeps making decisions based on the currently available input. The online problem serves to model real-time and dynamic environments. To analyze the solution quality of an online algorithm, we have the notion of *competitive ratio* that compares the solution of an online algorithm with that of an optimal offline algorithm. There are two types of OFL problems built on UFL. The first is called the *online median* problem where a total of k facilities (k is not known in advance) are placed one at a time. In addition, a facility cannot be moved once it is placed and all client-side input data are known beforehand. After proposing this problem, Mettu and Plaxton [113] gave an elegant linear-time hierarchically greedy algorithm that is $O(1)$-competitive. In the second OFL problem, clients arrive one at a time and all facility-side input data are pre-known instead. Upon arrival, the clients must be assigned irrevocably to either an existing facility or a newly opened facility. This problem was proposed by Meyerson [115]. He showed that his randomized algorithm is $O(1)$-competitive for the case where clients arrive in random order, and $O(\log n)$-competitive for the worst case. Fotakis [54] further studied this problem and came up with a deterministic algorithm having a competitive ratio of $O\left(\frac{\log n}{\log \log n}\right)$.

3.2.1 Fault-Tolerant Facility Location

The Fault-Tolerant Facility Location ($FTFL$) problem was originally introduced by Jain and Vazirani [76]. The only difference between $FTFL$ and UFL is that $FTFL$ requires at least r_j distinct connections to open facilities instead of just one. This simple change adds robustness to the UFL model against connection failures but brings in a more difficult problem to solve. $FTFL$ can also be modeled by an integer program. The LP-relaxation of the program and the corresponding dual are the following. Notice that the third constraint of LP (3.5) is necessary to ensure a valid relaxation. This constraint can also be replaced by $\forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \leq 1$.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij} x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \geq r_j \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : y_i \leq 1 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij}, y_i \geq 0 \end{array} \tag{3.5}$$

$$\begin{array}{ll} \text{maximize} & \sum_{j\in\mathcal{C}} r_j\alpha_j - \sum_{i\in\mathcal{F}} z_i \\ \text{subject to} & \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} \beta_{ij} \leq f_i + z_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j - \beta_{ij} \leq c_{ij} \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j, \beta_{ij}, z_i \geq 0 \end{array} \tag{3.6}$$

$FTFL$ reduces to UFL if $\forall j \in \mathcal{C} : r_j = 1$. From the perspective of approximation, $FTFL$ seems to be more difficult than UFL, since there does not exist any constant factor combinatorial algorithm unless the problem is uniform (all r_j's are equal). For the general non-uniform case, the first algorithm is also by Jain and Vazirani. It is a primal-dual algorithm that yields a non-constant factor of $3 \cdot H_k$ where $k = \max_j r_j \leq n_f$[15] is the maximum requirement of the clients. The algorithm runs *in phases* and decomposes an $FTFL$ instance into multiple UFL instances. It reduces k by one in each phase, that is, in each phase it only considers the clients with k as their residual requirements and makes one connection for these clients. The problem in each phase is essentially UFL which is solvable by an UFL algorithm with a ratio ρ. In addition, the algorithm terminates once $k = 0$. In the analysis, we first let OPT_f denote the optimal cost of LP (3.5). Jain and Vazirani gave an LP model for the decomposed UFL instances and proved the optimal cost of this LP is at most $\frac{OPT_f}{k}$. Adding up the costs of the UFL instances for $1 \leq k \leq \max_j r_j$ eventually results in a ratio of ρH_k ($\rho = 3$ in [76] since in each phase it uses the JV algorithm [75] for UFL).

Constant factor results for the general case are all based on LP-rounding. Guha et al. [66, 67] obtained the first constant factor approximation algorithm. The algorithm rounds the optimal fractional solution of an alternative LP formulation for $FTFL$. Specifically, in the formulation, each client j is associated with r_j copies. Each copy j^r $(1 \leq r \leq r_j)$ is required to connect to one open facility and it is associated with a variable x_{ij}^r. The algorithm first converts the optimal solution to a structured fractional solution with certain useful properties. The structured solution is then filtered and rounded by adapting the techniques [129] for UFL. The key difference is that the algorithm clusters facilities around each copy. It also uses an *uncrossing step* to ensure different copies of a client connect to different facilities and thereby maintaining the solution feasibility. Guha *et al.* additionally showed that the greedy augmentation [64, 34] technique for UFL can be adapted for $FTFL$

[15] H_k is the kth harmonic number, i.e. $H_k = \sum_{i=1}^{k} \frac{1}{i}$. The factor can be also stated as $O(\log n)$.

with the same impact (see Lemma 3.1.11). Applying this to the solution produced from filtering and rounding leads to a factor of 2.408. Later, this result was improved to 2.076 by Swamy and Shmoys [132] with more sophisticated rounding techniques. The main source of the improvement in their work is pipage rounding [7] which was similarly applied to UFL by Sviridenko [131]. Recently, Byrka et al. [31] obtained the current best ratio of 1.7245. Their work adapts the best rounding algorithm and analysis [28, 29] for UFL. In addition, they applied the *dependent rounding* (DR) technique [57] and invented a novel *laminar clustering* (LC) technique. DR is a polynomial time randomized algorithm that rounds a fractional vector to a random integral vector. It can be viewed as a randomized variant of pipage rounding with more manageable properties. LC causes the clustered facilities (the subsets of facilities) to have the structure of not being disjoint and having different sizes, i.e. any two clusters are constructed to be either disjoint or one contains the other. This laminar structure of the facilities exploits DR to ensure both the solution feasibility and the reduced solution cost from randomization.

For the uniform case of $FTFL$, Bumb [26] adapted the primal-dual JV algorithm [75] for UFL to achieve a ratio of 1.853. Jain et al. [73, 103] showed their MMSV and JMS algorithms for UFL can be adapted to the uniform $FTFL$ while preserving the ratios of 1.861 and 1.61 respectively. Swamy and Shmoys [132] further improved the result to 1.52 using cost scaling and greedy augmentation. For the uniform resource allocation problems, we follow the same line as these works for $FTFL$.

3.2.2 Capacitated Facility Location

Capacitated Facility Location (CFL) is a well-studied facility location problem with numerous applications and solutions. It differs from UFL in that every facility has a capacity, i.e., it can only serve a limited number of demands/requests from clients. CFL has the following standard LP formulation in which d_j is the demand of a client j, u_i is the capacity of a facility i, and the third constraint encodes the capacity constraint. All d_j's and u_i's are usually assumed to be positive integers.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} d_j c_{ij} x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \geq 1 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} d_j x_{ij} \leq u_i y_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : y_i \in \{0,1\} \end{array} \tag{3.7}$$

In the TCS literature, several variants of CFL were studied from the perspective of approximation. These variants have different properties and solutions. One major distinction is between *hard* and *soft* capacities, so CFL can be renamed as $HCFL$ and $SCFL$ respectively. LP (3.7) models the $HCFL$ problem in which each facility is either opened or not. If the nonnegative constraint of y_i is relaxed to $\forall i \in \mathcal{F} : y_i \in \mathbb{Z}^+$ so that multiple copies of a facility with the same capacity can be opened, the problem becomes $SCFL$. Without loss of generality, we could set $\forall j \in \mathcal{C} : d_j = 1$ for ease of exposition. After setting this, the nonnegative constraint of x_{ij} can be changed to $x_{ij} \in \{0,1\}$ for $HCFL$ and $x_{ij} \in \mathbb{Z}^+$ for $SCFL$. Conventionally, $HCFL$ is said to be *uniform* if $\forall i \in \mathcal{F} : u_i = u$ and *nonuniform* otherwise. $SCFL$ is much easier than $HCFL$ in terms of the integrity gaps (IGs) of the respective LP formulations. In particular, Shmoys *et al.* [129] showed that the IG of $HCFL$ is unbounded while Mahdian *et al.* [107] proved that the IG of $SCFL$ is at least 2. These results are reasonable since $SCFL$ allows any violation of the hard capacity u_i, e.g., the capacity of a facility location i becomes ku_i if k copies of a facility are needed to open at i. The other distinction is between *splittable* and *unsplittable* demands and the respective problems are $CFLS$ and $CFLU$. Notice that this distinction is not meaningful ($CFLS$ and $CFLU$ become the same) if $\forall j \in \mathcal{C} : d_j = 1$. As a consequence, it only concerns the case when not all of the clients have single demands. Both $HCFL$ and $SCFL$ belong to $CFLS$ due to the constraint $\forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \geq 0$. This enables each client's demand to be fractionally split among more than one open facilities. If the constraint becomes $\forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \in \{0,1\}$, that is, the entire demand of any client j must be supplied by a unique facility, LP (3.7) then changes to a formulation for $CFLU$. From the perspective of approximation, $CFLU$ is much harder than $CFLS$. Bateni and Hajiaghayi [21] observed that it is $\mathcal{NP}$-hard to obtain any bounded approximation ratio for $CFLU$ assuming its hard capacity is not violated. On the other hand, many constant factor approximation algorithms have appeared for $CFLS$. Another variant of CFL is the Universal Facility Location ($UniFL$) problem [105]. $UniFL$ actually

generalizes UFL, $CFLS$, and some other facility location problems due to its specially defined facility cost function.

Because the $HCFL$ LP has an unbounded IG, nearly all non-trivial approximation algorithms for $HCFL$ are local search rather than LP based. The first algorithm is from Korupolu *et al.* [83], for the uniform $HCFL$. The algorithm uses the operations of add, delete and swap on facilities to achieve a ratio of $8+\epsilon$. For the uniform case, Chudak and Williamson [40] simplified and improved the analysis of [83] to get an improved factor of $6+\epsilon$. This result was further improved to $5.83+\epsilon$ with cost scaling (CG) [34]. Recently, Aggarwal *et al.* [8] obtained the current best ratio of $3+\epsilon$ by changing the swap operation (also the analysis), and using linear combinations to capture the local optimality. They have also shown their achieved ratio is tight. For the nonuniform $HCFL$, the first algorithm is by Pal *et al.* who achieved a performance guarantee of $9+\epsilon$ (or $8.53+\epsilon$ with CG). Their algorithm uses two more powerful local search operations called open and close. The operations allow opening (respectively closing) a facility and closing (respectively opening) a set of facilities at a time. Zhang *et al.* [152] extended the operation set further with a multi-exchange operation to reduce the ratio to $5.83+\epsilon$. Recently, Bansal *et al.* [18] modified the open, close and multi-exchange operations to obtain the current best ratio of $5+\epsilon$. In the analysis, they adopted to the idea of linear combinations [8]. This ratio is shown to be tight from the example provided by Zhang *et al.* [152]. For $SCFL$, different approximation techniques can be applied. For the uniform $SCFL$, the first algorithm is filtering and rounding [129] based which yields a ratio of 5.69. Chudak and Shmoys [39] used randomized rounding to get a factor of 3. For the nonuniform case, Arya *et al.* [13] presented a 3.73-approximation local search algorithm. The other authors attempted to reduce $SCFL$ to UFL in a series of papers [75, 35, 74, 106, 107]. The best reduction method [107] adopts the concept of bifactor approximate reduction and the LP-based JMS algorithm [74] for UFL to achieve a ratio of 2. This result is the best possible LP approach as it matches the IG of $SCFL$. In Chapter 4, we also study a soft capacitated resource allocation problem based on reduction. For $CFLU$, the research so far have focused on its hard capacitated variant (denoted as $HCFLU$) since the soft capacitated variant is much easier. Furthermore, $HCFLU$ is a special case of the generalized assignment problem (GAP). It also generalizes other problems such as knapsack, bin packing, and minimum makespan scheduling. For the uniform

case, Shmoys *et al.* [129] first presented a $(9, 4)$[16]-approximation algorithm. They used the rounding framework [128] for GAP to transform an LP-based solution to $HCFL$ to a solution to $HCFLU$ with cost and capacity blowups. With the same framework, one can transform the best local search solutions for $HCFL$ to obtain factors $(5, 2)$ and $(11, 2)$ for the uniform and nonuniform $HCFLU$ respectively [22]. Besides these results, for a smaller capacity blowup, Bateni and Hajiaghayi [21] gave a $(O(\log n), 1 + \epsilon)$ approximation algorithm. Behsaz *et al.* [22] obtained the first constant factor approximation algorithms with capacity violation less than 2.

3.2.3 K Uncapacitated Facility Location

The k Uncapacited Facility Location (k-UFL) problem is a common generalization of the fundamental problems UFL and k-median (both have been defined in Chapter 1). It differs from UFL in that at most k facilities can be opened. Also, it becomes the k-median problem when its facility costs are zero. The k-center and k-means are the other two closely related problems to k-UFL. The following are the LP-relaxation and dual LP of the problem in which the third constraint of LP (3.8) is a hard global constraint and its corresponding dual variable in LP (3.9) is z.

$$\begin{aligned}
\text{minimize} \quad & \textstyle\sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij} x_{ij} \\
\text{subject to} \quad & \forall j \in \mathcal{C} : \textstyle\sum_{i\in\mathcal{F}} x_{ij} \geq 1 \\
& \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\
& \textstyle\sum_{i\in\mathcal{F}} -y_i \geq -k \\
& \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij},\, y_i \geq 0
\end{aligned} \tag{3.8}$$

$$\begin{aligned}
\text{maximize} \quad & \textstyle\sum_{j\in\mathcal{C}} \alpha_j - zk \\
\text{subject to} \quad & \forall i \in \mathcal{F} : \textstyle\sum_{j\in\mathcal{C}} \beta_{ij} \leq z \\
& \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j - \beta_{ij} \leq c_{ij} \\
& \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j,\, \beta_{ij},\, z \geq 0
\end{aligned} \tag{3.9}$$

The approximation results for k-UFL mostly follow from the results for UFL and k-median. We have already seen the results for UFL. Next, we shall briefly look at the results for both k-median and k-UFL.

The k-median problem has numerous applications in clustering and data mining [109], network design [121], etc. Although it has apparent similarity

[16] An (α, β) bifactor approximation here essentially means that an algorithm returns a solution cost within α times of the optimal and violates the capacity within factor β. This is different from the definition for UFL we have seen before.

to UFL, due to the hard constraint on the number of open facilities, the algorithms for this problem and the generated results are somewhat different. In the following, we list several hardness results for k-median. These results improve its inherited hardness results from the metric UFL (with a ratio of 1.463) and carry over to k-UFL.

Theorem 3.2.1. *[63] The complete metric k-median (where $\mathcal{F} = \mathcal{C}$) cannot be approximated with a ratio smaller than $1+\frac{1}{e}$ unless $\mathcal{NP} \subseteq DTIME[n^{O(\log\log n)}]$.*

Theorem 3.2.2. *[74] The metric k-median cannot be approximated with a ratio smaller than $1 + \frac{2}{e}$ unless $\mathcal{NP} \subseteq DTIME[n^{O(\log\log n)}]$.*

Theorem 3.2.3. *[91][17] For the metric k-median, the integrity gap of the standard LP is at least 2.*

For the approximation algorithms, Lin and Vitter made the first attempt to solve the non-metric k-median. Their filtering and rounding algorithm achieves a bifactor of $\left[1+\epsilon,\, \left(1+\frac{1}{\epsilon}\right)(\ln n + 1)\, k\right]$ [100], that is, for any $\epsilon > 0$, the algorithm produces a solution for which the cost is within $1+\epsilon$ of the optimum, but is infeasible since it uses $\left(1+\frac{1}{\epsilon}\right)(\ln n + 1)\, k$ facilities. For the more interesting metric case, they gave a solution with a ratio of $\left[2\,(1+\epsilon)\,,\, \left(1+\frac{1}{\epsilon}\right) k\right]$ [99] which opens much less facilities. Korupolu *et al.* [83] at the first time used the local search operations of add, delete and swap (similarly for UFL), achieving bifactor ratios of $\left[1+\epsilon,\, 3+\frac{5}{\epsilon}\right]$ and $\left[1+\frac{5}{\epsilon},\, 3+\epsilon\right]$. The first feasible solutions are actually due to Bartal [20] and Charika et al. [33] who gave factors of $O(\log n \log\log n)$ and $O(\log k \log\log k)$ respectively. One year later, Charika et al. [36] gave the first constant factor approximation algorithm with a performance guarantee of $6\frac{2}{3}$. Their algorithm combines the filtering technique and a more sophisticated rounding procedure. In the same year, Jain and Vazirani [75] improved this factor to 6 with their 3-approximation JV algorithm for UFL. They achieved this breakthrough result by applying the *Lagrangian relaxation* (LR) technique. Briefly speaking, the technique moves the hard constraint of a k-median or k-UFL LP to the objective function. This yields a UFL problem and enables the transformation of a ρ-approximation UFL algorithm with the *Lagrangian multiplier preserving* (LMP) property (also called an LMP ρ-approximation algorithm for UFL) to a 2ρ-approximation algorithm for k-median. In Chapter 4 and 6, we adopt the LR technique for approximating a capacitated and a constrained resource allocation problem respectively.

[17] We cite the recent reference here since it clearly illustrates such a gap instance.

Shortly after the result of Jain and Vazirani, Charika and Guha [34] added an augmentation step to their algorithm to obtain an improved factor of 4. Independently, Jain *et al.* [74] showed their JMS algorithm with cost scaling (see Lemma 3.1.10) is in fact LMP 2-approximation. This led to another 4-approximation algorithm for k-median. Following this line of research, Archer *et al.* [10] looked into the LMP property and suggested an algorithm with a factor of 3. However, their algorithm runs in exponential time which requires to solve the maximum independent set problem. Surprisingly, until the beginning of this year, the previous best approximation result is due to a local search algorithm [13, 14] by Arya *et al.* Their algorithm only uses the very effective multiswap operation to achieve a ratio of $3 + \epsilon$ in polynomial time. Recently, Charika and Li [38] designed a faster LP rounding algorithm with a factor of 3.25. The main feature of their algorithm is a clever multi-phase randomized procedure. The procedure exploits dependent rounding. Just this year, Li and Svensson [91] achieved the current best ratio of $1 + \sqrt{3} + \epsilon$. They designed a bifactor $\left[1 + \sqrt{3} + \epsilon,\, k + O\left(1\right)\right]$ pseudo-approximation algorithm and showed that the algorithm can be turned into a feasible algorithm for k-median with the same approximation ratio. For the k-UFL problem, Charika *et al.* [36] showed that a simple adaptation of their algorithm for k-median yields an approximation bound of 9.8. Since the LR technique applies to k-UFL as well, the results of [75] and [74] for k-median directly hold for k-UFL with factors of 6 and 4 respectively. Based on the local search algorithm of [13, 14] for k-median, Zhang [153] used the operations of add, delete and multiswap to achieve a ratio of $2 + \sqrt{3} + \epsilon$. Charika and Li's algorithm [38] for k-median directly implies the current best performance guarantee of 3.25 for k-UFL.

Chapter 4

Unconstrained Fault-Tolerant Resource Allocation

In this chapter, we study the *Unconstrained Fault-Tolerant Resource Allocation* ($FTRA_\infty$) problem[1]. This problem was first introduced by Xu and Shen [145] as a relaxation of the classical *Fault-Tolerant Facility Location* problem [76]. In the $FTRA_\infty$ problem, we are given a set of sites containing facilities as resources, and a set of clients accessing these resources. Each site i is allowed to open an unconstrained number of facilities with cost f_i for each opened facility. Each client j requires an allocation of r_j open facilities and connecting j to any facility at site i incurs a connection cost c_{ij}. The connection costs form a metric. The goal is to minimize the total facility and connection cost. We present three greedy approximation algorithms for the uniform $FTRA_\infty$ in Sections 4.2, 4.3, and 4.4. The algorithms run in pseudo-polynomial time and achieve factors of 1.861, 1.61, and 1.52 respectively. Each algorithm improves on the previous one while they are analyzed with a similar set of techniques inspired from the work for *Uncapacitated Facility Location* (UFL) (see Chapter 3). The runtimes of these algorithms can be turned into strongly polynomial without affecting the approximation ratios via some acceleration heuristics. For a better presentation, we postpone the discussions of the runtime improvements to Chapter 5 and 6. In Section 4.5, we show a simple reduction from the general $FTRA_\infty$ to UFL. The reduction also implies an approximation preserving reduction from the uniform $FTRA_\infty$ to UFL. In Section 4.6, we obtain factors of 2.89 and 3.722 for the uniform capacitated $FTRA_\infty$ using the algorithms

[1]The problem is also called *Fault-Tolerant Facility Allocation* ($FTFA$) in [145, 125] and *Fault-Tolerant Facility Placement* ($FTFP$) in [148]. Nevertheless, we use different names of resource allocation to identify different application-oriented resource allocation scenarios. Our naming convention also follows from [56, 72, 81] for the set cover problems.

designed for the uniform $FTRA_\infty$. The results of this chapter are based on the works in [94, 93].

4.1 Introduction

Many facility location models are built around the classical *Uncapacitated Facility Location* (UFL) problem. Since the late 1990s, UFL has been studied extensively from the perspective of approximation algorithms for its practical use in clustering applications and network optimization. With the practical focus on the later, the *Fault-Tolerant Facility Location* ($FTFL$) model was introduced, which adds tolerance to UFL against connection/communication failures between facilities and clients. As the computer world evolves, optimal resource allocation starts to become essential in many contemporary applications [32]. A typical example is today's cloud computing platform. In particular, cloud providers own data centers equipped with physical machines as resources. Each physical machine runs a few virtual machines to serve remote clients' leases for computing power. This new resource sharing paradigm inevitably brings up two optimization problems from different aspects: 1) how do *cloud providers* cost-effectively share their resources among a large number of clients? 2) how can *clients* optimally rent resources in the clouds that incur the minimum overheads? Through a careful look at these questions, it is interesting to see they may be roughly interpreted as facility location problems where facility costs with respect to (w.r.t.) providers are physical machine costs, whereas to clients, they are administration overheads; Connection costs w.r.t. providers are network costs (usually proportional to distances) and they are response times to clients. However, existing models like UFL and $FTFL$ are insufficient to capture the common resource allocation scenarios. They both restrict at most one facility/resource to open at each site/data center, and connection requests/leases of a client must be served by facilities from different sites. This first motivates us the study of the *Unconstrained Fault-Tolerant Resource Allocation* ($FTRA_\infty$) due to its elasticity on the number of resources to allocate. The elasticity also corresponds to a simpler combinatorial structure to approximate. From a practical point of view, $FTRA_\infty$ has its clear advantages over $FTFL$. It allows many facilities to operate as a cluster at every site, and clients to connect with different facilities within the same site to achieve both intra-site and inter-site fault tolerance or parallel access.

In addition, the capacitated $FTRA_\infty$ ($CFTRA_\infty$) model is a further boost to $FTRA_\infty$ by considering facility's capacity or workload limit.

Problem formulation $FTRA_\infty$ is formulated by the following integer linear program.

$$\begin{array}{lll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x_{ij} & \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \geq r_j & (4.11) \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 & (4.12) \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \in \mathbb{Z}^+ & \\ & \forall i \in \mathcal{F} : y_i \in \mathbb{Z}^+ & \end{array} \tag{4.1}$$

In the formulation, $\mathcal{F}$ and $\mathcal{C}$ denote the sets of facilities and clients respectively where $|\mathcal{F}| = n_f$ and $|\mathcal{C}| = n_c$; $\boldsymbol{f}$ and $\boldsymbol{c}$ denote the facility and connection costs respectively; $\boldsymbol{r}$ denotes the clients' connection requirements, that is, each client $j \in \mathcal{C}$ is required to connect with r_j facilities either from different sites or not, for the purpose of fault tolerance or parallel access. For convenience, let $n = n_f + n_c$ and $m = n_f \times n_c$. The size of the ILP is $O(m)$. In addition, variable y_i denotes in the solution the number of facilities to open at site i, and x_{ij} the number of connections between a client j and site i. Compared to $FTFL$, domains of the variables are relaxed to be non-negative rather than 0-1 integers. Hence, $FTRA_\infty$ forms a relaxation of $FTFL$. The objective function represents the total cost to minimize. This cost is the sum of the facility opening and connection costs. Constraint (4.11) ensures the clients' fault tolerance or parallel access levels are satisfied. Constraint (4.12) restricts that each client can establish at most one connection to an open facility of any site. For a smoother presentation, we introduce the LP formulation of $CFTRA_\infty$ in Section 4.6. Recall that for both $FTRA_\infty$ and $CFTRA_\infty$, the problem illustrations corresponding to their LP formulations can be captured by the unified resource allocation model (see Figure 1.1) displayed in Chapter 1.

Related work $FTRA_\infty$ was first introduced by Xu and Shen [145] in the year 2009. They designed a phase-greedy algorithm for the general case of the problem and claimed a ratio of 1.861. After that, in parallel of our work, the general case was also studied by Yan and Chrobak. They first presented a 3.16-approximation LP-rounding algorithm [148] by adapting the algorithms [129, 42] for UFL. Later, they strengthened their results with a factor of 1.575 [150], building on the more recent results on UFL and $FTFL$ [67, 28, 29]. A

more complete summary of their work is documented in Yan's PhD thesis [147]. For the problem hardness, the approximation lower bound is 1.463 unless $\mathcal{P} = \mathcal{NP}$. This is inherited from UFL since UFL is a special case of $FTRA_\infty$ when $\forall j \in \mathcal{C} : r_j = 1$. The lower bound also holds for the other resource allocation problems studied in this thesis. Just recently, Rybicki and Byrka [123] gave an asymptotic approximation algorithm (w.r.t $\min_j r_j$) and some improved hardness results for $FTRA_\infty$.

Techniques and results We first present two *star-greedy algorithms* for constructing different solutions to the uniform $FTRA_\infty$. The algorithms are inspired by the greedy algorithms [73] for UFL. They work for the general $FTRA_\infty$ as well. However, we are currently unable to analyze their constant approximation ratios unless the problem is assumed to be uniform. A star consists of a facility and a set of clients. In each iteration of a star-greedy algorithm, a star is picked as a part of the final complete solution according to a greedy objective. The algorithm terminates once all clients are fully connected, i.e., every client j has r_j connections. This is in contrast with the phase-greedy approach [76, 145, 125] in which multiple stars are selected in one step as each phase solves a UFL problem as a part of the solution. To analyze a star greedy algorithm, we first restate it as an equivalent primal-dual algorithm to utilize the metric properties and then apply a set of general analysis tools mostly learned from the solutions to UFL. In fact, we confirm that the greedy algorithms for UFL can be extended for the resource allocation problems. More precisely, for the first 1.861-approximation algorithm, we adopt a novel approach that *relaxes feasibilities* of the problem's primal complementary slackness conditions, rather than the conventional ways [77, 142] of relaxing primal or dual conditions themselves. For the second 1.61-approximation algorithm, we apply *dual fitting* [73] and *inverse dual fitting* [145, 125] techniques for its single factor and bi-factor ratio analysis respectively. We will see more use of the inverse dual fitting technique and its formalization in the next chapter. To achieve tight approximation factors, we adopt *factor-revealing LPs* for analyzing both of the algorithms.

With the techniques of *cost scaling* and *greedy augmentation*, a hybrid greedy algorithm can further improve a bi-factor result of the second star-greedy algorithm to 1.52-approximation. We also attempt to reduce $FTRA_\infty$

to UFL. The simple reduction relies on some *generic LP properties* (see Chapter 2). For $CFTRA_\infty$, we adopt the *Lagrangian relaxation* technique. It enables us to use the algorithms designed for $FTRA_\infty$ as subroutines to obtain factors of 2.89 and 3.722 for the uniform $CFTRA_\infty$.

4.2 A greedy algorithm with ratio 1.861

A naive way to solve $FTRA_\infty$ is to reduce an $FTRA_\infty$ instance to an $FTFL$ instance and then apply an algorithm for $FTFL$. For the reduction, we first observe that it is sufficient to allocate each site of $FTRA_\infty$ $\max_{j\in\mathcal{C}} r_j$ facilities. Afterwards, we split these sites by making each split site to have at most one facility to open. It will create an equivalent $FTFL$ instance with a possibly exponential size $O\left(n_f \max_{j\in\mathcal{C}} r_j + n_c\right)$. This is because in $FTFL$, $\max_{j\in\mathcal{C}} r_j \leq n_f$, while r_j can be much larger than the input size n_f in $FTRA_\infty$. On the other hand, our solutions to $FTRA_\infty$ are based on two star-greedy algorithms. These algorithms do not differ much from each other, but their analyses are quite involved in each case. The algorithms both run in pseudo-polynomial time depending on r_j. The runtimes can be turned into strongly polynomial without affecting the approximation ratios. In the following, we present the first star-greedy algorithm together with its analysis.

The algorithm iteratively picks the star with the least average cost (the most cost-effectiveness). It terminates once all clients' connection requirements are satisfied. Specifically, in Algorithm 4.1, we incrementally build the solution y_i's and x_{ij}'s which are initially set to 0. Their values will then increase according to the stars being picked. We let the set $\mathcal{U}$ contain all clients that have not fulfilled their connection requirements. In order to ensure the feasibility of the solution, two conditions need to be met while iteratively choosing the stars: 1) the previously opened and used facility at a site will have zero opening cost in the next iteration; 2) validity of stars, i.e., a star to be chosen only consists of a facility and the clients have not connected to it. To meet these conditions, we consider *two types* of facilities for each site i: the closed facilities with cost f_i and the already opened ones with no cost. Further, w.r.t. the closed facilities at site i, we construct a set of clients to be chosen by $C1_i = \mathcal{U}$. Similarly for the previously opened facilities at site i, the target clients are put into a set $C2_i = \{j \in \mathcal{U} \,|\, x_{ij} < y_i\}$. Initially, $\forall i \in \mathcal{F} : C1_i = \mathcal{C}$, $C2_i = \emptyset$. By the above constructions, at each site i, the star to be chosen is either $\{(i, C') \,|\, i \in \mathcal{F}, C' \subseteq C1_i\}$ with facility cost f_i or $\{(i, C') \,|\, i \in \mathcal{F}, C' \subseteq C2_i\}$

with facility cost 0. In addition, since each client j has r_j demands, we can treat these demands as virtual ports in j and assign each port to a different facility. Without loss of generality (W.l.o.g.), we number the ports of j from 1 to r_j and connect them in ascending order. Also, we denote the port q of j as $j^{(q)}$ where $1 \leq q \leq r_j$, and $\phi\left(j^{(q)}\right)$ as the site which a port $j^{(q)}$ connects to. In every iteration of the algorithm, we use variable p_j to keep track of the port of client j to be connected. Initially $\forall j \in \mathcal{C} : p_j = 1$, and obviously $\mathcal{U} = \{j \in \mathcal{C} \,|\, p_j \leq r_j\}$. The *greedy objective* for picking the most cost-effective star (i, C') in the algorithm is defined as the minimum of $\min_{i \in \mathcal{F},\, C' \subseteq C2_i} \frac{\sum_{j \in C'} c_{ij}}{|C'|}$ and $\min_{i \in \mathcal{F},\, C' \subseteq C1_i} \frac{f_i + \sum_{j \in C'} c_{ij}}{|C'|}$.

Algorithm 4.1 SG-1: Star-Greedy Algorithm with Ratio 1.861

Input: $\mathcal{F}$, $\mathcal{C}$, $\boldsymbol{f}$, $\boldsymbol{c}$, $\boldsymbol{r}$, $\boldsymbol{R}$. **Output:** $(\boldsymbol{x}, \boldsymbol{y})$.
Initialization: Set $\mathcal{U} = \mathcal{C}$, $\forall i \in \mathcal{F},\, j \in \mathcal{C} : x_{ij},\, y_i = 0,\, p_j = 1$.

$\forall i \in \mathcal{F}$: sort the associated connection costs ascendingly and then construct the ordered sets $C1_i$ and $C2_i$.
While $\mathcal{U} \neq \emptyset$:

1. Choose the optimal star (i, C') with the minimum of: $\left(\min_{i \in \mathcal{F},\, C' \subseteq C2_i} \frac{\sum_{j \in C'} c_{ij}}{|C'|},\, \min_{i \in \mathcal{F},\, C' \subseteq C1_i} \frac{f_i + \sum_{j \in C'} c_{ij}}{|C'|}\right)$.
2. If $\exists j \in C'$: $x_{ij} = y_i$, then set $y_i \leftarrow y_i + 1$.
3. $\forall j \in C'$: set $\phi\left(j^{(p_j)}\right) \leftarrow i$ and $x_{ij} \leftarrow x_{ij} + 1$.
4. $\forall j \in C'$ s.t. $p_j = r_j$: set $\mathcal{U} \leftarrow \mathcal{U} \backslash \{j\}$, otherwise set $p_j \leftarrow p_j + 1$.
5. Update $C1_i$ and $C2_i$.

Analysis The primal and dual LPs for $FTRA_\infty$ are the following.

$$\begin{array}{lll} \text{minimize} & \sum_{i \in \mathcal{F}} f_i y_i + \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} c_{ij} x_{ij} & \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i \in \mathcal{F}} x_{ij} \geq r_j & \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 & \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \geq 0 & \\ & \forall i \in \mathcal{F} : y_i \geq 0 & \end{array} \tag{4.2}$$

$$\begin{array}{lll} \text{maximize} & \sum_{j \in \mathcal{C}} r_j \alpha_j & \\ \text{subject to} & \forall i \in \mathcal{F} : \sum_{j \in \mathcal{C}} \beta_{ij} \leq f_i & \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j - \beta_{ij} \leq c_{ij} & \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \beta_{ij} \geq 0 & \\ & \forall j \in \mathcal{C} : \alpha_j \geq 0 & \end{array} \tag{4.3}$$

Let $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ and $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ be the optimal fractional primal and dual solutions of the above LPs. The primal complementary slackness conditions

(CSCs) are: if $x^*_{ij} > 0$ then $\alpha^*_j - \beta^*_{ij} = c_{ij}$; if $y^*_i > 0$ then $\sum_{j \in \mathcal{C}} \beta^*_{ij} = f_i$. Dual CSCs are: if $\alpha^*_j > 0$ then $\sum_{i \in \mathcal{F}} x^*_{ij} = r_j$; if $\beta^*_{ij} > 0$ then $x^*_{ij} = y^*_i$.

Algorithm 4.2 PD-1: Primal-Dual Algorithm with Ratio 1.861

Input: $\mathcal{F}$, $\mathcal{C}$, $\boldsymbol{f}$, $\boldsymbol{c}$, $\boldsymbol{r}$, $\boldsymbol{R}$. **Output:** $(\boldsymbol{x}, \boldsymbol{y})$.
Initialization: Set $\mathcal{U} = \mathcal{C}$, $\forall i \in \mathcal{F}$, $j \in \mathcal{C}$: x_{ij}, $y_i = 0$, $p_j = 1$.

while $\mathcal{U} \neq \emptyset$, increase time t uniformly and execute the events below:

- Event 1: $\exists i \in \mathcal{F}$, $j \in \mathcal{U}$: $t = c_{ij}$ and $x_{ij} < y_i$.
  Action 1: set $\phi\left(j^{(p_j)}\right) \leftarrow i$, $x_{ij} \leftarrow x_{ij} + 1$ and $\alpha_j^{p_j} \leftarrow t$; If $p_j = r_j$, then $\mathcal{U} \leftarrow \mathcal{U} \backslash \{j\}$, otherwise $p_j \leftarrow p_j + 1$.

- Event 2: $\exists i \in \mathcal{F}$: $\sum_{j \in \mathcal{U}} \max(0, t - c_{ij}) = f_i$.
  Action 2: set $y_i \leftarrow y_i + 1$; $\forall j \in \mathcal{U}$ s.t. $t \geq c_{ij}$: do Action 1.

Remark 4.2.1. If more than one event happen at time t, the algorithm processes all of them in an arbitrary order. Also, the events themselves may repeatedly happen at any t since an unconstrained number of facilities are allowed to open at every site.

We restate Algorithm 4.1 as an equivalent primal-dual algorithm (Algorithm 4.2) for the sake of ratio analysis. The algorithm incrementally builds a primal solution consists of y_i's and x_{ij}'s in LP (4.2) in response to the changes of some dual variables. For a client j, the associated dual variables with its r_j ports are $\alpha_j^1, \ldots, \alpha_j^{r_j}$. For the primal variables y_i's and x_{ij}'s, they are initially set to be zero at the start of the algorithm. The variables will stop being updated and the algorithm will terminate once all clients j's are fully-connected with r_j facilities, i.e., the not-fully-connected client set $\mathcal{U} = \left\{ j \in \mathcal{C} \mid \sum_{i \in \mathcal{F}} x_{ij} < r_j \right\}$ (another definition) is empty. Note that in the primal-dual algorithm and its ratio analysis, we do not distinguish the facilities at a site for the reason that they are identical within a site, and this will not affect the solution and the analysis of the algorithm. The primal-dual algorithm is associated with a global time t that increases monotonically from 0. The value of α_j^q is assigned to the time when j's q-th port connects to $\phi(j^q)$. At any time t, we define the *payment* of a client $j \in \mathcal{U}$ to a site $i \in \mathcal{F}$ as t and the *contribution* as $t - c_{ij}$. As t increases, the algorithm runs in an event-driven fashion for constructing the primal solution. More precisely, we let the action that a client j connects (in port order) to a facility of i (x_{ij}'s value is increased by one) happens under two events: 1. j fully pays the connection cost c_{ij} with respect to an already opened facility at i which j is not connected

with (implying at this time $y_i > x_{ij}$); 2. the sum of the non-negative contributions of the clients in $\mathcal{U}$ to a closed facility at i fully pays its opening cost f_i and $t - c_{ij} \geq 0$. Moreover, Event 2 triggers the action that a new facility at i is opened (y_i's value is increased by one), so the clients with non-negative contributions can then connect to i. In the ratio analysis, we will associate the dual solutions α_j's and β_{ij}'s in LP (4.3) with the dual values $\alpha_j^{r_j}$'s and the non-negative contributions of the clients.

Lemma 4.2.2. *SG-1 and PD-1 are equivalent, that is, they produce the same feasible primal solutions.*

Proof. In the algorithm PD-1, after rearranging the conditions of the events for two expressions of t, it is easy to see that the expressions match the greedy objective expressions in the algorithm SG-1. Furthermore, the time t in PD-1 keeps increasing, so the smallest t is always reached to execute the events. This is equivalent to pick the star with the minimum greedy objective value. The feasibility of the solution is quite clear since both algorithms terminate once the clients are fully-connected. □

Lemma 4.2.3. *SG-1 runs in time $O(n^3 \max_j r_j)$ and PD-1 runs in time $O(n^2 \max_j r_j)$.*

Proof. Both algorithms terminate in at most $\sum_j r_j$ iterations since each iteration at least connects a client. In SG-1, sorting and set construction take time $O(n_f n_c \log n_c)$. In each iteration, step 1 spends most of the time. It takes $O(n_f n_c)$ to choose the best star because the sets $C1_i$ and $C2_i$ are sorted. The overall time is therefore $O\left(n_f n_c \sum_j r_j\right)$ or $O(n^3 \max_j r_j)$. In PD-1, we can maintain two heap data structures to sort the times of both Event 1 and Event 2. By doing this, it takes less time to find the smallest t and the runtime can be improved to $O(n^2 \max_j r_j)$. We leave the details of implementing such kind of primal-dual algorithm to Chapter 5 to avoid redundancy. □

This following ratio analysis is for the equivalent primal-dual algorithm PD-1. The algorithm actually produces a feasible primal solution but seemingly an infeasible dual solution if we let $\alpha_j = \alpha_j^{r_j}$ and $\beta_{ij} = \max(0, \alpha_j - c_{ij})$. This is because although the second constraint of LP (4.3) holds, the first constraint fails to hold since the algorithm only ensures at any time t, $\forall i \in \mathcal{F} : \sum_{j \in \mathcal{U}} \max(0, t - c_{ij}) \leq f_i$, but not $\forall i \in \mathcal{F} : \sum_{j \in \mathcal{C}} \max(0, \alpha_j - c_{ij}) \leq f_i$. However, if we consider to relax the feasibilities of the problem's primal CSCs and prove our algorithm actually ensures these relaxed primal CSCs, we can

then step by step find the algorithm's approximation factor. First, we adopt the bifactor definition (Definition 3.1.8) in Chapter 3 and from the definition let $F_{SOL} = \sum_{i\in\mathcal{F}} f_i y_i$ and $C_{SOL} = \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x_{ij}$. So the total cost is $cost\,(\boldsymbol{x},\,\boldsymbol{y}) = F_{SOL}+C_{SOL}$ where $SOL = (\boldsymbol{x},\,\boldsymbol{y})$ is any feasible primal solution. Then, we consider the following feasibility relaxed primal CSCs (FR-Primal CSCs) with relaxation factors ρ_f and ρ_c respectively:

- FR-Primal CSCs: $\forall i,j$: if $x_{ij} > 0$ then $\alpha_j - \beta_{ij} \le \rho_c c_{ij}$; if $y_i > 0$ then $\sum_{j\in\mathcal{C}} \beta_{ij} \le \rho_f f_i$.

In the next lemma, we show that if the FR-Primal CSCs are satisfied by PD-1 with the *constructed dual* solution $(\boldsymbol{\alpha},\boldsymbol{\beta})$ such that (s.t.) $\forall j:\ \alpha_j = \alpha_j^{r_j}$ and $\forall i,j:\ \beta_{ij} = \max(0, \alpha_j - \rho_c c_{ij})$, the corresponding dual solution cost will be bounded by any feasible scaled primal solution cost.

Lemma 4.2.4. *If $(\boldsymbol{\alpha},\boldsymbol{\beta})$ satisfies the FR-Primal CSCs with factors ρ_f and ρ_c, and $SOL = (\boldsymbol{x},\,\boldsymbol{y})$ is any feasible solution, then $cost\,(\boldsymbol{\alpha}) \le \rho_f F_{SOL} + \rho_c C_{SOL}$.*

Proof. Since $(\boldsymbol{x},\,\boldsymbol{y})$ is any feasible solution, all constraints of LP (4.2) will hold first. Together with the assumption of the FR-Primal CSCs, we have:

$$
\begin{aligned}
&cost\,(\boldsymbol{\alpha}) = \sum_{j\in\mathcal{C}} r_j\alpha_j \le \sum_{j\in\mathcal{C}}\sum_{i\in\mathcal{F}} x_{ij}\alpha_j \\
&\le \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} \left[\beta_{ij}y_i + (\alpha_j - \beta_{ij})\,x_{ij}\right] \\
&\le \sum_{i\in\mathcal{F}} \rho_f f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} \rho_c c_{ij}x_{ij} \\
&= \rho_f F_{SOL} + \rho_c C_{SOL}
\end{aligned}
$$

□

Now the analysis steps left are: 1) prove the assumption that $\exists\rho_f,\rho_c$ s.t. our constructed dual solution $(\boldsymbol{\alpha},\boldsymbol{\beta})$ satisfies the corresponding FR-Primal CSCs; 2) establish a relationship between $cost\,(\boldsymbol{x},\,\boldsymbol{y})$ and $cost\,(\boldsymbol{\alpha})$, where $(\boldsymbol{x},\,\boldsymbol{y})$ here is a feasible primal solution produced from PD-1. For the above step 2) we have the following lemma:

Lemma 4.2.5. *$cost\,(\boldsymbol{x},\,\boldsymbol{y}) \le cost\,(\boldsymbol{\alpha})$, where $(\boldsymbol{x},\,\boldsymbol{y})$ is the feasible primal solution produced from the algorithm PD-1.*

Proof. From Event 1 and Event 2 of the algorithm, it is clear that the sum of dual values of all ports fully pays all facility and connection costs. Hence, we have $cost\,(\boldsymbol{x},\,\boldsymbol{y}) = \sum_{j\in\mathcal{C}}\sum_{1\le q\le r_j}\alpha_j^q \le \sum_{j\in\mathcal{C}}\sum_{1\le q\le r_j}\alpha_j^{r_j} = cost\,(\boldsymbol{\alpha})$. □

The previous two lemmas and the bifactor definition immediately imply the next lemma.

Lemma 4.2.6. *The algorithm PD-1 is $(\rho_f,\,\rho_c)$-approximation if its constructed dual solution $(\boldsymbol{\alpha},\boldsymbol{\beta})$ from the algorithm satisfies the FR-Primal CSCs with factors ρ_f and ρ_c.*

The next lemma and corollary are more specific forms of Lemma 4.2.6 by substituting the constructed dual solution $(\boldsymbol{\alpha},\boldsymbol{\beta})$ s.t. $\forall j:\ \alpha_j = \alpha_j^{r_j}$ and $\forall i,j:\ \beta_{ij} = \max\left(0,\alpha_j-\rho_c c_{ij}\right)$ into a more general FR-Primal CSCs, i.e., $\forall i:\ \sum_{j\in\mathcal{C}}\beta_{ij}\le\rho_f f_i$ and $\forall i,j:\ \alpha_j-\beta_{ij}\le\rho_c c_{ij}$. After the substitution, the second general CSC immediately holds, so the lemma and corollary below only consider to satisfy the first more general CSC.

Lemma 4.2.7. *The algorithm PD-1 is $(\rho_f,\,\rho_c)$-approximation if $\forall i\in\mathcal{F}:\ \sum_{j\in\mathcal{A}}\left(\alpha_j^{r_j}-\rho_c c_{ij}\right)\le\rho_f f_i$, where $\mathcal{A}=\left\{j\in\mathcal{C}\,|\,\alpha_j^{r_j}\ge\rho_c c_{ij}\right\}$.*

Corollary 4.2.8. *W.l.o.g., for every site i order the corresponding $k=|\mathcal{A}|$ clients in $\mathcal{A}=\left\{j\in\mathcal{C}\,|\,\alpha_j^{r_j}\ge\rho_c c_{ij}\right\}$ s.t. $\alpha_1^{r_1}\le\alpha_2^{r_2}\le\cdots\le\alpha_k^{r_k}$. Then the algorithm PD-1 is $(\rho_f,\,\rho_c)$-approximation if $\forall i\in\mathcal{F}:\ \sum_{j=1}^{k}\left(\alpha_j^{r_j}-\rho_c c_{ij}\right)\le\rho_f f_i$.*

We proceed the proof to find ρ_f and ρ_c that bound $\alpha_j^{r_j}$'s. At this stage we will use two necessary assumptions (especially for Lemma 4.2.10) which are the metric properties of $FTRA_\infty$ and the uniform connection requirements of the clients. In fact, with these assumptions the algorithm PD-1 satisfies some important properties captured by the following two lemmas. These properties essentially reveal $FTRA_\infty$'s unique combinatorial structure similar to both UFL and $FTFL$. Moreover, we apply the factor-revealing technique for UFL to eventually get the approximation factor.

Lemma 4.2.9. *For any site i and the corresponding k clients in $\mathcal{A}$, we have $\forall 1\le j\le k:\sum_{h=j}^{k}\max\left(0,\,\alpha_j^{r_j}-c_{ih}\right)\le f_i$.*

Proof. At any time $t=\alpha_j^{r_j}-\epsilon$ and $1\le j\le k$ (a moment before j-th client in $\mathcal{A}$ becomes fully-connected), according to the previous corollary and the PD-1 algorithm, all clients ordered from j to k are in set $\mathcal{U}$ (not fully-connected) and

they have the same dual value $\alpha_j^{r_j}$. The lemma then follows also because the PD-1 algorithm ensures at any time t, $\forall i \in \mathcal{F} : \sum_{j \in \mathcal{U}} \max(0, t - c_{ij}) \leq f_i$ and for any $1 \leq j \leq k$, $\sum_{h=j}^{k} \max\left(0, \alpha_j^{r_j} - c_{ih}\right) \leq \sum_{j \in \mathcal{U}} \max\left(0, \alpha_j^{r_j} - c_{ij}\right)$. □

Lemma 4.2.10. *For any site i and clients j, j' with $r_j = r_{j'} = r$, we have $\alpha_j^{r_j} \leq \alpha_{j'}^{r_{j'}} + c_{ij} + c_{ij'}$.*

Proof. If $\alpha_j^{r_j} \leq \alpha_{j'}^{r_{j'}}$, the lemma obviously holds. Now consider $\alpha_j^{r_j} > \alpha_{j'}^{r_{j'}}$, it implies j' makes its final connection earlier than j in our algorithm. At time $t = \alpha_j^{r_j} - \epsilon$, client j' has already fully-connected with $r_{j'}$ open facilities while j has not. Thus among these $r_{j'}$ facilities there is at least one that j has not connected to, because otherwise j will have $r_{j'} = r = r_j$ connected facilities which is a contradiction. Denote this facility by i', by the triangle inequality in the metric properties we have $c_{i'j} \leq c_{ij} + c_{ij'} + c_{i'j'}$. Also, from the PD-1 algorithm, at time t, i' is already open but not connected with j so $\alpha_j^{r_j} \leq c_{i'j}$; j' is connected to i' then $\alpha_{j'}^{r_{j'}} \geq c_{i'j'}$. The lemma follows. □

From Lemma 4.2.9 and 4.2.10, it is clear that $\alpha_j^{r_j}$, f_i, and c_{ij} here constitute a feasible solution to the factor revealing program (4) in [73] for UFL. Also from the Lemma 3.6 of [73], we can directly get $\forall i \in \mathcal{F}$ and $k \geq 1$: $\sum_{j=1}^{k} \left(\alpha_j^{r_j} - 1.861 c_{ij}\right) \leq 1.861 f_i$. This result together with Lemma 4.2.2, Lemma 4.2.3, and Corollary 4.2.8 lead to the following theorem.

Theorem 4.2.11. *For $FTRA_\infty$ with uniform $\boldsymbol{r}$, the algorithms SG-1 and PD-1 both achieve 1.861-approximation and run in time $O\left(n^3 \max_j r_j\right)$ and $O\left(n^2 \max_j r_j\right)$ respectively.*

We will revisit the algorithm PD-1 with runtime improvement in the next chapter when considering a more general and practical problem derived from $FTRA_\infty$.

4.3 A greedy algorithm with ratio 1.61

In this section, we present an improved star-greedy algorithm SG-2 and its equivalent primal-dual algorithm PD-2. SG-2 iteratively picks the most cost-effectiveness star and at the same time optimizes the overall connection cost. It terminates once all clients' connection requirements are satisfied. The optimization of the overall connection cost actually happens when a closed facility is opened. At this time, any client in $\mathcal{C} \backslash \mathcal{U}$ will switch its most expensive connection to this newly opened facility if this connection cost decreases. In SG-2 and

PD-2, we adopt the same set of notations as in SG-1 and PD-1. The greedy objective for picking the most cost-effective $(i,\, C')$ in SG-2 becomes the minimum of $\min_{i\in\mathcal{F},\, C'\subseteq C2_i} \frac{\sum_{j\in C'} c_{ij}}{|C'|}$ and $\min_{i\in\mathcal{F},\, C'\subseteq C1_i} \frac{f_i+\sum_{j\in C'} c_{ij}-\sum_{j\in\mathcal{C}\setminus\mathcal{U}} \max\left(0,\, \max_q c_{\phi\left(j^{(q)}\right)j}-c_{ij}\right)}{|C'|}$.

Algorithm 4.3 SG-2: Star-Greedy Algorithm with Ratio 1.61

Input: $\mathcal{F}$, $\mathcal{C}$, $\boldsymbol{f}$, $\boldsymbol{c}$, $\boldsymbol{r}$, $\boldsymbol{R}$. **Output:** $(\boldsymbol{x},\, \boldsymbol{y})$.
Initialization: Set $\mathcal{U}=\mathcal{C}$, $\forall i\in\mathcal{F},\, j\in\mathcal{C}:\, x_{ij},\, y_i=0,\, p_j=1$.

$\forall i\in\mathcal{F}$: sort the associated connection costs ascendingly and then construct the ordered sets $C1_i$ and $C2_i$.
While $\mathcal{U}\neq\emptyset$:

1. Choose the optimal star $(i,\, C')$ with the minimum of: $\left(\min_{i\in\mathcal{F},\, C'\subseteq C2_i} \frac{\sum_{j\in C'} c_{ij}}{|C'|},\, \min_{i\in\mathcal{F},\, C'\subseteq C1_i} \frac{f_i+\sum_{j\in C'} c_{ij}-\sum_{j\in\mathcal{C}\setminus\mathcal{U}} \max\left(0,\, \max_q c_{\phi\left(j^{(q)}\right)j}-c_{ij}\right)}{|C'|}\right)$.
2. If $\exists j\in C'$: $x_{ij}=y_i$, then set $y_i\leftarrow y_i+1$.
3. $\forall j\in C'$: set $\phi\left(j^{(p_j)}\right)\leftarrow i$ and $x_{ij}\leftarrow x_{ij}+1$; $\forall j\in\mathcal{C}\setminus\mathcal{U}$ s.t. $\max_q c_{\phi\left(j^{(q)}\right)j}-c_{ij}>0$: set $i_j^*\leftarrow \arg\max_{\phi\left(j^{(q)}\right)} c_{\phi\left(j^{(q)}\right)j}$, $x_{i_j^*j}\leftarrow x_{i_j^*j}-1$, $x_{ij}\leftarrow x_{ij}+1$ and $\phi\left(j^{\left(\arg\max_q c_{\phi\left(j^{(q)}\right)j}\right)}\right)\leftarrow i$.
4. $\forall j\in C'$ s.t. $p_j=r_j$: set $\mathcal{U}\leftarrow\mathcal{U}\setminus\{j\}$, otherwise set $p_j\leftarrow p_j+1$.
5. Update $C1_i$ and $C2_i$.

Analysis For ratio analysis, we again restate the algorithm SG-2 as an equivalent primal-dual algorithm PD-2. In PD-2, at any time t, the *payment* of any client j to a site i is defined as t, and the *contribution* is $\max(0,\, t-c_{ij})$ for the clients in $\mathcal{U}$ and $\max\left(0,\, \max_q c_{\phi\left(j^{(q)}\right)j}-c_{ij}\right)$ for the clients in $\mathcal{C}\setminus\mathcal{U}$. As t increases, the action that a client j connects to a facility of i (x_{ij} is increased by 1) also happens under two events. The first is the same as the Event 1 in the algorithm SG-1. The description of the second event is similar as well except that the clients' contributions are defined differently. In addition, Event 2 in PD-2 triggers the action that any client $j\in\mathcal{C}\setminus\mathcal{U}$ with $\max_q c_{\phi\left(j^{(q)}\right)j}-c_{ij}>0$ will switch one of its most expensive connections from $\arg\max_{\phi\left(j^{(q)}\right)} c_{\phi\left(j^{(q)}\right)j}$ to i.

Algorithm 4.4 PD-2: Primal-Dual Algorithm with Ratio 1.61

Input: $\forall i,\, j:\, f_i,\, c_{ij},\, r_j$.
Output: $\forall i,\, j:\, y_i,\, x_{ij}$.
Initialization: Set $\mathcal{U} = \mathcal{C}$, $\forall i,\, j:\, y_i = 0,\, x_{ij} = 0,\, p_j = 1$.

While $\mathcal{U} \neq \emptyset$, increase time t uniformly and execute the events below:

- Event 1: $\exists i \in \mathcal{F},\, j \in \mathcal{U}$: $t = c_{ij}$ and $x_{ij} < y_i$
  Action 1: set $\phi\left(j^{(p_j)}\right) \leftarrow i$, $x_{ij} \leftarrow x_{ij} + 1$ and $\alpha_j^{p_j} \leftarrow t$; If $p_j = r_j$, then set $\mathcal{U} \leftarrow \mathcal{U} \backslash \{j\}$, otherwise set $p_j \leftarrow p_j + 1$.

- Event 2: $\exists i \in \mathcal{F}$: $\sum_{j \in \mathcal{U}} \max(0,\, t - c_{ij}) +$
  $\sum_{j \in \mathcal{C} \backslash \mathcal{U}} \max\left(0,\, \max_q c_{\phi\left(j^{(q)}\right)j} - c_{ij}\right) = f_i$
  Action 2: set $y_i \leftarrow y_i + 1$; $\forall j \in \mathcal{C} \backslash \mathcal{U}$ s.t. $\max_q c_{\phi\left(j^{(q)}\right)j} - c_{ij} > 0$: set $i_j^* \leftarrow \arg\max_{\phi\left(j^{(q)}\right)} c_{\phi\left(j^{(q)}\right)j}$, $x_{i_j^* j} \leftarrow x_{i_j^* j} - 1$, $x_{ij} \leftarrow x_{ij} + 1$ and $\phi\left(j^{\left(\arg\max_q c_{\phi\left(j^{(q)}\right)j}\right)}\right) \leftarrow i$; $\forall j \in \mathcal{U}$ s.t. $t \geq c_{ij}$: do Action 1.

Remark 4.3.1. If more than one event happen at time t, the algorithm processes all of them in an arbitrary order. Also, the events themselves may repeatedly happen at any t since an unconstrained number of facilities are allowed to open at every site.

Lemma 4.3.2. *SG-2 and PD-2 are equivalent, that is, they produce the same feasible primal solutions.*

Proof. Similar to the proof for Lemma 4.2.2. □

Lemma 4.3.3. *Both SG-2 and PD-2 run in time* $O(n^3 \max_j r_j)$.

Proof. Both algorithms terminate in at most $\sum_j r_j$ iterations since each iteration at least connects a client. In SG-1, in each iteration, step 1, 3 and 4 can all take time $O(n_f n_c)$ by maintaining $\max_q c_{\phi\left(j^{(q)}\right)j}$ for all $j \in \mathcal{C} \backslash \mathcal{U}$. This is because finding $\max_q c_{\phi\left(j^{(q)}\right)j}$ will then take time $O(1)$ while updating $\max_q c_{\phi\left(j^{(q)}\right)j}$ takes time $O(n_f)$. The overall time of the algorithm is therefore $O\left(n_f n_c \sum_j r_j\right)$ or $O(n^3 \max_j r_j)$. In PD-2, clients' reconnections dominate the time complexity. Once they happen in Event 2, it takes time $O(n_c n_f)$ to update clients' contributions to other facilities for computing anticipated time of events. Therefore, the total time is $O(n^3 \max_j r_j)$. □

Before proceeding the analysis for PD-2, for simplicity we consider to decompose any solutions of ILP (4.1) into a collection of stars from set $\mathcal{S} = \{(i, \mathcal{C}') \mid i \in \mathcal{F}, \mathcal{C}' \subseteq \mathcal{C}\}$ and construct the equivalent ILP (4.4). Note that the

star considered here consists of a site and a set of clients. It is different from the definition in the greedy algorithm where a star includes two types of facilities. However, this will not make any difference because $C1_i$ and $C2_i$ can eventually combine into a star belonging to $\mathcal{S}$. Moreover, we are allowed to have duplicate stars in a solution. This directly implies multiple identical facilities can be opened at every site. The variable x_s in ILP (4.4) denotes the number of duplicate star s. Also, the cost of s denoted by c_s is equal to $f_s + \sum_{j \in s \cap \mathcal{C}} c_{sj}$. Here we use s to index the site in star s, therefore f_s is the facility cost of site s and c_{sj} is the connection cost between site s and client j.

$$\begin{aligned} \text{minimize} \quad & \sum_{s \in \mathcal{S}} c_s x_s \\ \text{subject to} \quad & \forall j \in \mathcal{C} : \sum_{s:j \in s} x_s \geq r_j \\ & \forall s \in \mathcal{S} : x_s \in \mathbb{Z}^+ \end{aligned} \tag{4.4}$$

Its LP-relaxation and dual LP are the following:

$$\begin{aligned} \text{minimize} \quad & \sum_{s \in \mathcal{S}} c_s x_s \\ \text{subject to} \quad & \forall j \in \mathcal{C} : \sum_{s:j \in s} x_s \geq r_j \\ & \forall s \in \mathcal{S} : x_s \geq 0 \end{aligned} \tag{4.5}$$

$$\begin{aligned} \text{maximize} \quad & \sum_{j \in \mathcal{C}} r_j \alpha_j \\ \text{subject to} \quad & \forall s \in \mathcal{S} : \sum_{j \in s \cap \mathcal{C}} \alpha_j \leq c_s \\ & \forall j \in \mathcal{C} : \alpha_j \geq 0. \end{aligned} \tag{4.6}$$

Single factor analysis We apply the dual fitting technique [73] for the PD-2's single factor analysis. In order to utilize the weak duality relationship between LP (4.6) and LP (4.5), we need an algorithm that produces feasible primal (x_s's) and dual (α_j's) solutions. Denote the objective values of LPs (4.4), (4.5) and (4.6) by SOL_{ILP}, SOL_{LP} and SOL_D respectively, such an algorithm establishes the relationship $SOL_D \leq SOL_{LP} \leq SOL_{ILP}$. Note that

$SOL_D \leq SOL_{LP}$ implies *any* feasible SOL_D is upper bounded by *all* feasible SOL_{LP}, then apparently after defining the optimal values of LPs (4.5) and (4.4) as OPT_{LP} and OPT_{ILP} respectively, we have $SOL_D \leq OPT_{LP} \leq OPT_{ILP}$. The algorithm PD-2 produces a feasible primal solution but an infeasible dual. This is because some stars may overpay c_s and therefore violate the constraint of LP (4.6). Nevertheless, if we shrink the dual by a factor ρ and prove the fitted dual $\frac{\alpha_j}{\rho}$ is feasible, we get $\frac{SOL_D}{\rho} \leq SOL_{LP} \leq SOL_{ILP}$. Therefore, if we denote SOL_P as the total cost of the primal solution produced by our algorithm, the *key* steps to obtain the approximation factor are: 1) establish a relationship between SOL_P and SOL_D from PD-2; 2) find a minimum ρ and prove the fitted dual $\frac{\alpha_j}{\rho}$ is feasible. For step 1), we have the following lemmas:

Lemma 4.3.4. *The total cost of the primal solution SOL_P produced by the algorithm PD-2 is $\sum_{j\in\mathcal{C}} \sum_{1\leq q\leq r_j} \alpha_j^q$.*

Proof. It is clear that in the algorithm, the sum of dual values of all ports fully pays all facility and connection costs even with reconnection of clients. Then the lemmas follows. □

Lemma 4.3.5. *Let the dual solution $\alpha_j^{r_j}$ returned by the algorithm PD-2 be a solution to LP (4.6), i.e. $\alpha_j = \alpha_j^{r_j}$, then the corresponding $SOL_D \geq SOL_P$.*

Proof. For a city j, $\alpha_j^{r_j}$ is the largest dual among its ports. Because we let $\alpha_j = \alpha_j^{r_j}$ in LP (4.6), $SOL_D = \sum_{j\in\mathcal{C}} r_j \alpha_j^{r_j} \geq \sum_{j\in\mathcal{C}} \sum_{1\leq q\leq r_j} \alpha_j^q = SOL_P$. □

For step 2), if we find a minimum ρ s.t. the fitted dual $\frac{\alpha_j}{\rho}$ ($\alpha_j = \alpha_j^{r_j}$, from now on we use α_j for simplicity) is feasible, we will then get $SOL_D \leq \rho \cdot OPT_{ILP}$. Together with the previous lemma, our algorithm is ρ-approximation. The following lemma and corollary are immediate.

Lemma 4.3.6. *Fitted dual $\frac{\alpha_j}{\rho}$ is feasible iff $\forall s \in \mathcal{S} : \sum_{j\in s\cap\mathcal{C}} \alpha_j \leq \rho \cdot c_s$.*

Corollary 4.3.7. *W.l.o.g., assume a star s consists of a site with opening cost f_s and k clients s.t. $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_k$. Denote the connection cost of j to the site as c_{sj}, then the fitted dual is feasible iff $\forall s \in \mathcal{S} : \sum_{j=1}^{k} \alpha_j \leq \rho \cdot \left(f_s + \sum_{j=1}^{k} c_{sj}\right)$, i.e. $\rho \geq \frac{\sum_{j=1}^{k} \alpha_j}{f_s + \sum_{j=1}^{k} c_{sj}}$.*

In order to find such a ρ, we again apply the factor revealing technique for a tight ratio. The following properties we are going to prove will guide the construction of a new series of factor-revealing programs.

Lemma 4.3.8. *At time $t = \alpha_j - \epsilon$, a moment before port $j^{(r_j)}$ first time gets connected (because $\alpha_j = \alpha_j^{r_j}$), $\forall 1 \leq h < j < k$, let $r_{h,j} = \max_i c_{ih}$ if port $h^{(r_h)}$ is already connected to a facility of a site, otherwise let $r_{h,j} = \alpha_h$ ($\alpha_h = \alpha_j$), then $r_{h,j} \geq r_{h,j+1}$.*

Proof. A client's ports always reconnect to a facility of a site with less connection cost, so its maximum connection cost will never increase. The lemma follows. □

Lemma 4.3.9. *For any star s with k clients, $\forall 1 \leq j \leq k : \sum_{h=1}^{j-1} \max(r_{h,j} - c_{sh}, 0)$ $+ \sum_{h=j}^{k} \max(\alpha_j - c_{sh}, 0) \leq f_s$.*

Proof. The lemma follows because at time $t = \alpha_j - \epsilon$, in the algorithm PD-2 the contribution of all clients (either connected or unconnected) in star s will not exceed the facility's opening cost at site s. □

Lemma 4.3.10. *For clients h, j in any star s with k clients s.t. $1 \leq h < j \leq k : r_h = r_j = r$, then $\alpha_j \leq r_{h,j} + c_{sh} + c_{sj}$.*

Proof. This is where we must enforce all clients have uniform $\boldsymbol{r}$. At time $t = \alpha_j - \epsilon$, if port $h^{(r_h)}$ is still not connected, by Lemma 4.3.8 $\alpha_j = r_{h,j}$ and this lemma holds. Otherwise, client h's ports have already connected to r different facilities (not necessary on different sites) and $r_{h,j} = \max_i c_{ih}$. At time t, since j has at most $r-1$ connections, there is at least a facility s.t. h connects to it but j does not. Denote this facility by i', by triangle inequality we have $c_{i'j} \leq c_{sj} + c_{sh} + c_{i'h}$. Also because i' is already open, then $\alpha_j \leq c_{i'j}$. The lemma holds from $r_{h,j} = \max_i c_{ih} \geq c_{i'h}$. □

Theorem 4.3.11. *Let $\rho = sup_{k \geq 1} \{\lambda_k\}$, i.e., the maximum value of λ_k among all k and*

$$
\begin{aligned}
\lambda_k = \textit{maximize} \quad & \frac{\sum_{j=1}^{k} \alpha_j}{f + \sum_{j=1}^{k} d_j} \\
\textit{subject to} \quad & \forall 1 \leq j < k : \alpha_j \leq \alpha_{j+1} \\
& \forall 1 \leq h < j < k : r_{h,j} \geq r_{h,j+1} \\
& \forall 1 \leq h < j \leq k : \alpha_j \leq r_{h,j} + d_h + d_j \qquad (4.7) \\
& 1 \leq j \leq k : \sum_{h=1}^{j-1} \max(r_{h,j} - d_h, 0) + \sum_{h=j}^{k} \max(\alpha_j - d_h, 0) \leq f \\
& 1 \leq h \leq j < k : \alpha_j, d_j, f, r_{h,j} \geq 0,
\end{aligned}
$$

then the fitted dual is feasible.

Proof. Let $f = f_s$, $d_j = c_{sj}$ together with α_j, $r_{h,j}$ constitute a feasible solution to the above program due to Lemma 4.3.8, 4.3.9, and 4.3.10. Hence $\forall s \in S$, $\dfrac{\sum_{j=1}^{k} \alpha_j}{f_s + \sum_{j=1}^{k} c_{sj}} \leq \lambda_k \leq \rho$ and the theorem follows from Corollary 4.3.7. □

Theorem 4.3.12. *For $FTRA_\infty$ with uniform $\boldsymbol{r}$, the algorithms SG-2 and PD-2 achieve 1.61-approximation in time $O(n^3 \max_j r_j)$.*

Proof. The previous theorem and the weak duality theorem imply when $\rho = sup_{k\geq 1}\{\lambda_k\}$, $SOL_D \leq \rho \cdot OPT_{ILP}$. Together with Lemma 4.3.5 and 4.3.3, it concludes that the algorithms are ρ-approximation in time $O(n^3 \max_j r_j)$. Also the factor-revealing program (4.7) we obtained is equivalent to program (25) of [73], then we can directly use the result therein to get $\forall k$, $\lambda_k \leq 1.61$ and hence $\rho = 1.61$. □

4.4 A hybrid greedy algorithm with ratio 1.52

Bi-factor analysis We apply the inverse dual fitting technique [145, 125] to the PD-2's bi-factor analysis. Inverse dual fitting considers scaled instances of the problem, and shows the duals of the original instances are feasible to the scaled instances. For $FTRA_\infty$, we scale any original instance $\mathcal{I}$'s facility cost by ρ_f and connection cost by ρ_c to get an instance $\mathcal{I}'$. In the original problem, let $SOL_{LP} = F_{SOL} + C_{SOL}$, where F_{SOL} and C_{SOL} represent the total facility cost and connection cost (they are possibly fractional) of any SOL_{LP} respectively. In the scaled problem, if we define the corresponding primal and dual costs as SOL'_{LP} and SOL'_D (with dual variable α'_j), then clearly $SOL'_{LP} = \rho_f \cdot F_{SOL} + \rho_c \cdot C_{SOL}$. Also, if $\alpha'_j = \alpha_j$ that is feasible to the scaled problem, by weak duality and Lemma 4.3.5, we have $SOL_P \leq SOL_D = SOL'_D \leq SOL'_{LP}$ and the following lemma and corollary.

Lemma 4.4.1. *The algorithm PD-2 is (ρ_f, ρ_c)-approximation iff $\forall s \in \mathcal{S}$: $\sum_{j\in s\cap\mathcal{C}} \alpha_j \leq \left(\rho_f \cdot f_s + \rho_c \sum_{j\in s\cap\mathcal{C}} c_{sj}\right)$.*

Corollary 4.4.2. *W.l.o.g., assume a star s consists of a site with opening cost f_s and k clients s.t. $\alpha_1 \leq \alpha_2 \leq \cdots \leq \alpha_k$. Denote the connection cost of j to the site as c_{sj}, then the algorithm PD-2 is (ρ_f, ρ_c)-approximation iff $\forall s \in \mathcal{S} : \sum_{j=1}^{k} \alpha_j \leq \left(\rho_f \cdot f_s + \rho_c \sum_{j=1}^{k} c_{sj}\right)$, i.e. $\rho_c \geq \frac{\sum_{j=1}^{k} \alpha_j - \rho_f \cdot f_s}{\sum_{j=1}^{k} c_{sj}}$.*

Similar to dual fitting, now we wish to find the minimum value of ρ_c for any $\rho_f \geq 1$. We can construct a new factor revealing program with an objective

function: $\lambda'_k = \text{maximize } \frac{\sum_{j=1}^{k} \alpha_j - \rho_f f}{\sum_{j=1}^{k} d_j}$ and having same constraints as program (4.7). If $\rho_c = sup_{k\geq 1}\{\lambda'_k\}$, we have $\forall s \in S$, $\frac{\sum_{j=1}^{k} \alpha_j - \rho_f f_s}{\sum_{j=1}^{k} c_{sj}} \leq \lambda'_k \leq \rho_c$, which implies a (ρ_f, ρ_c)-approximation from Corollary 4.4.2. Further, this program is equivalent to program (36) of [73]. Therefore from the result of [106] (see Lemma 2 therein), the algorithm PD-2 is bi-factor $(1.11, 1.78)$-approximation.

Finally, we can treat an $FTRA_\infty$ instance as an $FTFL$ instance and apply the cost scaling and greedy augmentation (CSGA) for $FTFL$ [67, 132]. CSGA has the same effect on $FTFL$ as on UFL (see Theorem 3.1.12 in Chapter 3). We set the scaling factor of the facility cost to be 1.504. Together with the greedy augmentation ensured by the following lemma, the factor of 1.52 can be achieved in time $O(n^3 \max_j r_j)$.

Lemma 4.4.3. *Greedy augmentation for $FTRA_\infty$ runs in time $O(n^3 \max_j r_j)$.*

Proof. The augmentation has to consider $n_f \max_j r_j$ facilities in total. In each iteration, it finds the facility with the largest gain in time $O(n_f n_c)$. Recall that the gain of a facility is the total decrease in the cost if the facility is open and some clients reconnects to the facility for less connection costs. In addition, reconnecting clients takes time $O(n_c)$. It also takes time $O(n_f n_c)$ to update $\forall j : \max_q c_{\phi(j^{(q)})j}$ after reconnection. Hence, the total time is $O(n^3 \max_j r_j)$. □

Theorem 4.4.4. *For $FTRA_\infty$ with uniform $\boldsymbol{r}$, the algorithms SG-2 and PD-2 with cost scaling and greedy augmentation achieve 1.52-approximation in time $O(n^3 \max_j r_j)$.*

4.5 A simple reduction to UFL

We first consider the following dual LP of $FTRA_\infty$ (left) and dual LP of UFL (right). The constraints of these LPs are the same. They only have different objective functions. Let OPT_{FTRA_∞} and OPT_{UFL} denote the fractional optimal costs to LP (4.8) and LP (4.9) respectively. By the strong duality theory, they are also the respective optimal fractional costs of the primal problems of $FTRA_\infty$ and UFL. From the objective functions and the identical constraints of the LPs, we can easily get $\min_j r_j OPT_{UFL} \leq OPT_{FTRA_\infty} \leq \max_j r_j OPT_{UFL}$.

$$\begin{array}{ll} \text{maximize} & \sum_{j\in\mathcal{C}} r_j\alpha_j \\ \text{subject to} & \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} \beta_{ij} \le f_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j - \beta_{ij} \le c_{ij} \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \beta_{ij} \ge 0 \\ & \forall j \in \mathcal{C} : \alpha_j \ge 0 \end{array} \tag{4.8}$$

$$\begin{array}{ll} \text{maximize} & \sum_{j\in\mathcal{C}} \alpha_j \\ \text{subject to} & \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} \beta_{ij} \le f_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j - \beta_{ij} \le c_{ij} \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \beta_{ij} \ge 0 \\ & \forall j \in \mathcal{C} : \alpha_j \ge 0 \end{array} \tag{4.9}$$

Guided by the above LPs, the reduction from the general non-uniform $FTRA_\infty$ to UFL can be easily carried out. First, suppose that for any instance of UFL, there is a ρ-approximation algorithm that produces a solution with cost $cost_{UFL}$. We can then augment the UFL instance and its solution to an $FTRA_\infty$ instance with a feasible solution as follows: at each open facility (location) of UFL, we build a site of $FTRA_\infty$ by opening $\max_j r_j$ copies of the facility; for a client j in UFL that connects to a facility i, we make r_j connections from the client j in $FTRA_\infty$ to the site i. Denote the cost of this constructed $FTRA_\infty$ solution as $cost_{FTRA_\infty}$. From the reduction, we have $cost_{FTRA_\infty} \le \max_j r_j cost_{UFL}$. From the assumption of the algorithm for UFL, we get $cost_{UFL} \le \rho OPT_{UFL}$. Collecting all the inequalities yields:

$$\begin{aligned} cost_{FTRA_\infty} &\le \max_j r_j cost_{UFL} \\ &\le \rho \max_j r_j OPT_{UFL} \\ &\le \rho \frac{\max_j r_j}{\min_j r_j} OPT_{FTRA_\infty}. \end{aligned}$$

This indicates an approximate solution to the general $FTRA_\infty$ with a factor of $\rho\frac{\max_j r_j}{\min_j r_j}$. The ratio seems trivial because the difference between $\max_j r_j$ and $\min_j r_j$ might be large. However, for the uniform $FTRA_\infty$ where $\max_j r_j = \min_j r_j$, the reduction turns out to be very effective since it implies that the uniform $FTRA_\infty$ can directly reduce to UFL with the same ratio ρ in polynomial time. Comparing to the previous more complicated star-greedy algorithms for $FTRA_\infty$, it is interesting that the simple reduction here achieves the similar solution quality (since ρ can also be 1.861, 1.61, and 1.52 for UFL).

4.6 Capacitated $FTRA_\infty$

We observe that there is a strong connection between the well studied *Soft Capacitated Facility Location* ($SCFL$) problem [129, 77, 106] and the capacitated $FTRA_\infty$ ($CFTRA_\infty$) problem considered here. In $SCFL$, a facility i is allowed to open multiple times with an identical cost f_i. This is similar to $CFTRA_\infty$ where a site has an unconstrained number of resources to allocate. We formulate the $CFTRA_\infty$ problem as ILP (4.10). The third constraint of the ILP limits the total requests a site is able to serve (the capacity of the site) when the capacity of every facility at site i is u_i.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \geq r_j \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} x_{ij} \leq u_i y_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \in \mathbb{Z}^+ \\ & \forall i \in \mathcal{F} : y_i \in \mathbb{Z}^+ \end{array} \tag{4.10}$$

Through investigating the work for $SCFL$ in [106], we discover that the similar results also hold for $CFTRA_\infty$.

Theorem 4.6.1. *Any (ρ_f, ρ_c)-approximation algorithm for $FTRA_\infty$ implies a $(\rho_f + \rho_c)$-algorithm for $CFTRA_\infty$.*

Proof. With the Lagrangian relaxation technique similar to [106] for $SCFL$, we can move the third constraint of $CFTRA_\infty$ into its objective function with a Lagrangian multiplier $\lambda = \frac{\rho_c}{\rho_f+\rho_c} \in [0, 1]$, and thereby constructing a new $FTRA_\infty$ problem as a relaxation of $CFTRA_\infty$. The new problem instance has facility opening cost $(1 - \lambda) f_i$ and connection cost $c_{ij} + \lambda\frac{f_i}{u_i}$. Afterwards, the facility opening cost of $FTRA_\infty$ is further scaled by $\frac{\rho_c}{\rho_f}$. The scaled instance is then solved by a (ρ_f, ρ_c)-approximation algorithm with output solution Y_i's and X_{ij}'s. In order to obtain a feasible solution to $CFTRA_\infty$, we construct the solution as $y_i = \max\left(\lceil\sum_{j\in\mathcal{C}} X_{ij}/u_i\rceil, Y_i\right)$ and $x_{ij} = X_{ij}$ which is feasible to ILP (4.10). For the analysis of this solution, we do not elaborate here since we can use a similar argument as the Theorem 5 in [106] for $SCFL$ with small modifications. This will eventually lead to $(\rho_f + \rho_c)$-approximation. □

The above theorem essentially says that any algorithm for $FTRA_\infty$ can be used as a 'black box' to solve $CFTRA_\infty$. Furthermore, the runtime of $FTRA_\infty$

is preserved for $CFTRA_\infty$. Therefore, the following theorems are immediate from our (1.861, 1.861)-approximation algorithm[2] PD-1 and (1.11, 1.78)-approximation algorithm PD-2 for $FTRA_\infty$ with uniform $\boldsymbol{r}$.

Theorem 4.6.2. *$CFTRA_\infty$ with uniform $\boldsymbol{r}$ can be approximated with a factor of 3.722 in time $O(n^2 \max_j r_j)$.*

Theorem 4.6.3. *$CFTRA_\infty$ with uniform $\boldsymbol{r}$ can be approximated with a factor of 2.89 in time $O(n^3 \max_j r_j)$.*

Notice that the runtimes of the algorithms for $CFTRA_\infty$ can also be turned into strongly polynomial without affecting the approximation ratios. This is because the runtimes of the algorithms for $FTRA_\infty$ can be made strongly polynomial, and the reduction steps from $CFTRA_\infty$ to $FTRA_\infty$ in Theorem 4.6.1 run in strongly polynomial time as well. We will revisit the algorithm PD-2 and the greedy augmentation procedure (both with runtime improvements) in Chapter 6 when considering a uniform constrained resource allocation problem.

4.7 Summary

In this chapter, we developed three greedy approximation algorithms for $FTRA_\infty$. The first two are star-greedy algorithms while the third is a hybrid greedy algorithm. These algorithms achieve factors of 1.861, 1.61, and 1.52 respectively for the uniform case of the problem, improving on the 1.861-approximation in [145]. Although the presented algorithms run in pseudo-polynomial time, as will be shown in the next chapters, their runtime can be turned into strongly polynomial without affecting the approximation ratios. We also showed a reduction from the general $FTRA_\infty$ to UFL. The reduction uses some generic LP properties and implies an approximation preserving reduction from the uniform $FTRA_\infty$ to UFL. For the uniform capacitated $FTRA_\infty$, we obtained factors of 2.89 and 3.722 via Lagrangian relaxation.

[2]From the factor revealing technique, PD-1 is actually a bifactor (1.861, 1.861) and a single factor 1.861 approximation algorithm.

Chapter 5

Reliable Resource Allocation

In this chapter, we initiate the study of the *Reliable Resource Allocation* (RRA) problem. In this problem, we are given a set of sites $\mathcal{F}$ each with an unconstrained number of facilities as resources. Every facility at site $i \in \mathcal{F}$ has an opening cost and a service reliability p_i. There is also a set of clients $\mathcal{C}$ to be allocated to facilities. Every client $j \in \mathcal{C}$ accesses a facility at i with a connection cost and reliability l_{ij}. In addition, every client j has a minimum reliability requirement (MRR) r_j for accessing facilities. The objective of the problem is to decide the number of facilities to open at each site and connect these facilities to clients such that all clients' MRRs are satisfied at a minimum total cost. The *Unconstrained Fault-Tolerant Resource Allocation* ($FTRA_\infty$) problem studied in Chapter 4 is a special case of RRA. In Section 5.2, we develop two equivalent primal-dual algorithms for RRA where the second is an acceleration of the first and runs in quasi-quadratic time. This also implies a faster primal-dual algorithm for $FTRA_\infty$. We provide ratio analysis for the primal-dual algorithm in Section 5.3. The analysis is based on the inverse dual fitting technique. We formalize this technique in Section 5.4 for analyzing the classical minimum set cover problem. In Section 5.5, we consider a special case of RRA where all r_j's and l_{ij}'s are uniform by reducing it to UFL. For a more general case where only l_{ij}'s are assumed to be uniform, we present its reduction to $FTRA_\infty$ in Section 5.6. The results of this chapter are based on the works in [95, 96].

5.1 Introduction

In recent decades, fault-tolerant system design has become a necessary requirement for providing reliable and robust network services. As a result, the

Fault-Tolerant Facility Location ($FTFL$) problem that generalizes UFL has attracted the attention of many researchers [76, 73, 67, 132, 31]. Although the facility location model $FTFL$ considers both cost optimization and fault-tolerance, it lacks the ability of capturing the 'grouped' property of resources at every site, and limits the connectivities between clients and sites. Consequently, we have considered the *Unconstrained Fault-Tolerant Resource Allocation* ($FTRA_\infty$) model for addressing these challenges. However, both $FTFL$ and $FTRA_\infty$ measure fault-tolerance only by specifying the number of connections each client makes. We observe this measurement is not sufficient in many applications like the very-large-scale integration (VLSI) design, and the resource allocation paradigm considered in this thesis. For instance, a client j in $FTRA_\infty$ may connect to r_j facilities that are all susceptible to failure (with very low reliability) and therefore j is still very likely to encounter faults. This observation motivates us to study an alternative model called *Reliable Resource Allocation* (RRA) which provides more solid fault tolerance. In particular, RRA assumes facilities and their connections with clients both possess estimated probabilities (between 0 and 1) of being reliable (with no fault). Also, the fault tolerance levels of the clients are ensured by their fractional minimum reliability requirement (MRR) values for accessing facilities. In this chapter, we also consider the metric RRA problem where the connection costs c_{ij}'s form a metric.

Problem formulation In the RRA problem, we are given a set of sites $\mathcal{F}$ and a set of clients $\mathcal{C}$, where $|\mathcal{F}| = n_f$ and $|\mathcal{C}| = n_c$. Let $n = n_f + n_c$, $m = n_f \times n_c$ for the convenience of runtime analysis. Every site $i \in \mathcal{F}$ has an unconstrained number of facilities with f_i as the cost and p_i ($0 \leq p_i \leq 1$) as the reliability. Every connection between a facility at site i and a client $j \in \mathcal{C}$ also has a cost c_{ij} and a reliability l_{ij} ($0 \leq l_{ij} \leq 1$). In addition, every client j has a fractional MRR that must be satisfied from accessing the facilities. For simplicity, we also denote it as r_j where $\forall j \in \mathcal{C} : r_j \geq 0$. The objective is to optimally allocate a certain number of facilities from each site to clients for satisfying their MRRs at a minimum total cost. The problem can be formulated as the following ILP in which y_i denotes the number of facilities to open at site i, and x_{ij} the number of connections between i and j.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} p_i l_{ij} x_{ij} \geq r_j \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \in \mathbb{Z}^+ \\ & \forall i \in \mathcal{F} : y_i \in \mathbb{Z}^+ \end{array} \tag{5.1}$$

Comparing to $FTRA_\infty$, the key difference in RRA is the introduction of p_i's, l_{ij}'s, and r_j's that are all fractional. These inputs appear together in the first constraint of the ILP to provide more robust fault tolerance for clients. In particular, it is reasonable to assume the reliabilities of the facilities and connections are independent and therefore a connection of a client j to a site i will provide j $p_i l_{ij}$ portion of its MRR r_j. Notice that the first linear constraint can also be replaced by a non-linear constraint $\forall j \in \mathcal{C} : 1-\prod_{x_{ij}=1}(1-p_i l_{ij}) \geq r_j$ where r_j is between 0 and 1. This non-linear RRA problem is much harder to approximate, unless when all p_i's and l_{ij}'s are uniform, it can be easily reduced to the $FTRA_\infty$ problem with the modified constraint $\forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \geq \left\lceil \frac{\log(1-r_j)}{\log(1-pl)} \right\rceil$. For each $i \in \mathcal{F}$, if $\forall j, j' \in \mathcal{C} : l_{ij} = l_{ij'}$, we say connection reliabilities are *partially uniform*. In addition, by imposing the constraints $\forall i \in \mathcal{F}$, $j \in \mathcal{C} : l_{ij} = 1$ or $\forall i \in \mathcal{F} : p_i = 1$, it introduces another two special cases of RRA that only considers either the facility reliability or connection reliability alone. If both of these constraints hold and $\forall j \in \mathcal{C} : r_j$ is a positive integer, RRA becomes $FTRA_\infty$. Figure 5.1 illustrates an example of the abstract system model of RRA consisting of 3 sites with facilities as resources and 4 clients. In the figure, the solution found for (x_{ij}, y_i) is: $x_{11} = 2$, $x_{12} = 2$, $x_{32} = 1$, $x_{33} = 3$, $x_{34} = 4$, $y_1 = 2$, $y_3 = 4$ and other variables remain to be 0. This figure may help understanding of the following presented algorithms and their analyses, so readers could refer back to it when necessary.

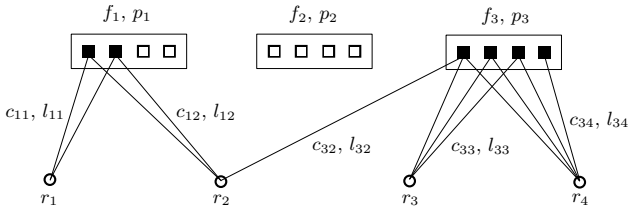

Figure 5.1: An example of the RRA model

Techniques and results We initiate the study of the RRA problem towards the provision of more robust fault-tolerance in the resource allocation paradigm. To the best of our knowledge, this is the first theoretical work that

takes into account the *quality of service* (QoS) requirement for resource allocation. Further, our presented ideas have potential to influence solutions to some other relevant problems. For the general RRA, we present two equivalent *primal-dual algorithms* built on the PD-1 algorithm for $FTRA_\infty$ in Chapter 4. In particular, the second algorithm is a significant improvement of the first in runtime that is quasi-quadratic. Since $FTRA_\infty$ is a special case of RRA, this also implies a fast strongly polynomial time algorithm for $FTRA_\infty$. In fact, the runtime improvement presented here is more generally defined in Chapter 6 as an *acceleration heuristic* that leverages the combinatorial structure of $FTRA_\infty$. Besides, the heuristics therein for the constrained resource allocation problem are more complicated to analyze. For the approximation ratio analysis, although with the similar techniques, RRA is a harder problem than $FTRA_\infty$ and the main difficulty we overcome is to cope with the fractional reliabilities by novel *dual constructions*. We apply the *inverse dual fitting* (IDF) technique introduced in [145, 125] as the central idea for analyzing the algorithm. Our analysis further elaborates and generalizes this generic technique. It naturally yields a non-constant approximation factor for the problem. For the case where connection reliabilities are partially uniform, and every client is provided with the same MRR that is at least the highest reliability a single access to a facility is able to provide (a lower bound reliability threshold), we obtain constant approximation factors of $2 + 2\sqrt{2}$ and 3.722. From practical point of view, the reliability threshold ensures the clients' lowest fault tolerance level. We also provide discussions about the IDF technique and formalize this technique for analyzing the classical minimum set cover problem. Furthermore, we get even better ratios for two special cases of RRA through *novel reductions* to UFL and $FTRA_\infty$ respectively. The reductions demonstrate some useful and generic linear programming techniques.

5.2 Primal-dual algorithms

For the general RRA problem displayed in LP (5.1), its LP-relaxation and dual LP are shown below.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij} x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} p_i l_{ij} x_{ij} \geq r_j \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : y_i \geq 0 \end{array} \tag{5.2}$$

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_{j\in\mathcal{C}} r_j\alpha_j \\ \text{subject to} \quad & \forall i \in \mathcal{F} : \textstyle\sum_{j\in\mathcal{C}} \beta_{ij} \le f_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j p_i l_{ij} - \beta_{ij} \le c_{ij} \\ & \forall j \in \mathcal{C} : \alpha_j \ge 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \beta_{ij} \ge 0 \end{aligned} \tag{5.3}$$

Now we describe two primal-dual algorithms (Algorithm 5.1 and Algorithm 5.2) with events that increase primal solutions y_i's and x_{ij}'s in LP (5.1) at different rates, in response to the changes of some dual variables. Initially, they are all 0. Both of the algorithms terminate when all clients' MRRs are satisfied, i.e., the set $\mathcal{U} = \left\{ j \in \mathcal{C} \mid \sum_{i\in\mathcal{F}} p_i l_{ij} x_{ij} < r_j \right\}$ is empty. Without loss of generality, assuming in the solution a client j totally makes d_j connections in the order from 1 to d_j and each connection is associated with a *virtual port* of j denoted by j^{vp} ($1 \le vp \le d_j$). The algorithm then associates every client j with d_j dual variables $\alpha_j^1, \ldots, \alpha_j^{d_j}$. Denoted by $\phi\left(j^{vp}\right)$, the facility/site client j's vpth port connected with, we can interpret the termination condition of the algorithm as $\forall j \in \mathcal{C} : \sum_{vp=1}^{d_j} p_{\phi(j^{vp})} l_{\phi(j^{vp})j} \ge r_j$. Note that although unlike $FTRA_\infty$, the required number of connections d_j is not pre-known for each client j in RRA, the above settings are necessary since they establishes a relationship between the fractional r_j's and the integral d_j's for the algorithm's analysis. In addition, a virtual port can only establish one connection with a facility of any site. Throughout the algorithm and analysis, we do not distinguish facilities within a site individually (like the function ϕ we denote), for the reason that the facilities at a site are identical. This will not affect the solution and the analysis of the algorithm.

Similar to the primal-dual algorithms for $FTRA_\infty$, both primal-dual algorithms for RRA are associated with a *global time* t that increases monotonically from 0. In the algorithm PD-3, we use variable d_j to keep track of the ports of client j that connect in order, and the value of $\alpha_j^{d_j}$ is *assigned* the time at which j's port d_j establishes a connection to $\phi\left(j^{d_j}\right)$. At any t, the *payment* of a client $j \in \mathcal{U}$ to a site $i \in \mathcal{F}$ is defined as $p_i l_{ij} t$ and the *contribution* as $\max\left(0,\, p_i l_{ij} t - c_{ij}\right)$. As t increases, the *action* that j connects to a facility of i (solution x_{ij} is increased by one) happens under two *events*: 1) j fully pays the connection cost of an already opened facility at i that it is not connected to (implying at this time $y_i > x_{ij}$); 2) sum of the contributions of clients in $\mathcal{U}$ to a closed facility at i fully pays its opening cost f_i and $p_i l_{ij} t \ge c_{ij}$. Moreover, event 2) first triggers the action that a new facility at i is opened (solution y_i is increased by one), so clients can then connect to this facility at i.

Lemma 5.2.1. *The algorithm PD-3 computes a feasible primal solution to the RRA problem and its runtime complexity is* $O\left(n^2\lceil\frac{\max_j r_j}{\min_{i,j} p_i l_{ij}}\rceil\right)$.

Proof. The primal solution is feasible since the output of the algorithm obeys the constraints and variable domains of ILP (5.1). For runtime, we use two binary heaps (both sorted by time t) to store *anticipated times* of Event 1 and Event 2 respectively. For Event 1, t is computed as $\frac{c_{ij}}{p_i l_{ij}}$ according to the algorithm, whereas t is $\frac{f_i+\sum_{j\in\mathcal{U}_i} c_{ij}}{p_i \sum_{j\in\mathcal{U}_i} l_{ij}}$ for Event 2. Therefore, detecting the next event (with smallest t) to process from two heaps takes time $O(1)$ and updating the heaps takes $O(\log m)$ in each iteration. Similar to the JV [77] and MMSV [104] algorithms for UFL, it actually takes $O(n_f \log m)$ to process every Action 1-b occurred, $O(1)$ for Action 1-a and $O(n_c)$ for Action 2-a. In addition, Action 1-b is triggered totally n_c times to fulfill all MRRs, and Action 1-a and 2-a both at most $\sum_{j\in\mathcal{C}} d_j$ times. Since $\sum_{j\in\mathcal{C}} d_j \le n_c \max_{j\in\mathcal{C}} d_j \le n_c\lceil\frac{\max_j r_j}{\min_{i,j} p_i l_{ij}}\rceil$, the total time complexity is $O\left(n_c^2\lceil\frac{\max_j r_j}{\min_{i,j} p_i l_{ij}}\rceil\right)$. □

Notice that the analysis in the above lemma also tells how the PD-1 algorithm for $FTRA_\infty$ in Chapter 4 can be implemented in time $O(n^2 \max_j r_j)$.

Algorithm 5.1 PD-3: Primal-Dual Algorithm for *RRA*

Input: $\forall i, j: f_i, p_i, c_{ij}, l_{ij}, r_j$.
Output: $\forall i, j: y_i, x_{ij}$.
Initialization: Set $\mathcal{U}=\mathcal{C}$, $\forall i, j: d_j=1, y_i=0, x_{ij}=0$.

While $\mathcal{U}\neq\emptyset$, increase time t uniformly and execute the events below:

- Event 1: $\exists i\in\mathcal{F}, j\in\mathcal{U}$ s.t. $p_i l_{ij} t = c_{ij}$ and $x_{ij} < y_i$.
  Action 1-a: Set $x_{ij}=x_{ij}+1$, $\alpha_j^{d_j}=t$ and $\phi\left(j^{d_j}\right)=i$;
  Action 1-b: If $\sum_{i\in\mathcal{F}} p_i l_{ij} x_{ij} \ge r_j$ then set $\mathcal{U}=\mathcal{U}\backslash\{j\}$, else set $d_j=d_j+1$.

- Event 2: $\exists i\in\mathcal{F}$ s.t. $\sum_{j\in\mathcal{U}} \max(0, p_i l_{ij} t - c_{ij}) = f_i$.
  Action 2-a: Set $y_i=y_i+1$ and $\mathcal{U}_i=\{j\in\mathcal{U} \mid p_i l_{ij} t \ge c_{ij}\}$; $\forall j\in\mathcal{U}_i$: do Action 1-a;
  Action 2-b: $\forall j\in\mathcal{U}_i$: do Action 1-b.

Remark 5.2.2. For the convenience of runtime analysis, sequential actions of events are separated as above. If more than one event happen at the same time, the algorithm processes all of them in an arbitrary order. Also, the events themselves may repeatedly happen at any time t because an unconstrained number of facilities at a site are allowed to open.

Remark 5.2.3. The above algorithm is adapted from the PD-1 algorithm for $FTRA_\infty$. If we consider adapting the PD-2 algorithm for $FTRA_\infty$ that also considers optimizing clients' total connection costs, it may render a feasible solution to RRA infeasible due to the clients' reliability constraints.

Algorithm 5.2 APD-3: Accelerated Primal-Dual Algorithm for RRA

Input: $\forall i,\, j:\, f_i,\, p_i,\, c_{ij},\, l_{ij},\, r_j$.
Output: $\forall i,\, j:\, y_i,\, x_{ij}$.
Initialization: Set $\mathcal{U} = \mathcal{C}$, $\forall i,\, j:\, y_i = 0,\, x_{ij} = 0,\, FR_j = 0$.

While $\mathcal{U} \neq \emptyset$, increase time t uniformly and execute the events below:

- Event 1: $\exists i \in \mathcal{F},\, j \in \mathcal{U}$ s.t. $p_i l_{ij} t = c_{ij}$ and $x_{ij} < y_i$.
  Action 1-a: Set $ToC = \min\left(y_i - x_{ij},\, \lceil \frac{r_j - FR_j}{p_i l_{ij}} \rceil\right)$;
  Action 1-b: Set $x_{ij} = x_{ij} + ToC$ and $FR_j = FR_j + p_i l_{ij} \cdot ToC$;
  Action 1-c: If $FR_j \geq r_j$ then set $\mathcal{U} = \mathcal{U} \setminus \{j\}$.

- Event 2: $\exists i \in \mathcal{F}$ s.t. $\sum_{j \in \mathcal{U}} \max(0,\, p_i l_{ij} t - c_{ij}) = f_i$.
  Action 2-a: Set $\mathcal{U}_i = \{j \in \mathcal{U} \,|\, p_i l_{ij} t \geq c_{ij}\}$, $ToC = \min_{j \in \mathcal{U}_i} \lceil \frac{r_j - FR_j}{p_i l_{ij}} \rceil$ and $y_i = y_i + ToC$; $\forall j \in \mathcal{U}_i$: do Action 1-b;
  Action 2-b: $\forall j \in \mathcal{U}_i$: do Action 1-c.

Remark 5.2.4. For the convenience of runtime analysis, sequential actions of events are separated as above. If more than one event happen at the same time, process Event 2 first so that no repeated events are needed.

PD-3 runs in pseudo-polynomial time that depends on p_i's, l_{ij}'s and r_j's. However, through a more careful look at the algorithm, we are able to speed it up to strongly polynomial time. First, we can combine the *repeated events* happened at time t into a single event by growing the solution y_i's and x_{ij}'s at a faster rate, and thereby reducing the number of events to process. This is because similar to $FTRA_\infty$, RRA allows multiple connections between each client-site pair. Thus, once a facility of a site is opened and connected with a group of clients' ports, according to the algorithm, in the repeated events additional facilities at this site will subsequently open and connect with this group of clients' other ports until *at least one* of these clients fulfills its MRR. This is also because Event 2 requires $\exists i \in \mathcal{F}$ s.t. $\sum_{j \in \mathcal{U}} \max(0,\, p_i l_{ij} t - c_{ij}) = f_i$, and when set $\mathcal{U}$ changes the current event will stop repeating. Similarly, once a client's port starts to connect to an open facility at site i in Event 1, its other ports may get connected to i at the same time until either there are no remaining open facilities at i or the client's MRR is already satisfied. Formally

in the Accelerated Primal-Dual Algorithm (the algorithm APD-3), let FR_j denote the already fulfilled reliability of client j and ToC the total number of connections to make after combining repeated events. The incremental rate of the solution is then determined by the value of ToC defined in the algorithm. Furthermore, it is not necessary to waste computation time to explicitly record the variables $\alpha_j^{d_j}$'s and $\phi\left(j^{d_j}\right)$'s for the total $\sum_{j\in\mathcal{C}} d_j$ connections (as in Algorithm 5.1). This is because these variables can implicitly exist only for the algorithm's ratio analysis. By making these changes, the new algorithm equivalent to Algorithm 5.1 in fact runs in quasi-quadratic time.

Lemma 5.2.5. *The algorithm APD-3 computes a feasible primal solution to the general RRA and its runtime complexity is* $\tilde{O}\left(n^2\right)$.

Proof. The primal solution is feasible because the algorithm is identical to Algorithm 5.1 in terms of the solution y_i's and x_{ij}'s produced. The difference is that it combines multiple repeated events in order to reduce the total occurrences of the actions. Therefore for runtime, we are able to bound the total numbers of both Event 2 and Action 2-a to n_c rather than $\sum_{j\in\mathcal{C}} d_j$, since as mentioned before once a facility of a site is opened, it will trigger at least one client's MRR to be satisfied and there are n_c clients in total. In addition, there are at most $n_f n_c$ Event 1. This is because for any client j and site i only when $t = \frac{c_{ij}}{p_i l_{ij}}$, j gets connected to open facilities at site i, and there are n_f sites and n_c clients in total. Thus the numbers of both Action 1-a and 1-b are bounded by $n_f n_c$. Finally, same as the algorithm PD-3, it takes $O\left(1\right)$ for Action 1-a and 1-b, $O\left(n_c\right)$ for Action 2-a and $O\left(n_f \log m\right)$ to process each of total n_c Action 1-c, yielding a total time of $O\left(n^2 \log n\right)$. □

By setting $\forall i \in \mathcal{F}, j \in \mathcal{C} : p_i = 1, l_{ij} = 1$ and letting r_j's be positive integers in PD-3, the analyses in the above lemma and Lemma 5.2.1 together imply that the runtime of the PD-1 algorithm for $FTRA_\infty$ can be improved to $O\left(n^2 \log n\right)$.

5.3 The inverse dual fitting analysis

We generalize the inverse dual fitting (IDF) technique introduced in [145] and elaborated in [125] for the algorithm PD-3's analysis. We observe that this technique is more generic and powerful than the dual fitting technique in [73], especially for the multi-factor analysis of the optimization problems involving different types of costs. The IDF technique will be discussed and formalized

on the classical minimum set cover problem in Section 5.4. IDF considers the *scaled instance* of the problem and shows that dual solution of the original instance is feasible to the scaled instance. Also, it is clear that the original instance's primal solution is feasible to the scaled instance. As for the *RRA* problem, we can construct a new instance $\mathcal{I}'$ by scaling any original instance $\mathcal{I}$'s facility cost by ρ_f and connection cost by ρ_c ($\rho_f \geq 1$ and $\rho_c \geq 1$). The scaled problem will then have the following formulation.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} \rho_f f_i y'_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} \rho_c c_{ij} x'_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} p_i l_{ij} x'_{ij} \geq r_j \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y'_i - x'_{ij} \geq 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x'_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : y'_i \geq 0 \end{array} \tag{5.4}$$

$$\begin{array}{ll} \text{maximize} & \sum_{j\in\mathcal{C}} r_j \alpha'_j \\ \text{subject to} & \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} \beta'_{ij} \leq \rho_f f_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha'_j p_i l_{ij} - \beta'_{ij} \leq \rho_c c_{ij} \\ & \forall j \in \mathcal{C} : \alpha'_j \geq 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \beta'_{ij} \geq 0 \end{array} \tag{5.5}$$

Our *constant factor analysis* for *RRA* step by step yields from IDF with partially uniform l_{ij}'s and the reliability threshold conditions $\forall j \in \mathcal{C} : r_j = r$ and $\forall j \in \mathcal{C} : r_j \geq \max_{i,j} p_i l_{ij}$. In particular, Lemma 5.3.5 holds with the partially uniform l_{ij}'s and the first threshold condition (also called the uniform reliability condition). Moreover, without the second threshold condition, we can first obtain an approximation factor depending on r.

Following IDF, we denote the total solution costs of LPs (5.2), (5.3), (5.4) and (5.5) by SOL_{LP}, SOL_D, SOL'_{LP} and SOL'_D respectively. In the original problem, let $SOL_{LP} = F_{SOL} + C_{SOL}$, where F_{SOL} and C_{SOL} represent the total facility cost and connection cost (both are possibly fractional) of any solution SOL, then $SOL'_{LP} = \rho_f \cdot F_{SOL} + \rho_c \cdot C_{SOL}$. Also, we can get the corresponding $SOL'_D = SOL_D$ by letting $\alpha'_j = \alpha_j$. An interesting observation is that the variable α_j we used for SOL_D does not necessarily need to be feasible to LP (5.3). Instead, IDF conducts the feasibility proof of $\alpha'_j = \alpha_j$ to scaled LP (5.5). We further denote SOL_P as the total cost of the feasible primal solution (y_i, x_{ij}) returned by the algorithm and let SOL_D represent the total cost of its corresponding *constructed dual solution* (α_j, β_{ij}). We will see later how this dual is constructed. It is clear that (y_i, x_{ij}) is a feasible solution to both LPs (5.2) and (5.4). By the weak duality theorem established between LPs (5.4) and (5.5), and if the constructed solution (α_j, β_{ij}) from the algorithm

is feasible to LP (5.5) after letting $\alpha'_j = \alpha_j$ and $\beta'_{ij} = \beta_{ij}$, then we have $SOL_D = SOL'_D \leq SOL'_{LP} = \rho_f \cdot F_{SOL} + \rho_c \cdot C_{SOL}$. Further, if $SOL_P \leq SOL_D$ is true, then it implies the algorithm is (ρ_f, ρ_c)-approximation. The following lemma is therefore immediate.

Lemma 5.3.1. *The algorithm PD-3 is (ρ_f, ρ_c)-approximation if its constructed dual solution (α_j, β_{ij}) is feasible to LP (5.5) and the corresponding $SOL_D \geq SOL_P$.*

The steps left are to construct a feasible dual (α_j, β_{ij}) from our algorithm and show $SOL_D \geq SOL_P$. For the second step, without loss of generality we can construct dual solution as $\forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j = (1 + \frac{\max_{i,j} p_i l_{ij}}{r_j})\alpha_j^{d_j}$, $\beta_{ij} = \max(0, p_i l_{ij}\alpha_j - \rho_c c_{ij})$, where $\alpha_j^{d_j}$ is the dual value of j's last connected port d_j from the algorithm PD-3, and β_{ij} can be interpreted as the parameterized contribution of this port to i by ρ_c. For this dual construction, we have the following lemma.

Lemma 5.3.2. *The constructed dual solution (α_j, β_{ij}) and the primal solution (y_i, x_{ij}) output from the algorithm PD-3 satisfy the corresponding $SOL_D \geq SOL_P$.*

Proof. From the primal-dual algorithm, $SOL_P = \sum_{j \in \mathcal{C}} \sum_{1 \leq vp \leq d_j} p_{\phi(j^{vp})} l_{\phi(j^{vp})j} \alpha_j^{vp}$ because the sum of the dual values of ports fully pay client connection and facility opening costs in the algorithm. In order to bound SOL_P with SOL_D, we aim to establish a relationship between r_j's (fractional) and d_j's (integral). From the construction of (α_j, β_{ij}), we have $SOL_D = \sum_{j \in \mathcal{C}} (1 + \frac{\max_{i,j} p_i l_{ij}}{r_j}) \alpha_j^{d_j} r_j = \sum_{j \in \mathcal{C}} \alpha_j^{d_j} (r_j + \max_{i,j} p_i l_{ij})$. Next we use a *key observation* that although $\forall j \in \mathcal{C} : r_j \leq \sum_{1 \leq vp \leq d_j} p_{\phi(j^{vp})} l_{\phi(j^{vp})j}$, we have $r_j + \max_{i,j} p_i l_{ij} \geq \sum_{1 \leq vp \leq d_j} p_{\phi(j^{vp})} l_{\phi(j^{vp})j}$. This is because before client j makes the last connection, it is true that $r_j \geq \sum_{1 \leq vp \leq d_j - 1} p_{\phi(j^{vp})} l_{\phi(j^{vp})j}$ and $\sum_{1 \leq vp \leq d_j - 1} p_{\phi(j^{vp})} l_{\phi(j^{vp})j} + \max_i p_i l_{ij} \geq \sum_{1 \leq vp \leq d_j} p_{\phi(j^{vp})} l_{\phi(j^{vp})j}$. Hence, $SOL_D \geq \sum_{j \in \mathcal{C}} \sum_{1 \leq vp \leq d_j} p_{\phi(j^{vp})} l_{\phi(j^{vp})j} \alpha_j^{d_j} \geq SOL_P$ (since $\alpha_j^{d_j} \geq \alpha_j^{vp}$). □

Now the only step left is to show (α_j, β_{ij}) is indeed a feasible solution. For simplicity, let $\forall j \in \mathcal{C} : z_j = 1 + \frac{\max_{i,j} p_i l_{ij}}{r_j}$, so $\alpha_j = z_j \alpha_j^{d_j}$. The second constraint of LP (5.5) holds from $\alpha_j = z_j \alpha_j^{d_j}$ and $\beta_{ij} = \max(0, p_i l_{ij}\alpha_j - \rho_c c_{ij})$. The remaining is to show the first constraint also holds. Built upon Lemma 5.3.1, we have the following lemma and corollary.

Lemma 5.3.3. *The algorithm PD-3 is (ρ_f, ρ_c)-approximation if $\forall i \in \mathcal{F} : \sum_{j \in \mathcal{A}} \left(p_i l_{ij} z_j \alpha_j^{d_j} - \rho_c c_{ij} \right) \leq \rho_f f_i$, where $\mathcal{A} = \left\{ j \in \mathcal{C} \,|\, \alpha_j^{d_j} \geq \frac{\rho_c}{z_j} \cdot \frac{c_{ij}}{p_i l_{ij}} \right\}$.*

Corollary 5.3.4. *Without loss of generality, for every site i order the corresponding $k = |\mathcal{A}|$ clients in $\mathcal{A} = \left\{ j \in \mathcal{C} \,|\, \alpha_j^{d_j} \geq \frac{\rho_c}{z_j} \cdot \frac{c_{ij}}{p_i l_{ij}} \right\}$ s.t. $\alpha_1^{d_1} \leq \alpha_2^{d_2} \leq \cdots \leq \alpha_k^{d_k}$. Then the algorithm PD-3 is (ρ_f, ρ_c)-approximation if $\forall i \in \mathcal{F}$: $\sum_{j=1}^{k} \left(p_i l_{ij} z_j \alpha_j^{d_j} - \rho_c c_{ij} \right) \leq \rho_f f_i$.*

We proceed the proof to find ρ_f and ρ_c that bound all $\alpha_j^{d_j}$'s. The next lemma captures the metric property of the problem and Lemma 5.3.6 generates one pair of satisfying (ρ_f, ρ_c). Note that from now on, the analysis follows the conditions that $\forall i \in \mathcal{F}$, $j, j' \in \mathcal{C}$: $l_{ij} = l_{ij'}$ and $\forall j \in \mathcal{C}$: $r_j = r$. This is because as shown in the next lemma, the necessary metric property of the problem may not hold without the first condition, and the second condition is important for the metric property to hold with dual variables. Also, from the second condition we now have $\forall j \in \mathcal{C}$: $z_j = z = 1 + \frac{\max_{i,j} p_i l_{ij}}{r}$.

Lemma 5.3.5. *For any site i and clients j, j' with $l_{ij} = l_{ij'}$ and $r_j = r_{j'} = r$, we have $p_i l_{ij} \alpha_j^{d_j} \leq p_i l_{ij'} \alpha_{j'}^{d_{j'}} + c_{ij} + c_{ij'}$.*

Proof. For any site i, if $\alpha_j^{d_j} \leq \alpha_{j'}^{d_{j'}}$ the lemma holds since $p_i l_{ij} \alpha_j^{d_j} \leq p_i l_{ij'} \alpha_{j'}^{d_{j'}}$ when $l_{ij} = l_{ij'}$. Now consider $\alpha_j^{d_j} > \alpha_{j'}^{d_{j'}}$, it implies j' makes its final connection earlier than j in our algorithm. At time $t = \alpha_j^{d_j} - \epsilon$, client j' has already satisfied its MRR $r_{j'}$ through connections with $d_{j'}$ open facilities while j has not fulfilled r_j. Thus among these $d_{j'}$ facilities there is at least one that j has not connected to, because otherwise j will have $r_{j'} = r = r_j$ fulfilled reliability which is a contradiction. Denote this facility by i', by triangle inequality of the metric property we have $c_{i'j} \leq c_{ij} + c_{ij'} + c_{i'j'}$. Since i' is already open at time t, then $p_i l_{ij} \alpha_j^{d_j} \leq c_{i'j}$ from our algorithm; j' is connected to i', then $p_i l_{ij'} \alpha_{j'}^{d_{j'}} \geq c_{i'j'}$. The lemma follows. □

The next lemma and the subsequent bi-factor approximation ratio are naturally generated from the IDF analysis. On the other hand, they are difficult to establish using the traditional dual fitting technique, which further demonstrates the advantage of IDF.

Lemma 5.3.6. *For any site i with $s = |\mathcal{B}|$ clients s.t. $\mathcal{B} = \left\{ j \in \mathcal{C} \,|\, \alpha_j^{d_j} \geq x \cdot \frac{c_{ij}}{p_i l_{ij}} \right\}$, l_{ij}'s are partially uniform on i, $x > 0$ and $\alpha_1^{d_1} \leq \alpha_2^{d_2} \leq \cdots \leq \alpha_s^{d_s}$, then $\sum_{j=1}^{s} \left(p_i l_{ij} \alpha_j^{d_j} - \left(2 + \frac{1}{x}\right) c_{ij} \right) \leq \left(1 + \frac{1}{x}\right) f_i$.*

Proof. First, we claim $\forall i \in \mathcal{F}$: $\sum_{j=1}^{s} \max\left(0,\, p_i l_{ij} \alpha_1^{d_1} - c_{ij}\right) \leq f_i$. This is true because at time $t = \alpha_1^{d_1} - \epsilon$, all the clients in $\mathcal{B}$ are also in $\mathcal{U}$ which implies from

our algorithm their total contribution should not exceed any facility's opening cost. So we also have:

$$\forall i \in \mathcal{F} : \sum_{j=1}^{s} \left(p_i l_{ij} \alpha_1^{d_1} - c_{ij} \right) \leq f_i \tag{5.6}$$

Setting $j' = 1$ in the inequality of Lemma 5.3.5, also because for $j = 1$ in $\mathcal{B}$ $\alpha_1^{d_1} \geq \frac{x}{p_i l_{i1}} c_{i1}$ and l_{ij}'s are partially uniform, yields:

$$\forall i \in \mathcal{F},\, j \in \mathcal{B} : p_i l_{ij} \alpha_j^{d_j} \leq \left(1 + \frac{1}{x}\right) p_i l_{ij} \alpha_1^{d_1} + c_{ij} \tag{5.7}$$

Therefore, after combining inequalities (5.6) and (5.7), $\forall i \in \mathcal{F}$:

$$\begin{aligned}
&\sum_{j=1}^{s} p_i l_{ij} \alpha_j^{d_j} \leq \sum_{j=1}^{s} \left(1 + \frac{1}{x}\right) p_i l_{ij} \alpha_1^{d_1} + \sum_{j=1}^{s} c_{ij} \\
&= \left(1 + \frac{1}{x}\right) \sum_{j=1}^{s} \left(p_i l_{ij} \alpha_1^{d_1} - c_{ij} \right) + \left(2 + \frac{1}{x}\right) \sum_{j=1}^{s} c_{ij} \\
&\leq \left(1 + \frac{1}{x}\right) f_i + \left(2 + \frac{1}{x}\right) \sum_{j=1}^{s} c_{ij}
\end{aligned}$$

The lemma then follows. □

Relating this lemma to Corollary 5.3.4, if $\mathcal{B} \supseteq \mathcal{A}$ then it implies $(\rho_f,\, \rho_c)$-approximation where $\rho_f = z\left(1 + \frac{1}{x}\right)$ and $\rho_c = z\left(2 + \frac{1}{x}\right)$. Also, $\mathcal{B} \supseteq \mathcal{A}$ iff $x \leq \frac{\rho_c}{z} = 2 + \frac{1}{x}$ (because $\forall j \in \mathcal{C} : z_j = z$), then we have $0 < x \leq 1 + \sqrt{2}$. Therefore, when $x = 1 + \sqrt{2}$, the algorithm is $\left(\sqrt{2}z,\, z + \sqrt{2}z\right)$-approximation. Recall that $z = \frac{r + \max_{i,j} p_i l_{ij}}{r}$, so the approximation ratio for the general RRA is $\left(\frac{\sqrt{2}(r + \max_{i,j} p_i l_{ij})}{r},\, \frac{\left(1+\sqrt{2}\right)(r + \max_{i,j} p_i l_{ij})}{r} \right)$ or $\left(\sqrt{2} + \frac{\sqrt{2}}{r},\, 1 + \sqrt{2} + \frac{1+\sqrt{2}}{r} \right)$ because we know $\max_{i,j} p_i l_{ij} \leq 1$. Now, with the threshold condition $\forall j \in \mathcal{C} : r_j \geq \max_{i,j} p_i l_{ij}$, it yields a constant bi-factor $\left(2\sqrt{2},\, 2 + 2\sqrt{2}\right)$ or a single factor $2 + 2\sqrt{2}$.

However, the approximation ratio of the algorithm can achieve a tighter bound through the factor revealing technique in [73]. Consider the following lemma that captures the execution of PD-3 more precisely than the claim in Lemma 5.3.6.

Lemma 5.3.7. *For any site i and the corresponding k clients in $\mathcal{A}$, we have* $\forall 1 \leq j \leq k : \sum_{h=j}^{k} \max\left(0,\, p_i l_{ij} \alpha_j^{d_j} - c_{ih}\right) \leq f_i$.

Proof. At time $t = \alpha_j^{d_j} - \epsilon$, all clients ordered from j to k are in set $\mathcal{U}$ (not fulfilled) and they have the same dual value $\alpha_j^{d_j}$. The lemma then follows because at any time in the algorithm PD-3, the total contribution of all clients in $\mathcal{U}$ will not exceed the facility's opening cost at site i. □

Fix any facility i, for the corresponding $1 \leq j \leq k = |\mathcal{A}|$:let $v_j = p_i l_{ij} \alpha_j^{d_j}$, $f = f_i$, $w_j = c_{ij}$ in the previous lemma and Lemma 5.3.5. It is clear that v_j, c_{ij} and f_i here constitute a feasible solution to the factor revealing program series (4) in [73] as shown below:

$$\begin{aligned}
\lambda_k = \text{maximize} \quad & \frac{\sum_{j=1}^{k} v_j}{f + \sum_{j=1}^{k} w_j} \\
\text{subject to} \quad & \forall j \in \{1, \ldots, k-1\} : v_j \leq v_{j+1} \\
& \forall h, j \in \{1, \ldots, k\} : v_j \leq v_h + w_h + w_j \\
& \forall j \in \{1, \ldots, k\} : \sum_{h=j}^{k} \max(v_j - w_h, 0) \leq f \\
& \forall j \in \{1, \ldots, k\} : v_j, w_j, f \geq 0.
\end{aligned}$$

From Lemma 3.6 in [73], we directly get $\forall i \in \mathcal{F}$ and corresponding k: $\sum_{j=1}^{k} (v_j - 1.861 c_{ij}) \leq 1.861 f_i$, i.e. $\sum_{j=1}^{k} \left(p_i l_{ij} z \alpha_j^{d_j} - 1.861 z c_{ij} \right) \leq 1.861 z f_i$. This result together with Lemma 5.2.5, and Corollary 5.3.4 with the value of z immediately lead to the following theorem and corollary.

Theorem 5.3.8. *The algorithm APD-3 achieves* $1.861\left(1 + \frac{1}{r}\right)$*-approximation for RRA in time* $\tilde{O}(n^2)$ *when connection reliabilities are partially uniform and all clients are provided with the same MRR.*

Corollary 5.3.9. *The algorithm APD-3 achieves* 3.722*-approximation for RRA in time* $\tilde{O}(n^2)$ *when connection reliabilities are partially uniform and all clients are provided with the same MRR that is at least the highest reliability a single access to a facility is able to provide.*

5.4 Minimum set cover: formalizing IDF

A majority of optimization problems target to either minimize or maximize the (mostly linear) aggregation of various types of costs under several constraints. These costs are involved in a problem's input instance. However, the problem like shortest path only consider one type of cost – weights of edges, whereas the facility location problems normally have two costs – facility and connection

costs. To approximate the problems involving different costs, the concept of multi-factor analysis arose naturally for balancing these costs in a solution, and thereby yielding a tighter approximation ratio. Although the IDF technique may be seen as an extension of dual fitting, it actually occupies a greater advantage by tightly coupling with the generic multi-factor analysis. Moreover in the ratio analysis of the *RRA* problem, we have shown this technique is able to simplify the ratio analysis and work seamlessly with the factor revealing technique. Next, we shall briefly see how the primal-dual method in [138, 74] together with IDF yields a simple analysis for the fundamental set cover problem.

In the minimum set cover problem, we are given a universe $\mathcal{U}$ of n elements and a collection $\mathcal{S}$ containing $s_1, \ldots, s_k$ that are subsets of $\mathcal{U}$ with the corresponding non-negative costs $c_1, \ldots, c_k$. The union of these subsets is $\mathcal{U}$. The objective is to pick a minimum cost collection from $\mathcal{S}$ whose union is $\mathcal{U}$. The problem can be easily formulated as the following LP in which variable x_s denotes whether set $s \in \mathcal{S}$ is selected.

$$\begin{aligned} \text{minimize} \quad & \sum_{s \in \mathcal{S}} c_s x_s \\ \text{subject to} \quad & \forall j \in \mathcal{U} : \sum_{s : j \in s} x_s \geq 1 \\ & \forall s \in \mathcal{S} : \; x_s \in \{0, 1\} \end{aligned}$$

Its LP-relaxation and dual LP are:

$$\begin{aligned} \text{minimize} \quad & \sum_{s \in \mathcal{S}} c_s x_s \\ \text{subject to} \quad & \forall j \in \mathcal{U} : \sum_{s : j \in s} x_s \geq 1 \\ & \forall s \in \mathcal{S} : \; x_s \geq 0 \end{aligned}$$

$$\begin{aligned} \text{maximize} \quad & \sum_{j \in \mathcal{U}} \alpha_j \\ \text{subject to} \quad & \forall s \in \mathcal{S} : \sum_{j \in s \cap \mathcal{U}} \alpha_j \leq c_s \\ & \forall j \in \mathcal{U} : \; \alpha_j \geq 0. \end{aligned} \tag{5.8}$$

In the primal-dual algorithm, all of the uncovered elements j's simply raise their duals α_j's until the cost c_s of a set s in $\mathcal{S}$ is fully paid for. At this moment, s is selected (x_s is set to 1) and duals of j's in s are frozen and withdrawn from the sets other than s. The algorithm then iteratively repeat these steps until there are no uncovered elements left. At the end of algorithm, $\sum_{j\in\mathcal{U}} \alpha_j = \sum_{s\in\mathcal{S}} c_s x_s$.

In the analysis following IDF, we consider to scale the costs of all sets in $\mathcal{S}$ by a positive number ρ. Since the set cover problem has only one type of cost, IDF will only generate a single factor. Similar to the analysis for the *RRA* problem, if the solutions x_s's and α_j's produced here are feasible to the scaled problem, then we have $\sum_{j\in\mathcal{U}} \alpha_j \leq \sum_{s\in\mathcal{S}} \rho c_s x_s$ by the weak duality theorem. This implies the algorithm is ρ-approximation. It is clear that x_s's are feasible. The step left is to show LP (5.8)'s scaled constraint holds, i.e., $\forall s \in \mathcal{S} : \sum_{j\in s\cap\mathcal{U}} \alpha_j \leq \rho c_s$. Without loss of generality, we assume there are l_s elements in a set s and $\alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_{l_s}$. So now we need to show $\forall s \in \mathcal{S} : \sum_{j=1}^{l_s} \alpha_j \leq \rho c_s$. From the primal-dual algorithm, we can see that at time $t = \alpha_i - \epsilon$, $\forall s \in \mathcal{S}$, $1 \leq i \leq l_s : \sum_{j=i}^{l_s} \alpha_j \leq c_s$ and $\forall i \leq j \leq l_s : \alpha_j = \alpha_i$. These together imply $\alpha_i \leq \frac{1}{l_s-i+1} c_s$ and hence $\sum_{i=1}^{l_s} \alpha_i \leq \sum_{i=1}^{l_s} \frac{1}{l_s-i+1} c_s$. Therefore, $\rho = \max_{l_s} \sum_{i=1}^{l_s} \frac{1}{l_s-i+1} \leq \mathcal{H}_n$ (n-th harmonic number where $n = |\mathcal{U}|$) and the set cover is $\mathcal{H}_n$-approximation.

5.5 Reduction to UFL

In this section, we consider a special case of *RRA* where $\forall i \in \mathcal{F}$, $j \in \mathcal{C} : r_j = r$, $l_{ij} = l$, $r \in (0, +\infty)$, and $l \in (0, 1]$. We present some generic LP reduction techniques to solve the problem. Note that if $l = 0$ and/or $r = 0$, this problem becomes trivial. The problem's LP-relaxation is displayed as follows.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij} x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} p_i l x_{ij} \geq r \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : y_i \geq 0 \end{array} \tag{5.9}$$

From this LP, we can substitute $x'_{ij} = \frac{p_i l}{r} x_{ij}$ and $y'_i = \frac{p_i l}{r} y_i$, then it transforms into the following UFL's LP-relaxation, where $\forall i \in \mathcal{F}, j \in \mathcal{C} : f'_i = \frac{r}{p_i l} f_i$ and $c'_{ij} = \frac{r}{p_i l} c_{ij}$.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f'_i y'_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c'_{ij} x'_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x'_{ij} \geq 1 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y'_i - x'_{ij} \geq 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x'_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : y'_i \geq 0 \end{array} \tag{5.10}$$

If we denote the costs of the optimal solutions of LPs (5.9) and (5.10) as OPT_{RRA} and OPT_{UFL} respectively, then we have $OPT_{RRA} = OPT_{UFL}$. Suppose that there is a ρ-approximation algorithm that solves this UFL problem with integer solution $\left(\bar{x'_{ij}}, \bar{y'_i}\right)$ and total cost $S\bar{O}L' = \sum_{i\in\mathcal{F}} f'_i \bar{y'_i} + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c'_{ij}\bar{x'_{ij}}$, then $S\bar{O}L' \leq \rho OPT_{UFL} = \rho OPT_{RRA}$. For obtaining the solution to RRA, we can construct $(\bar{x_{ij}}, \bar{y_i})$ s.t. $\forall i \in \mathcal{F}, j \in \mathcal{C} : \bar{x_{ij}} = \left\lceil \frac{r}{p_i l} \right\rceil \bar{x'_{ij}}$ and $\bar{y_i} = \left\lceil \frac{r}{p_i l} \right\rceil \bar{y'_i}$ as a feasible integer solution to LP (5.1) where $\forall i \in \mathcal{F}, j \in \mathcal{C} : r_j = r, l_{ij} = l$. Its total cost is denoted as $S\bar{O}L$. Since $\left(\bar{x'_{ij}}, \bar{y'_i}\right)$ is feasible to LP (5.10), the feasibility of $(\bar{x_{ij}}, \bar{y_i})$ can be easily inferred by looking at the constraints of LPs (5.10) and (5.1). In addition, from the bound of $S\bar{O}L'$ we can derive a bound for $S\bar{O}L$ in the following.

$$\begin{aligned} & S\bar{O}L' = \sum_{i\in\mathcal{F}} f'_i \bar{y'_i} + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c'_{ij}\bar{x'_{ij}} \leq \rho OPT_{RRA} \Rightarrow \\ & \sum_{i\in\mathcal{F}} \frac{r}{p_i l \left\lceil \frac{r}{p_i l} \right\rceil} f_i \bar{y_i} + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} \frac{r}{p_i l \left\lceil \frac{r}{p_i l} \right\rceil} c_{ij}\bar{x_{ij}} \leq \rho OPT_{RRA} \Rightarrow \\ & \sum_{i\in\mathcal{F}} \frac{r}{p_i l \left(\frac{r}{p_i l} + 1\right)} f_i \bar{y_i} + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} \frac{r}{p_i l \left(\frac{r}{p_i l} + 1\right)} c_{ij}\bar{x_{ij}} \leq \rho OPT_{RRA} \Rightarrow \\ & \sum_{i\in\mathcal{F}} f_i \bar{y_i} + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}\bar{x_{ij}} \leq \left(\frac{r + \max_i p_i l}{r}\right) \rho OPT_{RRA} \Rightarrow \\ & S\bar{O}L \leq \left(1 + \frac{1}{r}\right) \rho OPT_{RRA} \ \left(\because \max_i p_i l \leq 1\right) \end{aligned}$$

From the derivation, it is easy to see the approximation ratio of our constructed solution $(\bar{x_{ij}}, \bar{y_i})$ for RRA is $\left(1 + \frac{1}{r}\right)\rho$. As r becomes larger, the actual ratio will get closer to the ratio ρ of UFL. For the choice of ρ, we can either seek for the best ratio of 1.488 [89] in weakly polynomial time, or 1.52 [108] in strongly polynomial time. Also, practically if r is above the threshold reliability that a single access to a facility is able to provide, i.e. $r \geq \max_i p_i l$, this directly implies approximation ratios of 2.976 in weakly polynomial time and 3.04 in strongly polynomial time for RRA. For completeness, we sketch the overall algorithm as:

1. For any input instance of RRA with input ($\mathcal{F}$, $\mathcal{C}$, f_i, c_{ij}, p_i, l, r), construct the UFL instance with corresponding $\left(\mathcal{F}, \mathcal{C}, f'_i, c'_{ij}\right)$.

2. Solve this UFL instance using a ρ-approximation algorithm either from [89] or [108]. It outputs an integer solution $\left(\bar{x'_{ij}}, \bar{y'_i}\right)$.

3. Construct $(\bar{x_{ij}}, \bar{y_i}) = \left(\left\lceil \frac{r}{p_i l} \right\rceil \bar{x'_{ij}}, \left\lceil \frac{r}{p_i l} \right\rceil \bar{y'_i}\right)$ as the integer solution for RRA, and the total cost produced is $\bar{SOL}$.

Note that the above algorithm can also be easily adapted to solve the case where l_{ij}'s are partially uniform. However, this algorithm requires uniform conditions (r and l) in the algorithm and therefore does not produce feasible solution to the general RRA problem. On the other hand, a primal-dual algorithm is more versatile which produces feasible solution to the general RRA problem and only requires uniform conditions in its analysis.

5.6 Reduction to $FTRA_\infty$

Finally, we consider a more general case of RRA where we only require $\forall i \in \mathcal{F}$, $j \in \mathcal{C} : l_{ij} = l$[1]. Its LP-relaxation is displayed in the following.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij} x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} p_i l x_{ij} \geq r_j \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : y_i \geq 0 \end{array} \tag{5.11}$$

By substituting $x'_{ij} = p_i l x_{ij}$ and $y'_i = p_i l y_i$, the above LP transforms into the following $FTRA_\infty$'s LP-relaxation, where $\forall i \in \mathcal{F}, j \in \mathcal{C} : f'_i = \frac{1}{p_i l} f_i$ and $c'_{ij} = \frac{1}{p_i l} c_{ij}$.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f'_i y'_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c'_{ij} x'_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x'_{ij} \geq r_j \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y'_i - x'_{ij} \geq 0 \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x'_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : y'_i \geq 0 \end{array} \tag{5.12}$$

However, the above LP is only a *fractional* LP for $FTRA_\infty$ since r_j in RRA is fractional instead of integral in $FTRA_\infty$. Consequently, we cannot directly reduce RRA to $FTRA_\infty$. We resolve this problem by rounding r_j to $\lceil r_j \rceil$ in

[1]Partially uniform l_{ij}'s can also suffice. For the convenience of presentation, we keep assuming all l_{ij}'s are uniform.

the LP to construct a valid $FTRA_\infty$ LP. Now we can look at the following dual LPs corresponding to the fractional LP and the valid LP respectively.

$$\begin{array}{ll} \text{maximize} & \sum_{j\in\mathcal{C}} r_j\alpha_j \\ \text{subject to} & \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} \beta_{ij} \le f_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j - \beta_{ij} \le c_{ij} \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \beta_{ij} \ge 0 \\ & \forall j \in \mathcal{C} : \alpha_j \ge 0 \end{array} \qquad \begin{array}{ll} \text{maximize} & \sum_{j\in\mathcal{C}} \lceil r_j \rceil\alpha_j \\ \text{subject to} & \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} \beta_{ij} \le f_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j - \beta_{ij} \le c_{ij} \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \beta_{ij} \ge 0 \\ & \forall j \in \mathcal{C} : \alpha_j \ge 0 \end{array}$$

Let OPT_{f-FTRA_∞} and OPT_{FTRA_∞} denote the fractional optimal costs of the above dual LPs respectively. By the strong duality theory, they are also the respective optimal fractional costs of the primal problems (in particular, OPT_{f-FTRA_∞} is the optimal cost of LP (5.12)). From the objective functions and the identical constraints of the LPs, we get:

$$\frac{OPT_{FTRA_\infty}}{OPT_{f-FTRA_\infty}} = \frac{\sum_{j\in\mathcal{C}} \lceil r_j \rceil\alpha_j}{\sum_{j\in\mathcal{C}} r_j\alpha_j} \le 1 + \frac{\sum_{j\in\mathcal{C}} \alpha_j}{\sum_{j\in\mathcal{C}} r_j\alpha_j} \le 1 + \frac{1}{\min_j r_j}.$$

Also, let OPT_{RRA} denote the optimal cost of LP (5.11), we then have $OPT_{RRA} = OPT_{f-FTRA_\infty}$. Suppose that there is a ρ-approximation algorithm that solves the $FTRA_\infty$ problem (with $\lceil r_j \rceil$ as the connection requirement) with integer solution $\left(\bar{x'_{ij}}, \bar{y'_i}\right)$ and total cost $S\bar{O}L' = \sum_{i\in\mathcal{F}} f'_i\bar{y'_i} + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c'_{ij}\bar{x'_{ij}}$. After collecting all the inequalities, we have:

$$S\bar{O}L' \le \rho OPT_{FTRA_\infty} \le \rho\left(1 + \tfrac{1}{\min_j r_j}\right) OPT_{f-FTRA_\infty} = \rho\left(1 + \tfrac{1}{\min_j r_j}\right) OPT_{RRA}.$$

For obtaining a solution to RRA, we can construct $(\bar{x_{ij}}, \bar{y_i})$ s.t. $\forall i \in \mathcal{F}, j \in \mathcal{C} : \bar{x_{ij}} = \left\lceil \frac{1}{p_i l} \right\rceil \bar{x'_{ij}}$ and $\bar{y_i} = \left\lceil \frac{1}{p_i l} \right\rceil \bar{y'_i}$ as a feasible integer solution to LP (5.1) where $\forall i \in \mathcal{F}$, $j \in \mathcal{C} : l_{ij} = l$. Its cost is denoted as $S\bar{O}L$. Since $\left(\bar{x'_{ij}}, \bar{y'_i}\right)$ is feasible to LP (5.12) (as $\lceil r_j \rceil \ge r_j$), the feasibility of $(\bar{x_{ij}}, \bar{y_i})$ yields from the constraints of LPs (5.12) and (5.1). In addition, from the bound of $S\bar{O}L'$ we can derive a bound for $S\bar{O}L$.

$$\bar{SOL'} = \sum_{i\in\mathcal{F}} f'_i \bar{y'_i} + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c'_{ij}\bar{x'_{ij}} \leq \rho\left(1+\frac{1}{\min_j r_j}\right) OPT_{RRA} \Rightarrow$$

$$\sum_{i\in\mathcal{F}} \frac{1}{p_i l\left\lceil\frac{1}{p_i l}\right\rceil} f_i \bar{y}_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} \frac{1}{p_i l\left\lceil\frac{1}{p_i l}\right\rceil} c_{ij}\bar{x_{ij}} \leq \rho\left(1+\frac{1}{\min_j r_j}\right) OPT_{RRA} \Rightarrow$$

$$\sum_{i\in\mathcal{F}} \frac{1}{p_i l\left(\frac{1}{p_i l}+1\right)} f_i \bar{y}_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} \frac{1}{p_i l\left(\frac{1}{p_i l}+1\right)} c_{ij}\bar{x_{ij}} \leq \rho\left(1+\frac{1}{\min_j r_j}\right) OPT_{RRA} \Rightarrow$$

$$\sum_{i\in\mathcal{F}} f_i \bar{y}_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}\bar{x_{ij}} \leq \left(1+\max_i p_i l\right)\left(1+\frac{1}{\min_j r_j}\right)\rho OPT_{RRA} \Rightarrow$$

$$\bar{SOL} \leq 2\rho\left(1+\frac{1}{\min_j r_j}\right) OPT_{RRA} \ \left(\because \max_i p_i l \leq 1\right)$$

It is clear that the approximation ratio of our constructed solution $(\bar{x_{ij}}, \bar{y_i})$ for RRA is $2\rho\left(1+\frac{1}{\min_j r_j}\right)$. As $\min_j r_j$ becomes larger, the actual ratio will get closer to two times of the ratio for $FTRA_\infty$. In fact, if $\min_j r_j \geq 1$ (which is very likely in practice), the ratio becomes 4ρ. For the choice of ρ, we can use the 1.575-approximation algorithm [150] for the general $FTRA_\infty$, and therefore the ratio could be 6.3 for RRA. For completeness, we sketch the overall algorithm as:

1. For any input instance of RRA with input $(\mathcal{F}, \mathcal{C}, f_i, c_{ij}, p_i, l, r_j)$, construct an $FTRA_\infty$ instance with the corresponding $\left(\mathcal{F}, \mathcal{C}, f'_i, c'_{ij}, \lceil r_j \rceil\right)$.

2. Solve this general $FTRA_\infty$ instance using a ρ-approximation algorithm from [150]. It outputs an integer solution $\left(\bar{x'_{ij}}, \bar{y'_i}\right)$.

3. Construct $(\bar{x_{ij}}, \bar{y_i}) = \left(\left\lceil\frac{1}{p_i l}\right\rceil \bar{x'_{ij}}, \left\lceil\frac{1}{p_i l}\right\rceil \bar{y'_i}\right)$ as an integer solution for RRA, and the total cost produced is $\bar{SOL}$.

5.7 Summary

In this chapter, we initiated the study of the RRA problem. For this problem, we developed two equivalent primal-dual algorithms where the second is an acceleration of the first and runs in quasi-quadratic time. For the approximation ratio analysis, we adopted the inverse dual fitting (IDF) technique as a central idea. The analysis naturally yields constant approximation factors of $2+2\sqrt{2}$ and 3.722. In addition, we formalized the IDF technique for analyzing the classical minimum set cover problem. We also considered a special case of RRA where all r_j's and l_{ij}'s are uniform by reducing it to UFL. The reduction

yields approximation ratios of 2.976 and 3.04. For a more general case where only l_{ij}'s are assumed to be uniform, we presented another novel reduction to $FTRA_\infty$ and obtained a factor of 6.3.

Chapter 6

Constrained Fault-Tolerant Resource Allocation

In this chapter, we introduce and study the *Constrained Fault-Tolerant Resource Allocation* ($FTRA$) problem. In $FTRA$, we are given a set of sites containing facilities as resources, and a set of clients accessing these resources. Specifically, each site i is allowed to open at most R_i facilities with cost f_i for each opened facility. Each client j requires an allocation of r_j open facilities and connecting j to any facility at site i incurs a connection cost c_{ij}. The goal is to minimize the total cost of this resource allocation scenario. $FTRA$ generalizes the *Unconstrained Fault-Tolerant Resource Allocation* ($FTRA_\infty$) [94] and the classical *Fault-Tolerant Facility Location* ($FTFL$) [76] problems: for every site i, $FTRA_\infty$ does not have the constraint R_i, whereas $FTFL$ sets $R_i = 1$. For the general metric $FTRA$, in Section 6.2 we first give an LP-rounding algorithm achieving an approximation ratio of 4. Then we show the problem reduces to $FTFL$ in Section 6.3. This implies a ratio of 1.7245. For the uniform $FTRA$ where all r_j's are the same, in Section 6.4 we provide a 1.52-approximation primal-dual algorithm in $O(n^4)$ time, where n is the total number of sites and clients. The algorithm directly yields another faster primal-dual algorithm for $FTRA_\infty$. In Section 6.5, we also consider the *Constrained Fault-Tolerant k-Resource Allocation* (k-$FTRA$) problem where additionally the total number of facilities can be opened across all sites is bounded by k. For the uniform k-$FTRA$, we give the first constant-factor approximation algorithm with a factor of 4. The results of this chapter are based on the works in [97, 98].

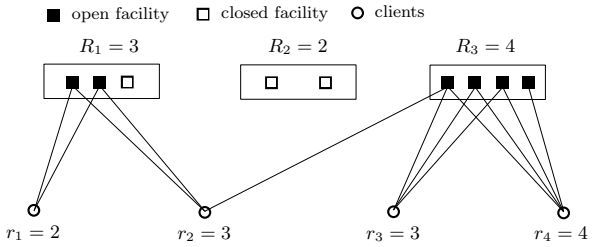

Figure 6.1: An $FTRA$ instance with a feasible solution

6.1 Introduction

In the *Constrained Fault-Tolerant Resource Allocation* ($FTRA$) problem, we are given a set $\mathcal{F}$ of sites and a set $\mathcal{C}$ of clients, where $|\mathcal{F}| = n_f$, $|\mathcal{C}| = n_c$ and $n = n_f + n_c$. Each site $i \in \mathcal{F}$ contains at most R_i ($R_i \geq 1$) facilities to open as resources and each client $j \in \mathcal{C}$ is required to be allocated r_j ($r_j \geq 1$) open facilities. Note that in $FTRA$, facilities at the same site are different and $\max_{j \in \mathcal{C}} r_j \leq \sum_{i \in \mathcal{F}} R_i$. Moreover, opening a facility at site i incurs a cost f_i and connecting j to any facility at i costs c_{ij}. The objective of the problem is to minimize the sum of facility opening and client connection costs under the resource constraint R_i. This problem is closely related to the *Unconstrained Fault-Tolerant Resource Allocation* ($FTRA_\infty$) considered in Chapter 4, and the classical *Fault-Tolerant Facility Location* ($FTFL$) [76] and *Uncapacitated Facility Location* (UFL) [129] problem. Both $FTRA_\infty$ and $FTFL$ are special cases of $FTRA$: R_i is unbounded in $FTRA_\infty$, whereas $\forall i \in \mathcal{F} : R_i = 1$ in $FTFL$. If $\forall j \in \mathcal{C} : r_j = 1$, they all reduce to UFL. Figure 6.1 displays an $FTRA$ instance with a feasible solution. We notice that both $FTRA$ and $FTRA_\infty$ have potential applications in numerous distributed systems such as cloud computing, content delivery networks, and Web services provisioning. The resource constraint R_i is common in these systems. The fault-tolerance attribute (r_j) can be also viewed as the parallel processing capability of these systems. Unless elsewhere specified, we consider the problems in metric space, that is, the connection costs c_{ij}'s satisfy the metric properties.

Problem formulation and properties The $FTRA$ problem has the following ILP formulation. From the ILP, we can verify that the problem becomes the special cases $FTFL$ if all R_i's are uniform and equal to 1, and $FTRA_\infty$ if the third resource constraint is removed.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \geq r_j \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : y_i \leq R_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij}, y_i \in \mathbb{Z}^+ \end{array} \tag{6.1}$$

The problem's LP-relaxation (primal LP) and dual LP are the following.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \geq r_j \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \forall i \in \mathcal{F} : y_i \leq R_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij}, y_i \geq 0 \end{array} \tag{6.2}$$

$$\begin{array}{ll} \text{maximize} & \sum_{j\in\mathcal{C}} r_j \alpha_j - \sum_{i\in\mathcal{F}} R_i z_i \\ \text{subject to} & \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} \beta_{ij} \leq f_i + z_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j - \beta_{ij} \leq c_{ij} \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j, \beta_{ij}, z_i \geq 0 \end{array} \tag{6.3}$$

Let $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ and $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*, \boldsymbol{z}^*)$ be the optimal fractional primal and dual solutions of the LPs, and $cost\,(\boldsymbol{x}, \boldsymbol{y})$ and $cost\,(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$ be the cost functions (objective value functions) of any primal and dual solutions respectively. By the strong duality theorem, $cost\,(\boldsymbol{x}^*, \boldsymbol{y}^*) = cost\,(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*, \boldsymbol{z}^*)$. Moreover, the primal complementary slackness conditions (CSCs) are:

(C1) If $x^*_{ij} > 0$ then $\alpha^*_j = \beta^*_{ij} + c_{ij}$.

(C2) If $y^*_i > 0$ then $\sum_{j\in\mathcal{C}} \beta^*_{ij} = f_i + z^*_i$.

Dual CSCs are:

(C3) If $\alpha^*_j > 0$ then $\sum_{i\in\mathcal{F}} x^*_{ij} = r_j$.

(C4) If $\beta^*_{ij} > 0$ then $x^*_{ij} = y^*_i$.

(C5) If $z^*_i > 0$ then $y^*_i = R_i$.

W.l.o.g., $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ and $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*, \boldsymbol{z}^*)$ have the following properties:

(P1) $\forall j \in \mathcal{C} : \alpha^*_j > 0$ and $\sum_{i\in\mathcal{F}} x^*_{ij} = r_j$.

(P2) $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is 'almost' complete, i.e. $\forall j \in \mathcal{C}$: if $x^*_{ij} > 0$ then $x^*_{ij} = y^*_i$ (the complete condition) or there is at most one i s.t. $0 < x^*_{ij} < y^*_i$ where i is the farthest site connecting j. (cf. [42, 132] for more details)

Techniques and results For the general $FTRA$, we first develop a *unified LP-rounding algorithm* through modifying and extending the 4-approximation LP-rounding algorithm [132] for $FTFL$. The algorithm can directly solve $FTRA$, $FTRA_\infty$, and $FTFL$ with the same approximation ratio of 4. This

is achieved by: 1) constructing some useful *properties* of the unified algorithm which enable us to directly round the optimal fractional solutions with values that might exceed one while ensuring the feasibility of the rounded solutions and the algorithm correctness; 2) exploiting the primal and dual complementary slackness conditions of the $FTRA$ problem's LP formulation. Then, we show $FTRA$ reduces to $FTFL$ using an *instance shrinking* technique inspired from the splitting idea of [149] for $FTRA_\infty$. This implies that these two problems may share the same approximability in weakly polynomial time. Hence, from the $FTFL$ result of [31], we obtain the ratio of 1.7245. For the non-metric $FTRA$, we get the first approximation factor of $O\left(\log^2 n\right)$ deduced from the work of [76, 100]. Note that, although our first rounding algorithm attains a worse approximation ratio, it could be more useful than the second to be adapted for other variants of the resource allocation problems. For the uniform $FTRA$, better results are obtained via the general primal-dual framework adopted in the previous two chapters. We first present a *primal-dual algorithm* that runs in pseudo-polynomial time. Although the algorithm is adapted from the PD-2 algorithm for $FTRA_\infty$ (see Chapter 4), it is much more difficult to analyze. We adopt a *constraint-based analysis* to derive the ratio of 1.61. Compared to dual fitting and inverse dual fitting techniques, our analysis approach is simpler and more convenient for handling more complicated dual constructions. Later, with a carefully designed *acceleration heuristic* applied to the primal-dual algorithm, we obtain a strongly polynomial time algorithm for $FTRA$ which has the same ratio of 1.61 but with a better runtime of $O\left(n^4\right)$. Moreover, by applying another similar heuristic to the greedy augmentation technique [67], the previous 1.61 ratio reduces to 1.52. Since $FTRA_\infty$ is a special case of $FTRA$, similar acceleration heuristics can be applied to the PD-2 algorithm and the greedy augmentation for $FTRA_\infty$. Also, the 3.16-approximation rounding result of [148] for the general $FTRA_\infty$ can be improved to 2.408. Lastly, we consider an important variant of $FTRA$ – the *Constrained Fault-Tolerant k-Resource Allocation* (k-$FTRA$) problem which adds an extra global constraint that at most k facilities across all sites can be opened as resources. For the uniform k-$FTRA$, based on the work of [77, 73, 132], we give the first constant-factor approximation algorithm for this problem with a factor of 4. In particular, the algorithm relies on a polynomial time *greedy pairing procedure* we develop for efficiently splitting sites into paired and unpaired facilities. The results described above directly hold for $FTRA_\infty$ and k-$FTRA_\infty$, and the techniques developed will be useful for

other variants of the resource allocation problems. For ease of analysis and implementation, the algorithms presented mostly follow the pseudocode style.

6.2 A unified LP-rounding algorithm

The algorithm ULPR (Algorithm 6.1) starts by solving the primal and dual LPs to get the optimal solutions ($\boldsymbol{x}^*$, $\boldsymbol{y}^*$) and ($\boldsymbol{\alpha}^*$, $\boldsymbol{\beta}^*$, $\boldsymbol{z}^*$) to work with. In order to utilize the dual LP for analyzing the approximation ratio of the output solution ($\boldsymbol{x}$, $\boldsymbol{y}$), we need to first deal with how to bound the $-\sum_{i\in\mathcal{F}} R_i z_i$ term in the dual objective function, introduced by imposing the new resource constraint $\forall i \in \mathcal{F} : y_i \leq R_i$ in the primal LP. To resolve this, we exploit the dual CSC (C5). This condition guides us to come up with Stage 1 of the algorithm ULPR which fully opens all (facilities of) sites with $y_i^* = R_i$ and put these sites into the set $\mathcal{P}$ for pruning in the future. Moreover, for successfully deriving the bound stated in Lemma 6.2.1, in the algorithm the client connections x_{ij}^* with the opened sites in $\mathcal{P}$ are rounded up to $\lceil x_{ij}^* \rceil$; in the analysis the other primal and dual CSCs are also exploited. At the end of Stage 1, for each j, we calculate its established connection $\hat{r_j}$, residual connection requirement $\bar{r_j}$ and record its connected sites not in $\mathcal{P}$ as set $\mathcal{F}_j$ for the use of next stage.

While most LP-rounding algorithms round optimal solutions with values in [0, 1], for $FTRA$, our approach directly rounds the solutions with values in [0, R_i] and later we shall analyze its correctness via establishing some useful properties. Like the major LP-rounding algorithms [131, 28, 29, 89] for UFL, Stage 2 of our algorithm also inherits the classical iterative clustering idea [129, 42]. The clustering and rounding here terminate when all $\bar{r_j}$'s are satisfied, i.e. the set of not-fully-connected clients $\bar{\mathcal{C}} = \emptyset$ in the algorithm. Stage 2 consists of two substages 2.1 and 2.2, dealing with cluster construction and cluster guided rounding respectively. Stage 2.1 essentially adopts the facility cloning idea [132] for the deterministic rounding of $FTFL$. Nevertheless, here we are splitting sites. In each iteration, it first picks the cluster center j_o with the smallest optimal dual value, and then builds a cluster $\mathcal{S}$ around it which contains a subset of ordered sites in $\mathcal{F}_{j_o}$, starting from the cheapest site until $\sum_{i\in\mathcal{S}} y_i^* \geq \bar{r_{j_o}}$. In order to maintain the invariant $\forall j \in \bar{\mathcal{C}} : \sum_{i\in\mathcal{F}_j} y_i^* \geq \bar{r_j}$ in every iteration, the stage then splits the last site $i_l \in \mathcal{S}$ into i_1 and i_2, updates the client connections w.r.t. i_1 and i_2, and in $\mathcal{S}$ includes i_1 while excluding i_l to keep $\sum_{i\in\mathcal{S}} y_i^* = \bar{r_{j_o}}$.

Algorithm 6.1 ULPR: Unified LP-Rounding Algorithm

Input: $\mathcal{F}$, $\mathcal{C}$, $\boldsymbol{f}$, $\boldsymbol{c}$, $\boldsymbol{r}$, $\boldsymbol{R}$. **Output:** $(\boldsymbol{x}, \boldsymbol{y})$
Initialization: Solve LPs (6.2) and (6.3) to obtain the optimal fractional solutions $(\boldsymbol{x^*}, \boldsymbol{y^*})$ and $(\boldsymbol{\alpha^*}, \boldsymbol{\beta^*}, \boldsymbol{z^*})$. $\boldsymbol{x} \leftarrow \boldsymbol{0}$, $\boldsymbol{y} \leftarrow \boldsymbol{0}$, $\mathcal{P} \leftarrow \emptyset$
Stage 1: Pruning and Rounding
for $i \in \mathcal{F}$
  if $y_i^* = R_i$ **do**
    $y_i \leftarrow R_i$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{i\}$
    for $j \in \mathcal{C}$
      if $x_{ij}^* > 0$ **do**
        $x_{ij} \leftarrow \lceil x_{ij}^* \rceil$
set $\forall j \in \mathcal{C}$: $\hat{r_j} \leftarrow \sum_{i \in \mathcal{P}} x_{ij}$, $\bar{r_j} \leftarrow r_j - \hat{r_j}$, $\mathcal{F}_j \leftarrow \left\{ i \in \mathcal{F} \backslash \mathcal{P} \,|\, x_{ij}^* > 0 \right\}$

Stage 2: Clustered Rounding
set $\bar{\mathcal{C}} \leftarrow \{ j \in \mathcal{C} \,|\, \bar{r_j} \geq 1 \}$
while $\bar{\mathcal{C}} \neq \emptyset$
  //**2.1**: Construct a cluster $\mathcal{S}$ centered at j_o
  $j_o \leftarrow \arg\min_j \left\{ \alpha_j^* : j \in \bar{\mathcal{C}} \right\}$, order $\mathcal{F}_{j_o}$ by non-decreasing site facility costs
  pick $\mathcal{S} \subseteq \mathcal{F}_{j_o}$ starting from the cheapest site in $\mathcal{F}_{j_o}$ s.t. just $\sum_{i \in \mathcal{S}} y_i^* \geq \bar{r_{j_o}}$
  if $\sum_{i \in \mathcal{S}} y_i^* > \bar{r_{j_o}}$ **do**
    split the last most expensive site $i_l \in \mathcal{S}$ into i_1 and i_2:
    $y_{i_1}^* = \bar{r_{j_o}} - \sum_{i \in \mathcal{S} \backslash i_l} y_i^*$, $y_{i_2}^* = y_{i_l}^* - y_{i_1}^*$;
    forall j: set $x_{i_1 j}^*$, $x_{i_2 j}^*$ s.t. $x_{i_1 j}^* + x_{i_2 j}^* = x_{i_l j}^*$, $x_{i_1 j}^* \leq y_{i_1}^*$
    $x_{i_2 j}^* \leq y_{i_2}^*$ and update $\mathcal{F}_j$; $\mathcal{S} \leftarrow \mathcal{S} \backslash \{i_l\} \cup \{i_1\}$ (now $\sum_{i \in \mathcal{S}} y_i^* = \bar{r_{j_o}}$)
  //**2.2**: Rounding around j_o and $\mathcal{S}$
  //**2.2.1**: Finish rounding $\boldsymbol{y}$
  for $i \in \mathcal{S}$ from the cheapest site
    $y_i \leftarrow \lceil y_i^* \rceil$
    $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} \cup \{i\}$ //maintain a set of already rounded sites
    if $\sum_{i' \in \bar{\mathcal{S}}} y_{i'} \geq \bar{r_{j_o}}$
      $y_i \leftarrow \bar{r_{j_o}} - \sum_{i' \in \bar{\mathcal{S}} \backslash i} y_{i'}$ (resetting y_i to make $\sum_{i' \in \bar{\mathcal{S}}} y_i = \bar{r_{j_o}}$)
      break
  //**2.2.2**: Finish rounding $\boldsymbol{x}$
  for $j \in \bar{\mathcal{C}}$ //including j_o
    if $\mathcal{F}_j \cap \mathcal{S} \neq \emptyset$
      for $i \in \bar{\mathcal{S}}$ //order does not matter, could connect to the closest
        $x_{ij} \leftarrow \min(\bar{r_j}, y_i)$
        $\bar{r_j} \leftarrow \bar{r_j} - x_{ij}$
    $\mathcal{F}_j = \mathcal{F}_j \backslash \mathcal{S}$
  update $\bar{\mathcal{C}}$

Stage 2.2 does the final rounding steps around $\mathcal{S}$ in addition to Stage 1 to produce a feasible integral solution $(\boldsymbol{x}, \boldsymbol{y})$. This stage modifies and generalizes the rounding steps for $FTFL$. Its substage 2.2.1 rounds up the sites ($y_i^* \rightarrow$

$\lceil y_i^* \rceil$) from the cheapest site in $\mathcal{S}$ until $\bar{\mathcal{S}}$ (the set of sites rounded so far) just satisfies $\sum_{i' \in \bar{\mathcal{S}}} y_{i'} \geq \bar{r_{j_o}}$ (now these $y_{i'}$'s are already integral). To make sure $\sum_{i' \in \bar{\mathcal{S}}} y_i = \bar{r_{j_o}}$ for bounding the site facility opening cost (cf. Lemma 6.2.2), the integral facility opening y_i of the last site i in $\bar{\mathcal{S}}$ is reset to $\bar{r_{j_o}} - \sum_{i' \in \bar{\mathcal{S}} \setminus i} y_{i'}$, which is also integral. After the facilities at the sites in $\bar{\mathcal{S}}$ are opened according to the y_i's constructed in stage 2.2.1, stage 2.2.2 then connects every client j in $\bar{\mathcal{C}}$ which has connections to the sites in $\mathcal{S}$ (according to the x_{ij}^*'s) to min $(\bar{r_j}, \bar{r_{j_o}})$ of these open facilities. It does this by iterating through all sites in $\bar{\mathcal{S}}$, setting x_{ij}'s and updating $\bar{r_j}$'s as described in the algorithm. At the end, for the run of next iteration, the sites in the cluster $\mathcal{S}$ are excluded from $\mathcal{F}_j$, implying all clusters chosen in the iterations are disjoint; and $\bar{\mathcal{C}}$ is updated (at least j_o is removed from the set).

In the analysis, we first demonstrate the overall correctness of the algorithm ensured by the following properties. Note that some of the proofs in this section frequently refer to the LP properties in Section 6.1.

(P3) After Stage 1, $\forall i \in \mathcal{P}$, $j \in \mathcal{C} : x_{ij} \leq R_i$ and $\bar{r_j} = r_j - \hat{r_j} \geq 0$.

Proof. The first part of the property is obvious since $x_{ij} = 0$ or $x_{ij} = \lceil x_{ij}^* \rceil \leq \lceil y_i^* \rceil \leq R_i$. For the second part, $\forall j \in \mathcal{C}$: if all x_{ij}^* are integers, we are done. Now we only need to consider j's fractional x_{ij}^* connecting with $\mathcal{P}$. By the previous property (P2), there is at most one fractional x_{ij}^* with $\mathcal{P}$ because all y_i^*'s in $\mathcal{P}$ are integers. Therefore, in Stage 1, at most one fractional x_{ij}^* is rounded up which will not make $\hat{r_j}$ exceed r_j. □

(P4) Stage 2.2.1 rounds $y_{i_1}^*$ (the optimal fractional opening of the last site i_1 in $\mathcal{S}$ which is included in Stage 2.1) to at most $\lfloor y_{i_1}^* \rfloor$.

Proof. If $y_{i_1}^*$ is integral, the property clearly holds. Otherwise if $y_{i_1}^*$ is fractional, Case 1): if $\sum_{i' \in \bar{\mathcal{S}}} y_{i'} = \bar{r_{j_o}}$ before resetting the last site i in Stage 2.2.1, this i definitely appears before i_1 in $\mathcal{S}$ because otherwise $\sum_{i' \in \bar{\mathcal{S}}} \lceil y_{i'}^* \rceil$ will exceed $\bar{r_{j_o}}$, therefore i_1 is left unrounded; Case 2): If $\sum_{i' \in \bar{\mathcal{S}}} y_{i'} > \bar{r_{j_o}}$, the last site i is possibly i_1 and if it is then $\bar{\mathcal{S}} = \mathcal{S}$, and from the algorithm we have rounded $y_{i_1} = \bar{r_{j_o}} - \sum_{i' \in \mathcal{S} \setminus i_1} \lceil y_{i'}^* \rceil$ after resetting. If $y_{i_1} > \lfloor y_{i_1}^* \rfloor$ we get $\sum_{i' \in \mathcal{S} \setminus i_1} \lceil y_{i'}^* \rceil + \lfloor y_{i_1}^* \rfloor < \bar{r_{j_o}}$ which is not possible since $\sum_{i \in \mathcal{S}} y_i^* = \bar{r_{j_o}} = \sum_{i' \in \mathcal{S} \setminus i_1} y_{i'}^* + y_{i_1}^* = \left\lceil \sum_{i' \in \mathcal{S} \setminus i_1} y_{i'}^* \right\rceil + \lfloor y_{i_1}^* \rfloor \leq \sum_{i' \in \mathcal{S} \setminus i_1} \lceil y_{i'}^* \rceil + \lfloor y_{i_1}^* \rfloor$ (because $y_{i_1}^*$ is fractional). Hence, $y_{i_1}^*$ is rounded to at most $\lfloor y_{i_1}^* \rfloor$. □

(P5) $\forall i \in \mathcal{F}$: given $0 < y_{i_1}^* + y_{i_2}^* = y_i^* \leq R_i$, then we have $\lfloor y_{i_1}^* \rfloor + \lceil y_{i_2}^* \rceil \leq \lceil y_i^* \rceil \leq R_i$.

Proof. We first have $\lfloor y^*_{i_1} \rfloor \leq y^*_{i_1}$ and $\lceil y^*_{i_2} \rceil < y^*_{i_2} + 1$, so $\lfloor y^*_{i_1} \rfloor + \lceil y^*_{i_2} \rceil < y^*_{i_1} + y^*_{i_2} + 1 = y^*_i + 1$. Now if y^*_i is integral, because $\lfloor y^*_{i_1} \rfloor + \lceil y^*_{i_2} \rceil$ is also integral, $\lfloor y^*_{i_1} \rfloor + \lceil y^*_{i_2} \rceil \leq \lceil y^*_i \rceil$. Otherwise if y^*_i is fractional, $\lfloor y^*_{i_1} \rfloor + \lceil y^*_{i_2} \rceil \leq \lfloor y^*_i \rfloor + 1 = \lceil y^*_i \rceil$. The property then follows from $\forall i \in \mathcal{F} : \lceil y^*_i \rceil \leq R_i$. □

In summary, property (P3) shows the correctness of Stage 1 before going into Stage 2, (P4) and (P5) together ensure the splitting in Stage 2.1 and the rounding in Stage 2.2.1 produce feasible y_i's for $FTRA$. This is because for any split sites i_1 and i_2 from i, (P4) guarantees at i_1 at most $\lfloor y^*_{i_1} \rfloor$ facilities are open, and (P5) makes sure that even $\lceil y^*_{i_2} \rceil$ facilities are opened at i_2 in the subsequent iterations of the algorithm, no more than R_i facilities in total actually get opened at i. Note that, (P5) also covers the situation that a site is repeatedly (recursively) split. Furthermore, in each iteration, Stage 2 at least fully connects the client j_o and considers all sites in the cluster $\mathcal{S}$ centered at j_o. More importantly, the invariant $\forall j \in \bar{\mathcal{C}} : \sum_{i \in \mathcal{F}_j} y^*_i \geq \bar{r_j}$ is maintained for choosing the feasible cluster $\mathcal{S}$ in Stage 2.1. This is true in the first iteration. In the subsequent iterations, the invariant still preserves because for any j with $\mathcal{F}_j \cap \mathcal{S} \neq \emptyset$ that is not fully connected in the current iteration, in the next iteration, $\sum_{i \in \mathcal{F}_j} y^*_i$ is decreased by at most $\bar{r_{j_o}}$ (because Stage 2.1 splits sites to maintain $\sum_{i \in \mathcal{S}} y^*_i = \bar{r_{j_o}}$ and $\mathcal{S}$ is excluded from $\mathcal{F}_j$ in Stage 2.2.2) and $\bar{r_j}$ is decreased by exactly $\bar{r_{j_o}}$ (from Stage 2.2.2). Therefore, the overall algorithm is correct.

Furthermore, the time complexity of the rounding stages of Algorithm 6.1 is $O(n^3)$ since each iteration of Stage 2 at least fully connects one of n_c clients which takes time $O(n^2)$. In the following, we separately bound the partial solution costs incurred in the stages involving rounding and then combine these costs for achieving the approximation ratio.

Lemma 6.2.1. *After pruning and rounding, the partial total cost from Stage 1 is* $\sum_{j \in \mathcal{C}} \hat{r_j} \alpha^*_j - \sum_{i \in \mathcal{F}} R_i z^*_i$.

Proof. $\forall i \in \mathcal{P}$:

$$\begin{aligned}
\sum_{j \in \mathcal{C}} \lceil x^*_{ij} \rceil \alpha^*_j &= \sum_{j \in \mathcal{C}} \lceil x^*_{ij} \rceil c_{ij} + \sum_{j \in \mathcal{C}} \lceil x^*_{ij} \rceil \beta^*_{ij} \\
&= \sum_{j \in \mathcal{C}} \lceil x^*_{ij} \rceil c_{ij} + \sum_{j:\, x^*_{ij} = y^*_i = R_i} \lceil x^*_{ij} \rceil \beta^*_{ij} + \sum_{j:\, x^*_{ij} < y^*_i = R_i} \lceil x^*_{ij} \rceil \beta^*_{ij}
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{j\in\mathcal{C}} \left\lceil x^*_{ij} \right\rceil c_{ij} + \sum_{j:\, x^*_{ij}=y^*_i=R_i} \left\lceil x^*_{ij} \right\rceil \beta^*_{ij} \\
&= \sum_{j\in\mathcal{C}} \left\lceil x^*_{ij} \right\rceil c_{ij} + R_i \left(\sum_{j:\, x^*_{ij}=y^*_i=R_i} \beta^*_{ij} + \sum_{j:\, x^*_{ij}<y^*_i=R_i} \beta^*_{ij} \right) \\
&= \sum_{j\in\mathcal{C}} \left\lceil x^*_{ij} \right\rceil c_{ij} + R_i \sum_{j\in\mathcal{C}} \beta^*_{ij} \\
&= \sum_{j\in\mathcal{C}} \left\lceil x^*_{ij} \right\rceil c_{ij} + R_i f_i + R_i z^*_i .
\end{aligned}
$$

The first equality is due to the complementary slackness condition (C1), the third, fourth and fifth is because by (C4) we have $\forall i \in \mathcal{P},\ j \in \mathcal{C}$: if $\beta^*_{ij} > 0$ then $x^*_{ij} = y^*_i = R_i$, so $x^*_{ij} < y^*_i = R_i$ implies $\beta^*_{ij} = 0$ (by the contraposition in logic) and also $\sum_{j:\, x^*_{ij}<y^*_i=R_i} \beta^*_{ij} = 0$. The last equality is obtained from (C2), and the fact that $\forall i \in \mathcal{P} :\ y^*_i = R_i > 0$.

Summing both sides over all $i \in \mathcal{P}$, we bound the cost of Stage 1:

$$
\begin{aligned}
\textstyle\sum_{i\in\mathcal{P}} \sum_{j\in\mathcal{C}} \left\lceil x^*_{ij} \right\rceil c_{ij} + \sum_{i\in\mathcal{P}} R_i f_i \quad &= \sum_{i\in\mathcal{P}} \sum_{j\in\mathcal{C}} \left\lceil x^*_{ij} \right\rceil \alpha^*_j - \sum_{i\in\mathcal{P}} R_i z^*_i \\
&= \sum_{j\in\mathcal{C}} \hat{r_j} \alpha^*_j - \sum_{i\in\mathcal{F}} R_i z^*_i .
\end{aligned}
$$

The second equality follows from Stage 1 that $\forall j \in \mathcal{C} :\ \hat{r_j} = \sum_{i\in\mathcal{P}} \left\lceil x^*_{ij} \right\rceil$, and the condition (C5): if $z^*_i > 0$ then $y^*_i = R_i$, so $y^*_i < R_i$ implies $z^*_i = 0$. □

Lemma 6.2.2. *After rounding $\boldsymbol{y}$, the partial site facility opening cost from Stage 2.2.1 is at most $\sum_{i\in\mathcal{F}\setminus\mathcal{P}} f_i y^*_i$.*

Proof. Facilities at sites $i \in \mathcal{S} \subseteq \mathcal{F}\setminus\mathcal{P}$ are opened in f_i's non-decreasing order in Stage 2.2.1: In any iteration of the algorithm with picked cluster $\mathcal{S}$, before rounding we have $\sum_{i\in\mathcal{S}} y^*_i = \bar{r_{j_o}}$; after rounding set $\bar{\mathcal{S}}$ is formed starting from the cheapest site in $\mathcal{S}$ s.t. $\sum_{i'\in\bar{\mathcal{S}}} y_i = \bar{r_{j_o}}$. This makes the opening cost of all sites in cluster $\bar{\mathcal{S}}$ at most $\sum_{i\in\mathcal{S}} f_i y^*_i$. The lemma then follows from the fact that all chosen clusters are disjoint in the algorithm. □

Lemma 6.2.3. *After rounding $\boldsymbol{x}$, the partial connection cost from Stage 2.2.2 is at most $3\sum_{j\in\mathcal{C}} \bar{r_j} \alpha^*_j$.*

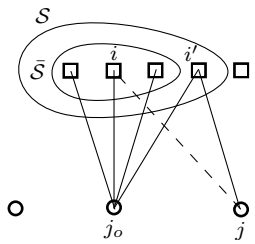

Figure 6.2: Illustration of bounding the connection costs

Proof. Let site i lie in the cluster $\mathcal{S}$ centered at j_o. If j is already connected to i ($x^*_{ij} > 0$), then $c_{ij} \leq \alpha^*_j$ from the condition (C1). Otherwise, if j connects to i after rounding, from the algorithm, it implies $\alpha^*_{j_o} \leq \alpha^*_j$ (because j with the smallest α^*_j is always chosen as j_o) and $\mathcal{F}_j \cap \mathcal{S} \neq \emptyset$. Figure 6.2 then displays the case $\mathcal{F}_j \cap \mathcal{S} \neq \emptyset$ where initially j connects to i' and it is connected to i after rounding. By the triangle inequality, we have $c_{ij} \leq c_{i'j} + c_{ij_o} + c_{i'j_o}$. Also, it is true that $x^*_{i'j}$, $x^*_{ij_o}$, $x^*_{i'j_o} > 0$, so from (C1) we have $c_{i'j} \leq \alpha^*_j$ and c_{ij_o}, $c_{i'j_o} \leq \alpha^*_{j_o}$. Hence, $c_{ij} \leq \alpha^*_j + 2\alpha^*_{j_o} \leq 3\alpha^*_j$. Since each j makes $\bar{r_j}$ connections, the total partial connection cost bound is $3\sum_{j\in\mathcal{C}} \bar{r_j}\alpha^*_j$ and the lemma follows. Note that Figure 6.2 does not show multiplicity of the connection between any client and site in an $FTRA$ solution. It is merely for simplicity and will not affect the correctness of the proof. □

Theorem 6.2.4. *The algorithm ULPR is 4-approximation for $FTRA$.*

Proof. Adding up the partial cost bounds stated in the previous lemmas, the total cost $cost\,(\boldsymbol{x},\, \boldsymbol{y})$ is therefore at most $\sum_{j\in\mathcal{C}} \hat{r_j}\alpha^*_j - \sum_{i\in\mathcal{F}} R_i z^*_i + \sum_{i\in\mathcal{F}\setminus\mathcal{P}} f_i y^*_i + 3\sum_{j\in\mathcal{C}} \bar{r_j}\alpha^*_j$. Also, we have $cost\,(\boldsymbol{x^*},\, \boldsymbol{y^*}) = \sum_{i\in\mathcal{F}} f_i y^*_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x^*_{ij} = cost\,(\boldsymbol{\alpha^*},\, \boldsymbol{\beta^*},\, \boldsymbol{z^*}) = \sum_{j\in\mathcal{C}} r_j\alpha^*_j - \sum_{i\in\mathcal{F}} R_i z^*_i = \sum_{j\in\mathcal{C}} \hat{r_j}\alpha^*_j + \sum_{j\in\mathcal{C}} \bar{r_j}\alpha^*_j - \sum_{i\in\mathcal{F}} R_i z^*_i$, so $cost\,(\boldsymbol{x},\, \boldsymbol{y}) \leq \sum_{j\in\mathcal{C}} \hat{r_j}\alpha^*_j - \sum_{i\in\mathcal{F}} R_i z^*_i + \left(\sum_{j\in\mathcal{C}} \hat{r_j}\alpha^*_j + \sum_{j\in\mathcal{C}} \bar{r_j}\alpha^*_j - \sum_{i\in\mathcal{F}} R_i z^*_i\right) + 3\sum_{j\in\mathcal{C}} \bar{r_j}\alpha^*_j = 4\sum_{j\in\mathcal{C}} \bar{r_j}\alpha^*_j + 2\sum_{j\in\mathcal{C}} \hat{r_j}\alpha^*_j - 2\sum_{i\in\mathcal{F}} R_i z^*_i = 4cost\,(\boldsymbol{\alpha^*},\, \boldsymbol{\beta^*},\, \boldsymbol{z^*}) - 2\left(\sum_{j\in\mathcal{C}} \hat{r_j}\alpha^*_j - \sum_{i\in\mathcal{F}} R_i z^*_i\right) \leq 4cost\,(\boldsymbol{x^*},\, \boldsymbol{y^*})$. The last inequality follows from the fact that $\left(\sum_{j\in\mathcal{C}} \hat{r_j}\alpha^*_j - \sum_{i\in\mathcal{F}} R_i z^*_i\right)$ is the cost of Stage 1 (cf. Lemma 6.2.1) which is nonnegative. □

6.3 Reduction to $FTFL$

Recently, the authors in [149] presented a splitting idea that is able to reduce any $FTRA_\infty$ instance with arbitrarily large r_j to another small $FTRA_\infty$ instance with polynomially bounded r_j. The direct consequence of this is that $FTRA_\infty$ is then reducible to $FTFL$, since we are able to naively split the sites

of the small $FTRA_\infty$ instance and the resulting instance is equivalent to an $FTFL$ instance with a polynomial size. Because $FTRA$ and $FTRA_\infty$ share a similar combinatorial structure and $FTRA_\infty$ is a special case of $FTRA$, the question then becomes whether the more general $FTRA$ reduces to $FTFL$ as well. In the following, we give an affirmative answer to this question with an *instance shrinking* technique.

Compared to the reduction in [149] for $FTRA_\infty$, first, the instance shrinking technique is more general. This is because the technique reduces any $FTRA$ instance with arbitrarily large R_i to another small $FTRA$ instance with polynomially bounded R_i^s, which works for $FTRA_\infty$ as well since an $FTRA_\infty$ instance can be treated as an $FTRA$ instance with all R_i's set to be $\max_{j\in\mathcal{C}} r_j$. The small $FTRA$ instance is then equivalent to an $FTFL$ instance with a polynomial size ($\sum_{i\in\mathcal{F}} R_i^s$), implying $FTRA$ and $FTFL$ share the same approximability in weakly polynomial time. Second, the reduction for $FTRA_\infty$ does not have a mechanism for bounding R_i^s polynomially in $FTRA$. Therefore, it can not directly yield a reduction result for $FTRA$. On the other hand, our technique initially includes the following *crucial* instance shrinking mechanism for bounding R_i^s.

Claim 6.3.1. $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ remains to be the optimal solution even if R_i is replaced with $\lceil y_i^* \rceil$ in LP (6.2).

Proof. Denote the instance with parameter R_i as $\mathcal{I}_o$, and $\mathcal{I}$ after replacing R_i with $\lceil y_i^* \rceil$. On one hand, solving $\mathcal{I}$ will not yield any better optimal solution $\left(\tilde{\boldsymbol{x}^*}, \tilde{\boldsymbol{y}^*}\right)$ with $cost\left(\tilde{\boldsymbol{x}^*}, \tilde{\boldsymbol{y}^*}\right) < cost\left(\boldsymbol{x}^*, \boldsymbol{y}^*\right)$, because this $\left(\tilde{\boldsymbol{x}^*}, \tilde{\boldsymbol{y}^*}\right)$ is also feasible to $\mathcal{I}_o$, which contradicts the optimality of $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ for $\mathcal{I}_o$. On the other hand, $cost\left(\tilde{\boldsymbol{x}^*}, \tilde{\boldsymbol{y}^*}\right) > cost\left(\boldsymbol{x}^*, \boldsymbol{y}^*\right)$ is not possible since $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is also a feasible solution to $\mathcal{I}$ as $y_i^* \leq \lceil y_i^* \rceil$, which contradicts the optimality of $\left(\tilde{\boldsymbol{x}^*}, \tilde{\boldsymbol{y}^*}\right)$ for $\mathcal{I}$. Hence, $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ stays optimal for $\mathcal{I}$. □

With this mechanism, instead we can consider the equivalent $FTRA$ instance $\mathcal{I}$ with $\forall i \in \mathcal{F} : R_i = \lceil y_i^* \rceil$ and the same optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$. Then in the reduction, $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ is split into a large integral solution with $y_i^l = \max\left(0, \lfloor y_i^* \rfloor - 1\right)$ and $x_{ij}^l = \min\left(\lfloor x_{ij}^* \rfloor, y_i^l\right)$, and a small fractional solution with $y_i^s = y_i^* - y_i^l$ and $x_{ij}^s = x_{ij}^* - x_{ij}^l$, for all $i \in \mathcal{F}, j \in \mathcal{C}$. Let the tuple $\langle \mathcal{F}, \mathcal{C}, \boldsymbol{f}, \boldsymbol{c}, \boldsymbol{r}, \boldsymbol{R} \rangle$ represent the instance $\mathcal{I}$, the reduction then proceeds by splitting $\mathcal{I}$ into a large instance $\mathcal{I}^l$: $\langle \mathcal{F}, \mathcal{C}, \boldsymbol{f}, \boldsymbol{c}, \boldsymbol{r^l}, \boldsymbol{R^l} \rangle$ and a small instance $\mathcal{I}^s$: $\langle \mathcal{F}, \mathcal{C}, \boldsymbol{f}, \boldsymbol{c}, \boldsymbol{r^s}, \boldsymbol{R^s} \rangle$ according to $\left(\boldsymbol{x^l}, \boldsymbol{y^l}\right)$ and $(\boldsymbol{x^s}, \boldsymbol{y^s})$. In particular, these two instances differ at two parameters $\boldsymbol{r}$ and $\boldsymbol{R}$, where we let

$r_j^l = \sum_{i\in\mathcal{F}} x_{ij}^l$, $r_j^s = r_j - r_j^l$ and $R_i^l = y_i^l$, $R_i^s = \lceil y_i^* \rceil - y_i^l$. Note that although the above splitting idea of the instance shrinking technique is inspired from the reduction for $FTRA_\infty$, the focus on splitting R_i is essentially different from reducing r_j. Also, here we can see that the construction of the shrunken instance $\mathcal{I}$ with $R_i = \lceil y_i^* \rceil$ is crucial for bounding R_i^s, since if the original R_i is used, R_i^s can not be bounded and the technique will not work. In the following, the first lemma mostly results from the original splitting idea where we provide a simpler proof for it. The second is directly from our instance shrinking and splitting on R_i. As shown later in the proof of Theorem 6.3.4, these lemmas are necessary for the approximation preserving reduction from $\mathcal{I}$ to $\mathcal{I}^s$.

Lemma 6.3.2. *$\left(\boldsymbol{x^l}, \boldsymbol{y^l}\right)$ is a feasible integral solution to $\mathcal{I}^l$ and $(\boldsymbol{x^s}, \boldsymbol{y^s})$ is a feasible fractional solution to $\mathcal{I}^s$.*

Proof. According to the LP (6.2), it is trivial to see the feasibility of the integral solution $\left(\boldsymbol{x^l}, \boldsymbol{y^l}\right)$. For the fractional solution $(\boldsymbol{x^s}, \boldsymbol{y^s})$, since $r_j = \sum_{i\in\mathcal{F}} x_{ij}^*$, $r_j^l = \sum_{i\in\mathcal{F}} x_{ij}^l$, $r_j^s = r_j - r_j^l$ and $x_{ij}^s = x_{ij}^* - x_{ij}^l$, we have $r_j^s = \sum_{i\in\mathcal{F}} x_{ij}^s$ and the first constraint of the LP holds. Further, it is easy to see $y_i^s \le R_i^s$ and we are left to show the second constraint $\forall i \in \mathcal{F}$, $j \in \mathcal{C}$: $y_i^s - x_{ij}^s \ge 0$ holds, i.e. $y_i^* - y_i^l \ge x_{ij}^* - \min\left(\lfloor x_{ij}^* \rfloor, y_i^l\right)$. Consider two cases: 1) $y_i^l \le \lfloor x_{ij}^* \rfloor$, then the inequality obviously follows from $y_i^* \ge x_{ij}^*$; 2) $y_i^l > \lfloor x_{ij}^* \rfloor$, the inequality $rhs = x_{ij}^* - \lfloor x_{ij}^* \rfloor$, and $lhs = y_i^* - \max(0, \lfloor y_i^* \rfloor - 1)$ after substituting y_i^l. Now again consider two sub cases: 2.1) $\lfloor y_i^* \rfloor \ge 1$, then $lhs \ge 1$ while $rhs \le 1$, so $lhs \ge rhs$ and the inequality follows; 2.2) $\lfloor y_i^* \rfloor < 1$, then $lhs = y_i^*$, and since $1 > y_i^* \ge x_{ij}^*$, $\lfloor x_{ij}^* \rfloor = 0$ and $rhs = x_{ij}^*$, then the inequality follows. Overall, $\left(x_{ij}^s, y_i^s\right)$ is a feasible solution. □

Lemma 6.3.3. *For the instances $\mathcal{I}^l$ and $\mathcal{I}^s$ the following holds:*

(i) $\max_{j\in\mathcal{C}} r_j^l \le \sum_{i\in\mathcal{F}} R_i^l$ *and* $\max_{j\in\mathcal{C}} r_j^s \le \sum_{i\in\mathcal{F}} R_i^s$.

(ii) $R_i^s \in \{0, 1, 2\}$.

Proof. (i) The previous lemma and the constraints of the LP (6.2) together ensure the bounds that $\forall j \in \mathcal{C}$: $r_j^l \le \sum_{i\in\mathcal{F}} x_{ij}^l \le \sum_{i\in\mathcal{F}} y_i^l \le \sum_{i\in\mathcal{F}} R_i^l$ and $r_j^s \le \sum_{i\in\mathcal{F}} x_{ij}^s \le \sum_{i\in\mathcal{F}} y_i^s \le \sum_{i\in\mathcal{F}} R_i^s$.

(ii) We have $R_i^s = \lceil y_i^* \rceil - \max(0, \lfloor y_i^* \rfloor - 1)$ after the substitution of y_i^l. If $\lfloor y_i^* \rfloor \ge 1$, then $R_i^s = \lceil y_i^* \rceil - \lfloor y_i^* \rfloor + 1 \in \{1, 2\}$, otherwise if $\lfloor y_i^* \rfloor < 1$, then $R_i^s = \lceil y_i^* \rceil \in \{0, 1\}$. □

Theorem 6.3.4. *If there is a ρ-approximation polynomial-time algorithm $\mathcal{A}$ for the general FTRA with polynomially bounded R_i, which always produces an*

integral solution that approximates the fractional optimal solution with factor $\rho \geq 1$. *Then there is also a polynomial-time* ρ*-approximation algorithm* $\mathcal{A}'$ *for the general* $FTRA$.

Proof. We will describe such an algorithm $\mathcal{A}'$. It first does the instance shrinking and splitting as described before for any instance $\mathcal{I}$ of $FTRA$. From (i) of Lemma 6.3.3, the split instances $\mathcal{I}^l$ and $\mathcal{I}^s$ are valid. From (ii), $\mathcal{I}^s$ has polynomially bounded R_i^s. Note that if $R_i^s = 0$, we can safely remove this site i in $\mathcal{I}^s$, and set the solution $\forall j \in \mathcal{C} : x_{ij}^s,\, y_i^s = 0$ when later combining it with $\mathcal{I}^l$. Then, $\mathcal{A}'$ uses $\mathcal{A}$ as a subroutine to solve $\mathcal{I}^s$ to obtain a feasible integral solution $\left(\bar{x_{ij}^s},\, \bar{y_i^s}\right)$ that approximates the fractional optimum. Also, from Lemma 6.3.2, $(\boldsymbol{x^s},\, \boldsymbol{y^s})$ is feasible to $\mathcal{I}^s$, so $cost\left(\bar{\boldsymbol{x^s}},\, \bar{\boldsymbol{y^s}}\right) \leq \rho \cdot cost\left(\boldsymbol{x^s},\, \boldsymbol{y^s}\right)$. Finally, $\mathcal{A}'$ combines $\left(\bar{x_{ij}^s},\, \bar{y_i^s}\right)$ with the readily available constructed integer solution $\left(x_{ij}^l,\, y_i^l\right)$ for $\mathcal{I}^l$. Because $\left(x_{ij}^l,\, y_i^l\right)$ is a feasible integral solution to $\mathcal{I}^l$, then when combined, they form a feasible integral solution to $\mathcal{I}$ as $r_j^l + r_j^s = r_j$ and $R_i^l + R_i^s = R_i = \lceil y_i^* \rceil$. The only thing left is to prove the combined solution from $\mathcal{A}'$ is ρ-approximation, i.e., $cost\left(\boldsymbol{x^l},\, \boldsymbol{y^l}\right) + cost\left(\bar{\boldsymbol{x^s}},\, \bar{\boldsymbol{y^s}}\right) \leq \rho \cdot cost\left(\boldsymbol{x^*},\, \boldsymbol{y^*}\right)$. This follows from $cost\left(\boldsymbol{x^l},\, \boldsymbol{y^l}\right) + cost\left(\bar{\boldsymbol{x^s}},\, \bar{\boldsymbol{y^s}}\right) \leq cost\left(\boldsymbol{x^l},\, \boldsymbol{y^l}\right) + \rho \cdot cost\left(\boldsymbol{x^s},\, \boldsymbol{y^s}\right) \leq \rho \cdot \left(cost\left(\boldsymbol{x^l},\, \boldsymbol{y^l}\right) + cost\left(\boldsymbol{x^s},\, \boldsymbol{y^s}\right)\right)$ and $\boldsymbol{x^l} + \boldsymbol{x^s} = \boldsymbol{x^*},\, \boldsymbol{y^l} + \boldsymbol{y^s} = \boldsymbol{y^*}$. □

Corollary 6.3.5. *The general* $FTRA$ *is reducible to the general* $FTFL$ *in weakly polynomial time.*

Proof. Any instance of $FTRA$ with polynomially bounded R_i can be treated as an equivalent $FTFL$ instance with facility size $\sum_{i \in \mathcal{F}} R_i$ which is polynomial. Then any polynomial time algorithm solves $FTFL$ with ratio ρ (w.r.t. the fractional optimum) can become the algorithm $\mathcal{A}$ for $\mathcal{A}'$ in the previous theorem to solve $FTRA$ with the same ratio. In addition, the reduction requires solving the LP first to obtain $(\boldsymbol{x^*},\, \boldsymbol{y^*})$ which takes weakly polynomial time. □

Therefore, from the above corollary and the result of [31] for the metric $FTFL$, we get the ratio of 1.7245 for the metric $FTRA$. Also, from the results of [76, 100], we can deduce that the non-metric $FTFL$ has an approximation ratio of $O\left(\log^2 n\right)$. This is because Jain and Vazirani [76] proved that $FTFL$ reduces to UFL with a ratio loss of $O\left(\log n\right)$, and Lin and Vitter [100] showed that the non-metric UFL can be approximated with the ratio of $O\left(\log n\right)$ w.r.t. the fractional optimum. For the non-metric $FTRA$, we can subsequently achieve the same ratio due to its reduction to $FTFL$ first. Moreover, in future,

any improved ratio for the general $FTFL$ might directly hold for the general $FTRA$.

6.4 The uniform $FTRA$

Interestingly, the reduction results in the previous section does not imply that the uniform $FTRA$ reduces to the uniform $FTFL$ in weakly polynomial time. This is because the instance shrinking technique may split a uniform instance into two general/non-uniform instances. As a consequence, the ratio of 1.52 in [132] for the uniform $FTFL$ does not directly hold for the uniform $FTRA$. Nevertheless, in this section, we show this ratio can still be preserved for the uniform $FTRA$ in strongly polynomial time with a primal-dual algorithm and two acceleration heuristics. Note that the following algorithms are generic which work for the general $FTRA$ as well. The uniform condition is only necessary in the analysis (particularly for Lemma 6.4.10).

We begin with a naive primal-dual algorithm PD-4 (Algorithm 6.2) for $FTRA$ with an approximation ratio of 1.61 and then present the first acceleration heuristic to improve the complexity of the algorithm to strongly polynomial $O(n^4)$. The PD-4 algorithm only differs from the PD-2 algorithm for $FTRA_\infty$ (see Chapter 4) in two places. First, Event 2 has an extra condition $y_i < R_i$ for maintaining the feasibility of the solution. In other words, Event 2 on a facility i stops occurring once $y_i = R_i$. Second, the new condition $y_i < R_i$ introduces some difficulties to the analysis. To cope with these difficulties, in Action 1, we use an extra variable $\hat{\boldsymbol{x}}$ to store the amounts of the clients' connections when they just become fully-connected. From the algorithm, it should be obvious that $cost(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j\in\mathcal{C}} \sum_{1\le q\le r_j} \alpha_j^q$.

The next lemma states that the time complexity of the algorithm PD-4 is the same as PD-2. The proof is similar to the proof of Lemma 4.3.3 for $FTRA_\infty$, and the proof of Lemma 5.2.1 for RRA.

Lemma 6.4.1. *The algorithm PD-4 computes a feasible solution to the uniform FTRA and runs in* $O(n^3 \max_{j\in\mathcal{C}} r_j)$.

Proof. The solution is feasible because $(\boldsymbol{x}, \boldsymbol{y})$ produced from PD-4 is feasible to LP (6.1). Each iteration of PD-4 at least connects a port of a client, so there are maximum $\sum_{j\in\mathcal{C}} r_j$ iterations. In addition, similar to Theorem 22.4 of [82] and Theorem 8 of [77] for UFL, the client switching in Action 2 dominates the time complexity. In each iteration, the switching takes time $O(n_c n_f)$ to update

clients' contributions to other facilities for computing the anticipated times of the events. Hence, the total time is $O\left(\sum_{j\in\mathcal{C}} r_j n_c n_f\right)$, i.e. $O\left(n^3 \max_{j\in\mathcal{C}} r_j\right)$. □

Algorithm 6.2 PD-4: Primal-Dual Algorithm for *FTRA*

Input: $\mathcal{F}$, $\mathcal{C}$, $\boldsymbol{f}$, $\boldsymbol{c}$, $\boldsymbol{r}$, $\boldsymbol{R}$. **Output:** $(\boldsymbol{x}, \boldsymbol{y})$.
Initialization: Set $\mathcal{U} = \mathcal{C}$, $\forall i \in \mathcal{F}$, $j \in \mathcal{C}$: x_{ij}, $y_i = 0$, $p_j = 1$.

while $\mathcal{U} \neq \emptyset$, increase time t uniformly and execute the events below:

- Event 1: $\exists i \in \mathcal{F}$, $j \in \mathcal{U}$: $t = c_{ij}$ and $x_{ij} < y_i$.
  Action 1: **set** $\phi\left(j^{(p_j)}\right) \leftarrow i$, $x_{ij} \leftarrow x_{ij} + 1$ and $\alpha_j^{p_j} \leftarrow t$; If $p_j = r_j$, then $\mathcal{U} \leftarrow \mathcal{U} \backslash \{j\}$ and $\hat{x_{ij}} = x_{ij}$, otherwise $p_j \leftarrow p_j + 1$.

- Event 2: $\exists i \in \mathcal{F}$: $\sum_{j\in\mathcal{U}} \max(0,\, t - c_{ij}) +$
  $\sum_{j\in\mathcal{C}\backslash\mathcal{U}} \max\left(0,\, \max_q c_{\phi\left(j^{(q)}\right)j} - c_{ij}\right) = f_i$ and $y_i < R_i$.
  Action 2: **set** $y_i \leftarrow y_i + 1$; $\forall j \in \mathcal{C}\backslash\mathcal{U}$ s.t. $\max_q c_{\phi\left(j^{(q)}\right)j} - c_{ij} > 0$: **set** $i_j^* \leftarrow \arg\max_{\phi\left(j^{(q)}\right)} c_{\phi\left(j^{(q)}\right)j}$, $x_{i_j^* j} \leftarrow x_{i_j^* j} - 1$, $x_{ij} \leftarrow x_{ij} + 1$ and $\phi\left(j^{\left(\arg\max_q c_{\phi\left(j^{(q)}\right)j}\right)}\right) \leftarrow i$; $\forall j \in \mathcal{U}$ s.t. $t \geq c_{ij}$: **do** Action 1.

Remark 6.4.2. If more than one event happen at time t, the PD-4 algorithm processes all of them in an arbitrary order. Also, the events themselves may repeatedly happen at any t because more than one facilities at a site are allowed to open.

The PD-4 algorithm produces a feasible primal solution $(\boldsymbol{x}, \boldsymbol{y})$ to *FTRA* but an infeasible dual solution $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$ if we simply let $\alpha_j = \alpha_j^{r_j}$, $\beta_{ij} = \max(0, \alpha_j - c_{ij})$ and $z_i = 0$ in LP (6.3). This is because although the LP's second constraint holds, the first constraint fails to hold since the algorithm only guarantees $\forall i \in \mathcal{F}$: $\sum_{j\in\mathcal{U}} \max(0,\, t - c_{ij}) \leq f_i$ where $\mathcal{U} \subseteq \mathcal{C}$. In order to get around this feasibility issue, there are ways like the classical dual fitting [73] and inverse dual fitting [145] analyses. However, we observe that these approaches are actually both based on constraints of the LP. Therefore, in the following we develop a simple and step-by-step constraint-based analysis for the ease of handling more complicated dual constructions. Together with the factor-revealing technique of [73], we derive the ratio of 1.61 for *FTRA*.

In the analysis, we first use the following dual constructions of $\boldsymbol{\alpha}$, $\boldsymbol{z}$ to bound the primal solution cost $cost(\boldsymbol{x}, \boldsymbol{y})$ with the dual cost $cost(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$.

$$\forall i \in \mathcal{F},\, j \in \mathcal{C} :\, \alpha_j = \alpha_j^{r_j},\, z_i = \sum_{j \in \mathcal{C}} \pi_{ij} \text{ and } \pi_{ij} = \begin{cases} \frac{\hat{x_{ij}}\left(\alpha_j - \alpha_j^{l_{ij}}\right)}{R_i} & \text{if } \hat{x_{ij}} = R_i \\ 0 & \text{if } \hat{x_{ij}} < R_i \end{cases}.$$

In this setting, $\hat{\boldsymbol{x}}$ stores the primary connection amounts of the clients after they become fully-connected but before they switch any of their connections. $\forall i \in \mathcal{F},\, j \in \mathcal{C} : l_{ij}$ denotes the last port of j *connecting to* i before switching, so $\alpha_j^{l_{ij}}$ is the dual value of the port l_{ij}. $\alpha_j^{r_j}$ is the dual of j's last port and π_{ij} can be interpreted as the conditional marginal dual/price depending on whether $\hat{x_{ij}}$ reaches R_i or not. $\forall i \in \mathcal{F} : z_i$ is therefore the sum of marginal duals of all clients w.r.t. i. The other dual variable $\boldsymbol{\beta}$ is to be constructed later in the analysis.

Lemma 6.4.3. *cost* $(\boldsymbol{x}, \boldsymbol{y}) \leq$ *cost* $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$ *where* $(\boldsymbol{x}, \boldsymbol{y})$ *is the feasible primal solution produced from the PD-4 algorithm and* $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$ *is constructed from above.*

Proof.

$$\begin{aligned} \textit{cost}\,(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z}) &= \sum_{j \in \mathcal{C}} r_j \alpha_j - \sum_{i \in \mathcal{F}} R_i z_i \\ &= \sum_{j \in \mathcal{C}} \sum_{i \in \mathcal{F}} x_{ij} \alpha_j^{r_j} - \underbrace{\sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} \hat{x_{ij}} \left(\alpha_j^{r_j} - \alpha_j^{l_{ij}}\right)}_{\text{if } \hat{x_{ij}} = R_i} - \underbrace{0}_{\text{if } \hat{x_{ij}} < R_i} \\ &= \sum_{j \in \mathcal{C}} \sum_{i \in \mathcal{F}} \hat{x_{ij}} \alpha_j^{r_j} - \underbrace{\sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} \hat{x_{ij}} \left(\alpha_j^{r_j} - \alpha_j^{l_{ij}}\right)}_{\text{if } \hat{x_{ij}} = R_i} \\ &= \underbrace{\sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} \hat{x_{ij}} \alpha_j^{r_j}}_{\text{if } \hat{x_{ij}} < R_i} + \underbrace{\sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} \hat{x_{ij}} \alpha_j^{l_{ij}}}_{\text{if } \hat{x_{ij}} = R_i}. \end{aligned}$$

Hence, *cost* $(\boldsymbol{x}, \boldsymbol{y}) = \sum_{j \in \mathcal{C}} \sum_{1 \leq q \leq r_j} \alpha_j^q \leq \underbrace{\sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} \hat{x_{ij}} \alpha_j^{r_j}}_{\text{if } \hat{x_{ij}} < R_i} + \underbrace{\sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} \hat{x_{ij}} \alpha_j^{l_{ij}}}_{\text{if } \hat{x_{ij}} = R_i} =$ *cost* $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$. □

Second, we exploit the dual constraints of LP (6.3) by relaxing their feasibilities with some relaxation factors. Again, we adopt the bifactor definition (Definition 3.1.8) in Chapter 3 and in the definition let any feasible solution be $SOL = (\boldsymbol{x''}, \boldsymbol{y''})$, then $F_{SOL} = \sum_{i \in \mathcal{F}} f_i y_i''$, $C_{SOL} = \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} c_{ij} x_{ij}''$ and *cost* $(\boldsymbol{x''}, \boldsymbol{y''}) = F_{SOL} + C_{SOL}$. In the following, we consider the feasibility relaxed dual constraints with the relaxation factors ρ_f and ρ_c:

(C6) $\forall i \in \mathcal{F},\, j \in \mathcal{C} :\, \alpha_j - \beta_{ij} \leq \rho_c c_{ij}$.
(C7) $\forall i \in \mathcal{F} :\, \sum_{j \in \mathcal{C}} \beta_{ij} \leq \rho_f f_i + z_i$.

Next, we show that if the dual variables $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$ satisfies these relaxed constraints, the corresponding dual cost will be bounded by any feasible primal cost scaled by the factors ρ_f and ρ_c.

Lemma 6.4.4. *If $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$ satisfies (C6) and (C7) while $SOL = (\boldsymbol{x''}, \boldsymbol{y''})$ is any feasible primal solution, then cost $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z}) \leq \rho_f F_{SOL} + \rho_c C_{SOL}$.*

Proof. Since $(\boldsymbol{x''}, \boldsymbol{y''})$ is any feasible solution, all constraints of the LP (6.2) should hold first. Together with (C6) and (C7), we have:

$$\begin{aligned}
\text{cost}\,(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z}) &= \sum_{j \in \mathcal{C}} r_j \alpha_j - \sum_{i \in \mathcal{F}} R_i z_i \\
&\leq \sum_{j \in \mathcal{C}} \sum_{i \in \mathcal{F}} x''_{ij} \alpha_j - \sum_{i \in \mathcal{F}} y''_i z_i \\
&\leq \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} \left[\beta_{ij} y''_i + (\alpha_j - \beta_{ij})\, x''_{ij}\right] - \sum_{i \in \mathcal{F}} y''_i z_i \\
&\leq \sum_{i \in \mathcal{F}} (\rho_f f_i + z_i)\, y''_i + \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} \rho_c c_{ij} x''_{ij} - \sum_{i \in \mathcal{F}} y''_i z_i \\
&= \sum_{i \in \mathcal{F}} \rho_f f_i y''_i + \sum_{i \in \mathcal{F}} \sum_{j \in \mathcal{C}} \rho_c c_{ij} x''_{ij} = \rho_f F_{SOL} + \rho_c C_{SOL}.
\end{aligned}$$

□

The previous two lemmas and the definition immediately imply the next lemma.

Lemma 6.4.5. *The algorithm PD-4 is (ρ_f, ρ_c)-approximation if $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$ satisfies (C6) and (C7).*

In the last step, we show $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$ indeed satisfies (C6) and (C7), so the algorithm is (ρ_f, ρ_c)-approximation. To satisfy (C6), obviously we can set $\forall i \in \mathcal{F},\, j \in \mathcal{C} :\, \beta_{ij} = \max(0, \alpha_j - \rho_c c_{ij})$ (because $\beta_{ij} \geq 0$), thereby finishing constructing $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$. The rest is to find the actual values of the factors ρ_f and ρ_c to make (C7) hold as well. The next lemma and corollary are more specific forms of the previous lemma, after substituting the setting of $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{z})$ into (C7).

Lemma 6.4.6. *The algorithm PD-4 is (ρ_f, ρ_c)-approximation if $\forall i \in \mathcal{F} :\, \sum_{j \in \mathcal{A}_i} \left(\alpha_j^{r_j} - \rho_c c_{ij} - \pi_{ij}\right) \leq \rho_f f_i$ where $\mathcal{A}_i = \left\{j \in \mathcal{C} \,|\, \alpha_j^{r_j} \geq \rho_c c_{ij}\right\}$.*

Proof. After the substitution, (C7) becomes:

$$\forall i \in \mathcal{F}: \sum_{j\in\mathcal{C}} (\beta_{ij} - \pi_{ij}) \le \rho_f f_i \Rightarrow$$
$$\forall i \in \mathcal{F}: \sum_{j\in\mathcal{C}} \left(\max\left(0,\, \alpha_j^{r_j} - \rho_c c_{ij}\right) - \pi_{ij}\right) \le \rho_f f_i \Rightarrow$$
$$\forall i \in \mathcal{F}: \underbrace{\sum_{j\in\mathcal{C}} \left(\alpha_j^{r_j} - \pi_{ij} - \rho_c c_{ij}\right)}_{\text{if } \alpha_j^{r_j} \ge \rho_c c_{ij}} - \underbrace{\sum_{j\in\mathcal{C}} \pi_{ij}}_{\text{if } \alpha_j^{r_j} < \rho_c c_{ij}} \le \rho_f f_i.$$

Since $\pi_{ij} \ge 0$, it is sufficient to prove $\forall i \in \mathcal{F}: \sum_{j\in\mathcal{A}_i} \left(\alpha_j^{r_j} - \pi_{ij} - \rho_c c_{ij}\right) \le \rho_f f_i$ where $\mathcal{A}_i = \left\{j \in \mathcal{C} \,|\, \alpha_j^{r_j} \ge \rho_c c_{ij}\right\}$ to satisfy the original (C7). □

If we set $\forall i \in \mathcal{F},\, j \in \mathcal{C}: u_{ij} = \alpha_j^{r_j} - \pi_{ij}$, then $u_{ij} = \begin{cases} \alpha_j^{l_{ij}} & \text{if } \hat{x_{ij}} = R_i \\ \alpha_j^{r_j} & \text{if } \hat{x_{ij}} < R_i \end{cases}$ and we have the corollary below.

Corollary 6.4.7. *W.l.o.g., for every site i, order the corresponding $n_i = |\mathcal{A}_i|$ clients in $\mathcal{A}_i = \left\{j \in \mathcal{C} \,|\, \alpha_j^{r_j} \ge \rho_c c_{ij}\right\}$ s.t. $u_{i1} \le \ldots \le u_{in_i}$. Then the PD-4 Algorithm is $(\rho_f,\, \rho_c)$-approximation if $\forall i \in \mathcal{F}: \sum_{j=1}^{n_i} (u_{ij} - \rho_c c_{ij}) \le \rho_f f_i$.*

In addition, for each i, any subset of the clients are ordered from 1 to k_i s.t. $u_{i1} \le \ldots \le u_{ik_i}$. Now, we proceed the proof to find ρ_f and ρ_c with the following lemmas. These lemmas are needed for the factor-revealing technique and they capture the properties of the PD-4 algorithm for the uniform $FTRA$.

Lemma 6.4.8. *For every site i, at time $t = u_{ij} - \epsilon$, $\forall 1 \le h < j < k_i$ let* $\omega_{h,\,j}^i = \begin{cases} u_{ih} & if\ \hat{x_{ih}} = R_i \\ \max_q c_{\phi\left(h^{(q)}\right)h} & if\ \hat{x_{ih}} < R_i \end{cases}$, *then* $\omega_{h,j}^i \ge \omega_{h,j+1}^i$.

Proof. If $\hat{x_{ih}} = R_i$, then $\omega_{h,\,j}^i = \omega_{h,j+1}^i$ (at time $t = u_{i(j+1)} - \epsilon$) $= u_{ih}$. Otherwise, $\hat{x_{ih}} < R_i$ implies $u_{ih} = \alpha_h^{r_h}$, so h is fully-connected at time t since $u_{ih} \le u_{ij}$. Therefore, $\omega_{h,j}^i \ge \omega_{h,j+1}^i$ because a fully-connected client's ports always reconnect to the sites with less connection cost, so its maximum connection cost will never increase. The lemma follows. □

Lemma 6.4.9. *For any site i and ordered k_i clients, $\forall 1 \le j \le k_i$:* $\sum_{h=1}^{j-1} \max\left(0,\, \omega_{h,j}^i - c_{ih}\right) + \sum_{h=j}^{k_i} \max\left(0,\, u_{ij} - c_{ih}\right) \le f_i$.

Proof. For any site i and at time $t = u_{ij} - \epsilon$, if $h < j$ client h's contribution is set to be $\max\left(0,\, \omega_{h,j}^i - c_{ih}\right)$. In particular, from the previous lemma and the setting of u_{ij}, if $\hat{x_{ih}} < R_i$, it implies h is fully-connected at time t and the contribution is $\max\left(0,\, \max_q c_{\phi\left(h^{(q)}\right)h} - c_{ih}\right)$. In addition, if $\hat{x_{ih}} = R_i$ the contribution is $\max\left(0,\, \alpha_h^{l_{ih}} - c_{ih}\right)$. Note that under this case, h still might be

fully-connected at time t, but because $\hat{x_{ih}} = R_i$ and following the algorithm, its contribution should not be set to $\max\left(0,\, \max_q c_{\phi\left(h^{(q)}\right)h} - c_{ih}\right)$ for ensuring the lemma. On the other hand, if $h \geq j$, h is not fully-connected since $t < \alpha_h^{r_h}$, so we set the contribution to $\max(0,\, t - c_{ih})$, i.e. $\max(0,\, u_{ij} - c_{ih})$. From the execution of the algorithm, at any time, the sum of these contributions will not exceed the facility's opening cost at site i, hence the lemma follows. □

Lemma 6.4.10. *For any site i and clients h, j s.t. $1 \leq h < j \leq k_i : r_h = r_j = r$, then $u_{ij} \leq \omega_{h,j}^i + c_{ij} + c_{ih}$.*

Proof. At time $t = u_{ij} - \epsilon$, if all facilities at site i are already open, then $u_{ij} \leq c_{ij}$ and the lemmas holds. Otherwise, if not all facilities are open, then at time t every client $h < j$ is fully-connected. This is because $u_{ih} \leq u_{ij}$ implies $u_{ih} = \alpha_h^{l_{ih}}$ or $\alpha_h^{r_h}$ at the time t. Since h can only connect to less than R_i facilities at i, this contradicts the condition $\hat{x_{ih}} = R_i$ for the setting of u_{ih}, so $u_{ih} = \alpha_h^{r_h}$. In addition, j itself is not fully-connected at t, whereas h is fully-connected and has already connected to r facilities. There is at least a facility to which h is connected but not j. (This is where we must enforce all clients have the uniform connection r.) Denote this facility (site) by i', we have $u_{ij} \leq c_{i'j}$ and $\omega_{h,j}^i \geq c_{i'h}$. Lastly, by the triangle inequality of the metric property, $c_{i'j} \leq c_{i'h} + c_{ij} + c_{ih}$ and then we have the lemma. □

$$
\begin{aligned}
z_k = \text{maximize} \quad & \frac{\sum_{j=1}^{k} \alpha_j}{f + \sum_{j=1}^{k} d_j} \\
\text{subject to} \quad & \forall 1 \leq j < k : \alpha_j \leq \alpha_{j+1} \\
& \forall 1 \leq h < j < k : r_{h,j} \geq r_{h,j+1} \\
& \forall 1 \leq h < j \leq k : \alpha_j \leq r_{h,j} + d_h + d_j \\
& \forall 1 \leq j \leq k : \sum_{h=1}^{j-1} \max(r_{h,j} - d_h,\, 0) + \sum_{h=j}^{k} \max(\alpha_j - d_h,\, 0) \leq f \\
& \forall 1 \leq h \leq j < k : \alpha_j,\, d_j,\, f,\, r_{h,j} \geq 0
\end{aligned}
$$

Consider the above factor-revealing program series (25) of [73]. If we let $k = k_i, \alpha_j = u_{ij}, r_{h,j} = \omega_{h,j}^i$, $f = f_i$, $d_j = c_{ij}$, from the previous lemmas it is clear that u_{ij}, $\omega_{h,j}^i$, f_i and c_{ij} constitute a feasible solution. Also, from Lemma 5.4 and Theorem 8.3 of [73], and Lemma 4 of [108] we can directly get $\forall i \in \mathcal{F} : \sum_{j=1}^{k_i}(u_{ij} - 1.61c_{ij}) \leq 1.61f_i$, $\sum_{j=1}^{k_i}(u_{ij} - 1.78c_{ij}) \leq 1.11f_i$ and $\sum_{j=1}^{k_i}(u_{ij} - 2c_{ij}) \leq f_i$. Furthermore, because $n_i = |\mathcal{A}_i|$ and k_i represents the

size of any subset of the clients, Lemma 6.4.1 and Corollary 6.4.7 directly lead to the following theorem.

Theorem 6.4.11. *The algorithm PD-4 is 1.61-, (1.11, 1.78)- and (1,2)-approximation in time* $O\left(n^3 \max_{j\in\mathcal{C}} r_j\right)$ *for the uniform FTRA.*

Algorithm 6.3 APD-4: Acceleration of the Primal-Dual Algorithm for *FTRA*

Input: $\mathcal{F}$, $\mathcal{C}$, $\boldsymbol{f}$, $\boldsymbol{c}$, $\boldsymbol{r}$, $\boldsymbol{R}$. **Output:** $(\boldsymbol{x}, \boldsymbol{y})$.
Initialization: Set $\mathcal{U} = \mathcal{C}$, $\forall i \in \mathcal{F}$, $j \in \mathcal{C}$: x_{ij}, $y_i = 0$, $FC_j = 0$.

while $\mathcal{U} \neq \emptyset$, increase time t uniformly and execute the events below:

- Event 1: $\exists i \in \mathcal{F}$, $j \in \mathcal{U}$ s.t. $t = c_{ij}$ and $x_{ij} < y_i$.
  Action 1-a: $ToC \leftarrow \min\left(y_i - x_{ij},\, r_j - FC_j\right)$;
  Action 1-b: **set** $x_{ij} \leftarrow x_{ij} + ToC$ and $FC_j \leftarrow FC_j + ToC$;
  Action 1-c: If $FC_j = r_j$ then $\mathcal{U} \leftarrow \mathcal{U} \backslash \{j\}$.

- Event 2: $\exists i \in \mathcal{F}$: $\sum_{j\in\mathcal{U}} \max\left(0,\, t - c_{ij}\right) +$
  $\sum_{j\in\mathcal{C}\backslash\mathcal{U}} \max\left(0,\, \max_{i'\in\mathcal{F}\&\&x_{i'j}>0} c_{i'j} - c_{ij}\right) = f_i$ and $y_i < R_i$.
  Action 2-a: $\mathcal{U}_i \leftarrow \{j \in \mathcal{U} \,|\, t - c_{ij} \geq 0\}$ and $NC \leftarrow \min_{j\in\mathcal{U}_i}\left(r_j - FC_j\right)$;
  Action 2-b: $\mathcal{S}_i \leftarrow \left\{j \in \mathcal{C}\backslash\mathcal{U} \,|\, \max_{i'\in\mathcal{F}\&\&x_{i'j}>0} c_{i'j} - c_{ij} > 0\right\}$, $\forall j \in \mathcal{S}_i$:
  $i_j^* \leftarrow \arg\max_{i'\in\mathcal{F}\&\&x_{i'j}>0} c_{i'j}$ and $NS \leftarrow \min_{j\in\mathcal{S}_i} x_{i_j^* j}$;
  Action 2-c: **set** $ToC \leftarrow \min\left(NC,\, NS,\, R_i - y_i\right)$ and $y_i \leftarrow y_i + ToC$;
  Action 2-d: $\forall j \in \mathcal{S}_i : x_{i_j^* j} \leftarrow x_{i_j^* j} - ToC$ and $x_{ij} \leftarrow x_{ij} + ToC$;
  Action 2-e: $\forall j \in \mathcal{U}_i$: **do** Action 1-b;
  Action 2-f: $\forall j \in \mathcal{U}_i$: **do** Action 1-c.

Remark 6.4.12. For the convenience of analysis, sequential actions of the events are separated as above. If more than one event happen at the same time, the algorithm process Event 2 first so that no repeated events are needed.

The previous PD-4 algorithm runs in pseudo-polynomial time depending on r_j. With the acceleration heuristic described in the following, the algorithm can then change to an essentially identical algorithm APD-4 (Algorithm 6.3) which is strongly polynomial. In the heuristic, $(\boldsymbol{x}, \boldsymbol{y})$ is able to increase at a faster rate rather than 1, through combining the repeated events into a single event to reduce the total number of events to process and hence achieve fast connections. In particular, for Event 2, once a facility of a site i is opened and connected with a group of clients' ports, according to the PD-4 algorithm, additional facilities at i may subsequently open and connect with this group of clients' other ports until their sum of contributions (SOC) becomes insufficient to pay f_i, or $y_i = R_i$. The SOC is not enough any more if a client

in $\mathcal{U}$ appears to be fully-connected, so $\sum_{j\in\mathcal{U}} \max(0,\, t - c_{ij})$ will decrease, or the most expensive connection of a client in $\mathcal{C}\backslash\mathcal{U}$ differs (after switching all such connections), in this case $\sum_{j\in\mathcal{C}\backslash\mathcal{U}} \max\left(0,\, \max_{i'\in\mathcal{F}\&\&x_{i'j}>0} c_{i'j} - c_{ij}\right)$[1] will decrease. Similarly, for Event 1, once a client j's port starts to connect to an already opened facility at a site i, its other ports may get connected to i at the same time until either there are no remaining open facilities at i or j reaches r_j connections.

Formally in the APD-4 algorithm, FC_j denotes the number of established connections of client j and ToC the total number of connections decided to make according to the heuristic. The incremental rate of $(\boldsymbol{x}, \boldsymbol{y})$ is then determined by ToC instead of 1. Moreover, in the more complicated Event 2 on a site i, NC denotes the (minimum) number of connections to make until one of the clients in $\mathcal{U}$ gets fully-connected, and NS the (minimum) number of switches until the most expensive connection of a client in $\mathcal{C}\backslash\mathcal{U}$ changes. Therefore, ToC is calculated as $\min(NC,\, NS,\, R_i - y_i)$, the (minimum) number of connections until the SOC becomes insufficient or $y_i = R_i$. Similarly, for Event 1, ToC is calculated as $\min(y_i - x_{ij},\, r_j - FC_j)$.

Lemma 6.4.13. *With the acceleration heuristic decided by ToC, the numbers of Event 1 and Event 2 in the APD-4 algorithm are bounded by $n_f n_c$ and $(n_c + n_f + n_c n_f)$ respectively which are independent of r_j.*

Proof. In the APD-4 algorithm, the number of Event 1 is at most $n_f n_c$ because for any client j and site i only when $t = c_{ij}$, j exhaustively gets connected to open facilities at site i, and there are n_f sites and n_c clients in total. Moreover, each Event 2 will cause at least one of the following 3 cases: (1) a client j in $\mathcal{U}$ becomes fully-connected; (2) a client j in $\mathcal{C}\backslash\mathcal{U}$ switches all of its most expensive connections; (3) a site opens all its facilities. It is easy to see that there are maximum n_c and n_f cases (1) and (3) respectively, so we are left to bound the number of case (2). For this case, it is important to observe that any client j has at most n_f possible sets of connections where connections in each set associate to the same site. So there are at most $n_c n_f$ such possible sets in total, and each case (2) removes at least one set of a client with currently most expensive connection cost, effectively reducing the number of possible sites for switching, since clients only switch to cheaper connections. Therefore, there are at most $n_c n_f$ case (2) and Event 2 is bounded by $(n_c + n_f + n_c n_f)$. □

[1] For simplicity of the algorithm description, we replace the term $\sum_{j\in\mathcal{C}\backslash\mathcal{U}} \max\left(0,\, \max_q c_{\phi\left(j^{(q)}\right)j} - c_{ij}\right)$ in the PD-4 algorithm with essentially the same term here.

Lemma 6.4.14. *The algorithm APD-4 computes a feasible solution to the $FTRA$ and runs in $O(n^4)$.*

Proof. The solution is feasible because APD-4 is essentially the same as the PD-4 algorithm for $FTRA$ except the implementation of the acceleration heuristic. In addition, from the previous lemma, the number of Event 1 is at most $n_f n_c$, so the numbers of both Action 1-a and 1-b are bounded by $n_f n_c$, while Action 1-c is bounded by n_c since there are n_c clients to be connected in total. In addition, the number of Event 2 is bounded by $(n_c + n_f + n_c n_f)$, so the numbers of Action 2-a, 2-b, 2-c, 2-d and 2-e are bounded by $O(n_c n_f)$ while Action 2-f is included in Action 1-c. Although the presented APD-4 algorithm is continuous on t, in real implementation, it can be easily discretized through finding the the smallest t that satisfy the conditions of events (similar to the Algorithm 2.2 in Chapter 2 for the vertex cover problem). Naively, finding such t for Event 1 takes time $O(n_c n_f)$, and Event 2 takes time $O\left(n_c n_f^2\right)$, so the algorithm runs in $O(n^5)$. However, similar to Theorem 22.4 of [82] and Theorem 8 of [77] for UFL, we can maintain two heaps to reduce the time to find t. In particular, each Action 2-d actually requires extra time $O(n_c n_f)$ after the client switchings to change clients' contributions to all facilities. This is for later eventually updating the anticipated times of the events in the heaps (that takes time $O(n_f \log n_c n_f)$) following each Action 1-c. This action dominates the overall runtime complexity. Hence the total time is $O\left(n_c^2 n_f^2\right)$. □

The above two lemmas imply that the runtime of the PD-2 algorithm for the uniform $FTRA_\infty$ can be improved to $O(n^4)$ with a slightly different heuristic. In this heuristic, for PD-2's Event 2, ToC is calculated as $\min(NC, NS)$ instead since the event only causes two cases (cases (1) and (2) in Lemma 6.4.13). In addition, the algorithm APD-4 computes the same solution as PD-4, so we have the following theorem.

Theorem 6.4.15. *The algorithm APD-4 is 1.61-, (1.11, 1.78)- and (1,2)-approximation in time $O(n^4)$ for the uniform $FTRA$.*

In order to further achieve the factor of 1.52 in strongly polynomial time that matches the best result [132] for the uniform $FTFL$, it is necessary to apply the cost scaling and greedy augmentation (GA) techniques [132, 67] for $FTFL$ to $FTRA$. However, like in [94, 148], the difficulty encountered is the application of greedy augmentation (GA) in polynomial time, since the naive way of treating an $FTRA/FTRA_\infty$ instance as an equivalent $FTFL$ instance and then directly applying GA after cost scaling will result in weakly

polynomial or pseudo-polynomial time algorithms, depending on whether using the instance shrinking technique in the previous section or not.

Algorithm 6.4 AGA: Acceleration of Greedy Augmentation

Input: $\mathcal{F}$, $\mathcal{C}$, $\boldsymbol{f}$, $\boldsymbol{c}$, $\boldsymbol{r}$, $\boldsymbol{R}$, $(\boldsymbol{x}, \boldsymbol{y})$. **Output:** $(\boldsymbol{x}, \boldsymbol{y})$.
Initialization:
for $j \in \mathcal{C}$ //optimize the total connection cost first
  for $i \in \mathcal{F}$ and $y_i > 0$, in the increasing order of distances w.r.t j
    $x_{ij} \leftarrow \min(r_j, y_i)$
    $r_j \leftarrow r_j - x_{ij}$
set residual vector $\bar{\boldsymbol{y}} \leftarrow \boldsymbol{R} - \boldsymbol{y}$ //for detecting the case y_i reaches R_i
set $CC \leftarrow \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij}x_{ij}$ as the current total connection cost
invoke calculateGain

while $\max_{i\in\mathcal{F}} gain(i) > 0$: //if $gain(i) > 0$, then $\bar{y}_i > 0$ from the calculateGain function
  pick $i^* = \arg\max_{i\in\mathcal{F}} \frac{gain(i)}{f_i}$
  $\mathcal{S}_i \leftarrow \left\{ j \in \mathcal{C} \mid \max_{i'\in\mathcal{F}\&\&x_{i'j}>0} c_{i'j} - c_{i^*j} > 0 \right\}$
  $\forall j \in \mathcal{S}_i : i_j^* \leftarrow \arg\max_{i'\in\mathcal{F}\&\&x_{i'j}>0} c_{i'j}$
  $NS \leftarrow \min_{j\in\mathcal{S}_i} x_{i_j^*j}$, $ToC \leftarrow \min(NS, \bar{y}_i)$
  set $y_{i^*} \leftarrow y_{i^*} + ToC$
  $\Delta \leftarrow 0$ //Δ stores the total connection cost decrease after all switches
  for $j \in \mathcal{S}_i$
    $\Delta \leftarrow \Delta + ToC \cdot \left(\max_{i'\in\mathcal{F}\&\&x_{i'j}>0} c_{i'j} - c_{i^*j} \right)$
    set $x_{i_j^*j} \leftarrow x_{i_j^*j} - ToC$ and $x_{i^*j} \leftarrow x_{i^*j} + ToC$
  set $CC \leftarrow CC - \Delta$
  update $\bar{\boldsymbol{y}}$
  invoke calculateGain

function calculateGain
  for $i \in \mathcal{F}$
    $C_i \leftarrow CC$ //C_i stores the total connection cost of i after connections are switched to i
    $gain(i) \leftarrow 0$
    if $\bar{y}_i > 0$
      for $j \in \mathcal{C}$
        if $\max_{i'\in\mathcal{F}\&\&x_{i'j}>0} c_{i'j} > c_{ij}$
          $C_i \leftarrow C_i - \max_{i'\in\mathcal{F}\&\&x_{i'j}>0} c_{i'j} + c_{ij}$
      $gain(i) \leftarrow CC - C_i - f_i$

Nevertheless, if GA is applied with another similar acceleration heuristic, it changes to the algorithm AGA (Algorithm 6.4) which runs in strongly polynomial time. Before describing AGA, we take a brief look at GA in [67] for $FTFL$. It defines $gain(i)$ of a facility i to be the decrease in total cost (the decrease in the total connection cost minus the increase in the facility cost of

i) of the solution after adding a facility i to open and connecting clients to their closest facilities. Note that once a set of open facilities are fixed, the total connection cost can be easily computed since every client simply chooses these facilities in increasing order of distance. GA then iteratively picks the facility with the largest gain ratio $\frac{gain(i)}{f_i}$ to open until there is no facility i with $gain\,(i) > 0$ left. On the other hand, AGA computes $gain\,(i)$ in the same way as GA. The difference is in $FTRA$ there are $\sum_{i\in\mathcal{F}} R_i$ facilities in total, therefore it is slow to consider one facility at a time (in each iteration of AGA). Fortunately, there is also an acceleration heuristic: because all facilities at a site i has $gain\,(i)$, once a facility at site i_m with $\max_i \frac{gain(i)}{f_i}$ is selected to open, additional facilities at i_m may also open at the same time until either (1) this maximum $gain\,(i_m)$ reduces due to insufficient decrease in the total connection cost; or (2) y_i reaches R_i. Moreover, (1) happens once a client has appeared to switch all of its most expensive connections to i_m, which is similar to the switching case in the previous algorithm APD-4.

Formally in the AGA algorithm, CC denotes the current total connection cost and C_i the connection cost after i is opened and client connections are switched. The calculateGain function computes $gain\,(i)$ and the while loop implements GA with the described heuristic. In each loop iteration, for updating CC, Δ stores the total decrease in the connection cost after client switching. Following the heuristic, ToC and NS are defined similarly as in the APD-4 algorithm. Note that in the initialization phase of AGA, the total connection cost is optimized first so that every client connects to its closest facilities. This is to ensure that in every iteration only the client connections with the largest costs need to be considered in computing the best possible connection cost C_i.

Lemma 6.4.16. *The algorithm AGA runs in $O\,(n^4)$ for $FTRA$.*

Proof. Each iteration of the while loop runs in $O\,(n_c n_f)$ due to the calculateGain function. Now, we bound the the total number of iterations. Similar to the acceleration heuristic analysis of the algorithm APD-4 (c.f. Lemma 6.4.13), in AGA once a site i_m with the maximum gain is chosen, AGA opens the facilities at i_m until either R_{i_m} is reached, or a client has appeared to switch all of its most expensive connections, causing reduced maximum gain. Further, there are at most n_f chances to reach R_{i_m} and $n_c n_f$ possible sets of connections for all clients. Since clients also only switch to cheaper connections, there are maximum $(n_f + n_c n_f)$ iterations. The total time is therefore $O\left(n_c^2 n_f^2\right)$. □

Now the important observation/trick for the analysis is that applying AGA to an $FTRA/FTRA_\infty$ instance[2] (with solution) obtains essentially the *same solution* (also the same cost) as treating this instance as an equivalent $FTFL$ instance (by naively splitting sites) and then directly applying GA. The difference is, with the acceleration heuristic, AGA is able to arrive at this solution faster, in strongly polynomial time. The observation then implies that AGA alone improves the 3.16-approximation result of [148] for the general $FTRA_\infty$ to 2.408 in polynomial time using the GA results [67] for $FTFL$. Moreover, the fast AGA algorithm implies that the PD-2 algorithm with cost scaling and greedy augmentation for the uniform $FTRA_\infty$ (see Chapter 4) can be further improved to 1.52-approximation in runtime $O(n^4)$. This result also holds for the uniform $FTRA$ since we have proved that the algorithm APD-4 is (1.11, 1.78)-approximation.

Theorem 6.4.17. *The uniform FTRA can be approximated with a factor of 1.52 in time $O(n^4)$.*

6.5 The uniform k-$FTRA$

Lastly, we consider the Constrained Fault-Tolerant k-Resource Allocation (k-$FTRA$) problem and show its uniform case achieves an approximation ratio of 4. In this important variant of $FTRA$, there is an additional constraint that at most k facilities ($\max_{j\in\mathcal{C}} r_j \le k$ and $k \le \sum_{i\in\mathcal{F}} R_i$) across all sites can be opened as resources. This problem has the following formulation:

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}} \sum_{j\in\mathcal{C}} c_{ij} x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \ge r_j \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \ge 0 \\ & \sum_{i\in\mathcal{F}} y_i \le k \\ & \forall i \in \mathcal{F} : y_i \le R_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij},\, y_i \in \mathbb{Z}^+. \end{array} \tag{6.4}$$

Its LP-relaxation (primal LP) and dual LP are:

[2]The heuristic TOC in AGA for $FTRA_\infty$ can be changed to $TOC = NS$ since there is no constraint R_i at each site i.

$$\begin{array}{ll} \text{minimize} & \sum_{i\in\mathcal{F}} f_i y_i + \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij} x_{ij} \\ \text{subject to} & \forall j \in \mathcal{C} : \sum_{i\in\mathcal{F}} x_{ij} \geq r_j \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : y_i - x_{ij} \geq 0 \\ & \sum_{i\in\mathcal{F}} y_i \leq k \\ & \forall i \in \mathcal{F} : y_i \leq R_i \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : x_{ij}, y_i \geq 0 \end{array} \tag{6.5}$$

$$\begin{array}{ll} \text{maximize} & \sum_{j\in\mathcal{C}} r_j \alpha_j - \sum_{i\in\mathcal{F}} R_i z_i - k\theta \\ \text{subject to} & \forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} \beta_{ij} \leq f_i + z_i + \theta \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j - \beta_{ij} \leq c_{ij} \\ & \forall i \in \mathcal{F}, j \in \mathcal{C} : \alpha_j, \beta_{ij}, z_i, \theta \geq 0. \end{array} \tag{6.6}$$

It is clear that k-$FTRA$ generalizes the well studied k-UFL [77, 73] and k-$FTFL$ [132] problems. In the following, besides adapting the algorithms and analyses therein, we also develop a greedy pairing (GP) procedure which in polynomial time constructs paired and unpaired sets of facilities from sites for randomly opening them afterwards.

Algorithm description The algorithm PK (Algorithm 6.5) consists of three sequential procedures: Binary Search (BS), Greedy Pairing (GP) and Randomized Rounding (RR). BS utilizes the previous (1, 2)-approximation algorithm APD-4 for $FTRA$ with the *modified* input facility cost $2\,(f_i + \theta)$, i.e., the cost is increased by θ first and then scaled by 2. As we will see later in the analysis, this modification is necessary for two reasons: 1) the Lagrangian relaxation of k-$FTRA$ is $FTRA$; 2) the scaling of the facility cost enables us to build on the approximation ratio (1, 2) of $FTRA$ for getting the ratio of k-$FTRA$. For simplicity, let APD-4(θ, λ) denote the parameterized APD-4 algorithm with the input facility cost perturbing factor θ and scaling factor λ, so APD-4(0, 1) produces the same solution as APD-4. From LP (6.1) and (6.4), it is clear that APD-4 produces an almost feasible integral solution to k-$FTRA$ except that it has to guarantee at most k facilities to open ($\sum_{i\in\mathcal{F}} y_i \leq k$) from all sites. This guarantee might not be even possible, but fortunately we can use APD-4(θ, λ) to get two solutions ($\boldsymbol{x_s}$, $\boldsymbol{y_s}$) and ($\boldsymbol{x_l}$, $\boldsymbol{y_l}$) with the small one having $\sum_{i\in\mathcal{F}} y_{s,i} = k_s < k$ and the large one $\sum_{i\in\mathcal{F}} y_{l,i} = k_l > k$ facilities open. A convex combination of these two solutions is able to give a feasible *fractional* solution ($\boldsymbol{x'}$, $\boldsymbol{y'}$) to k-$FTRA$ instead, i.e. $(\boldsymbol{x'}, \boldsymbol{y'}) = a\,(\boldsymbol{x_s}, \boldsymbol{y_s}) + b\,(\boldsymbol{x_l}, \boldsymbol{y_l})$

with $a + b = 1$ and $ak_s + bk_l = k$. The solutions can be obtained by binary searching two values (θ_1 and θ_2) of θ over the interval $\left[0, \frac{n_c c_{max}}{\lambda}\right]$ where $c_{max} = \max_{i \in \mathcal{F},\, j \in \mathcal{C}} c_{ij}$ and invoking APD-4(θ_1, λ) and APD-4(θ_2, λ). This specific interval is chosen because as the value of θ increases, the number of open facilities from APD-4(θ, λ) will decrease. So if $\theta = \frac{n_c c_{max}}{\lambda}$, the algorithm will only open the minimum number of $\max_{j \in \mathcal{C}} r_j$ facilities.[3] Moreover, as shown later, if θ_1 and θ_2 become sufficiently close ($\epsilon = \frac{c_{\min}}{8N^2}$ where $c_{\min}$ is the smallest positive connection cost and $N = \sum_{i \in \mathcal{F}} R_i$ [4]) in BS, the approximation ratio of APD-4 is almost preserved for building a ratio for k-$FTRA$.

However, for a feasible *integral* solution ($\boldsymbol{x}$, $\boldsymbol{y}$) with k open facilities, the algorithm instead relies on our *efficient* GP and RR procedures. These procedures extend the matching and rounding procedures (cf. the paragraph before Lemma 7.1 in [132]) for k-$FTFL$ respectively. In particular, based on the solution vectors $\boldsymbol{y_s}$ and $\boldsymbol{y_l}$ obtained from BS, GP splits the vector $\boldsymbol{y_l}$ into $\boldsymbol{y_p}$ and $\bar{\boldsymbol{y}}_{\boldsymbol{p}}$ s.t. $\boldsymbol{y_l} = \boldsymbol{y_p} + \bar{\boldsymbol{y}}_{\boldsymbol{p}}$ and $\sum_{i \in \mathcal{F}} y_{s,i} = \sum_{i \in \mathcal{F}} y_{p,i} = k_s$. Note that each of these integral vectors represents the facility opening amounts of all sites. To be precise, GP greedily constructs the paired ($\boldsymbol{y_p}$) and unpaired facilities ($\bar{\boldsymbol{y}}_{\boldsymbol{p}}$) from $\boldsymbol{y_l}$ against the small solution $\boldsymbol{y_s}$. It first pairs the facilities of the corresponding sites in $\boldsymbol{y_s}$ and $\boldsymbol{y_l}$ (both sites with open facilities) and records the pairing result in $\boldsymbol{y_p}$. Next, for each left unpaired site i in $\hat{\boldsymbol{y}}_{\boldsymbol{s}}$ in arbitrary order, GP exhaustively pairs the facilities at i with the facilities of the unpaired sites in $\hat{\boldsymbol{y}}_{\boldsymbol{l}}$ in the order of closest to i. In this pairing step, $\boldsymbol{y_p}$ is updated accordingly. At the end, $\bar{\boldsymbol{y}}_{\boldsymbol{p}}$ is simply set to be $\boldsymbol{y_l} - \boldsymbol{y_p}$. To be more precise, we consider a simple example with $\boldsymbol{y_s} = [3, 2, 0, 2]$ and $\boldsymbol{y_l} = [2, 0, 5, 3]$ from BS before running GP. After the first pairing step, $\boldsymbol{y_p} = [2, 0, 0, 2]$, $\hat{\boldsymbol{y}}_{\boldsymbol{s}} = [1, 2, 0, 0]$ and $\hat{\boldsymbol{y}}_{\boldsymbol{l}} = [0, 0, 5, 1]$. Now for simplicity, we assume that the distance between sites i and j is $|i - j|$ where we follow the ascending order of indices j's in resolving the ties of the closest distances. Therefore, after the second step, $\boldsymbol{y_p} = [2, 0, 3, 2]$, $\hat{\boldsymbol{y}}_{\boldsymbol{s}} = [0, 0, 0, 0]$ and $\hat{\boldsymbol{y}}_{\boldsymbol{l}} = \bar{\boldsymbol{y}}_{\boldsymbol{p}} = [0, 0, 2, 1]$, since both the unpaired $\hat{y_{s,1}}$ and $\hat{y_{s,2}}$ (1-based index) are paired to the closest unpaired $\hat{y_{l,3}}$.

From the $\boldsymbol{y_s}$, $\boldsymbol{y_p}$ and $\bar{\boldsymbol{y}}_{\boldsymbol{p}}$ obtained, the last procedure RR then randomly opens k facilities in a way that the expected facility opening cost of $\boldsymbol{y}$ is the *same* as the facility opening cost of the convex combination solution $\boldsymbol{y'}$. In

[3] We noticed that the binary search interval $[0, nrc_{max}]$ (c.f. the third paragraph of Section 7 of [132]) for k-$FTFL$ can be reduced to $\left[0, \frac{n_c c_{max}}{2}\right]$, because once the minimum number of $\max_{j \in \mathcal{C}} r_j$ facilities are opened and all facility costs are at least $n_c c_{max}$, from the primal-dual algorithm, all clients are already fully-connected.

[4] The algorithm analysis also holds if $N = \sum_{j \in \mathcal{C}} r_j$.

Algorithm 6.5 PK: Procedures for *k-FTRA*

Input: a *k-FTRA* instance $\langle \mathcal{F}, \mathcal{C}, f_i, c_{ij}, r_j, R_i, k \rangle$. **Output:** $(\boldsymbol{x}, \boldsymbol{y})$
Initialization: $\boldsymbol{x} \leftarrow \boldsymbol{0}$, $\boldsymbol{y} \leftarrow \boldsymbol{0}$
Procedure 1: Binary Search (BS)
$\theta_1 = 0$, $\theta_2 = \frac{n_c \max_{i \in \mathcal{F}, j \in \mathcal{C}} c_{ij}}{2}$
while $\theta_2 - \theta_1 > \epsilon$ **do**:
    $mid = \frac{\theta_2 - \theta_1}{2}$
    invoke APD-4 with $\langle \mathcal{F}, \mathcal{C}, 2(f_i + mid), c_{ij}, r_j, R_i \rangle$ and output $(\boldsymbol{x_{mid}}, \boldsymbol{y_{mid}})$; **set** $k_{mid} = \sum_{i \in \mathcal{F}} y_{mid,i}$
    if $k_{mid} < k$
        set $\theta_2 = mid$
    else if $k_{mid} > k$
        set $\theta_1 = mid$
    else
        return mid //if reached, all procedures afterwards can be ignored
invoke APD-4 with $\langle \mathcal{F}, \mathcal{C}, 2(f_i + \theta_1), c_{ij}, r_j, R_i \rangle$ and output $(\boldsymbol{x_l}, \boldsymbol{y_l})$
invoke APD-4 with $\langle \mathcal{F}, \mathcal{C}, 2(f_i + \theta_2), c_{ij}, r_j, R_i \rangle$ and output $(\boldsymbol{x_s}, \boldsymbol{y_s})$
$k_l = \sum_{i \in \mathcal{F}} y_{l,i} > k$ and $k_s = \sum_{i \in \mathcal{F}} y_{s,i} < k$
Procedure 2: Greedy Pairing (GP)
//vectors store the numbers of paired and unpaired facilities in $\boldsymbol{y_l}$
set $\boldsymbol{y_p} \leftarrow \boldsymbol{0}$, $\bar{\boldsymbol{y}}_{\boldsymbol{p}} \leftarrow \boldsymbol{0}$
for $i \in \mathcal{F}$
    if $y_{s,i} > 0$ and $y_{l,i} > 0$
        $y_{p,i} \leftarrow y_{p,i} + \min(y_{s,i}, y_{l,i})$
        //update vectors and store in $\hat{\boldsymbol{y}}_{\boldsymbol{s}}$ and $\hat{\boldsymbol{y}}_{\boldsymbol{l}}$ for the next pairing steps
        $\hat{y_{s,i}} \leftarrow y_{s,i} - \min(y_{s,i}, y_{l,i})$
        $\hat{y_{l,i}} \leftarrow y_{l,i} - \min(y_{s,i}, y_{l,i})$
for $i \in \mathcal{F}$ in arbitrary order
    if $\hat{y_{s,i}} > 0$
        for $i' \in \mathcal{F} \backslash i$ in the order of closest to i
            if $\hat{y_{l,i'}} > 0$
                $y_{p,i'} \leftarrow y_{p,i'} + \min(\hat{y_{s,i}}, \hat{y_{l,i'}})$
                $\hat{y_{s,i}} \leftarrow \hat{y_{s,i}} - \min(\hat{y_{s,i}}, \hat{y_{l,i'}})$
                $\hat{y_{l,i'}} \leftarrow \hat{y_{l,i'}} - \min(\hat{y_{s,i}}, \hat{y_{l,i'}})$
$\bar{\boldsymbol{y}}_{\boldsymbol{p}} \leftarrow \boldsymbol{y_l} - \boldsymbol{y_p}$ //at this time $\sum_{i \in \mathcal{F}} y_{p,i} = k_s$ and $\sum_{i \in \mathcal{F}} \bar{y}_{p,i} = k_l - k_s$
Procedure 3: Randomized Rounding (RR)
choose probabilities $a = \frac{k_l - k}{k_l - k_s}$ and $b = \frac{k - k_s}{k_l - k_s}$ so $ak_s + bk_l = k$ and $a + b = 1$
set $\boldsymbol{y} \leftarrow \boldsymbol{y_s}$ with probability a and $\boldsymbol{y} \leftarrow \boldsymbol{y_p}$ with probability $b = 1 - a$ //disjoint cases both open k_s facilities
select a random subset of $k - k_s$ facilities to open from $\bar{\boldsymbol{y}}_{\boldsymbol{p}}$ and add these to $\boldsymbol{y}$
//at this time $\sum_{i \in \mathcal{F}} y_i = k$ and each facility in $\boldsymbol{y_l}$ is opened with probability b
//connects each client j to its closest r_j opened facilities
for $j \in \mathcal{C}$
    for $i \in \mathcal{F}$ in the order of closest to j
        $x_{ij} \leftarrow \min(r_j, y_i)$
        $r_j \leftarrow r_j - x_{ij}$

addition, RR connects each client j to its closest r_j open facilities in $\boldsymbol{y}$, ensuring the expected connection cost of $\boldsymbol{x}$ is *bounded* by the connection cost of $\boldsymbol{x'}$.

Algorithm analysis The basic idea of the analysis is to first bound $cost\,(\boldsymbol{x'},\,\boldsymbol{y'})$ by $cost\,(\boldsymbol{\alpha'},\,\boldsymbol{\beta'},\,\boldsymbol{z'},\theta')$ where $(\boldsymbol{\alpha'},\,\boldsymbol{\beta'},\,\boldsymbol{z'},\theta')$ is a constructed feasible dual solution to LP (6.6). Then we bound the expected total cost $cost\,(\boldsymbol{x},\,\boldsymbol{y})$ with $cost\,(\boldsymbol{x'},\,\boldsymbol{y'})$ to further establish the approximation ratio ρ s.t. $cost\,(\boldsymbol{x},\,\boldsymbol{y}) \leq \rho cost\,(\boldsymbol{\alpha'},\,\boldsymbol{\beta'},\,\boldsymbol{z'},\theta')$. Finally, by the weak duality theorem, $cost\,(\boldsymbol{x},\,\boldsymbol{y}) \leq \rho cost\,(\boldsymbol{x_k^*},\,\boldsymbol{y_k^*})$ where $(\boldsymbol{x_k^*},\,\boldsymbol{y_k^*})$ is the optimal fractional solution to k-$FTRA$ (displayed as LP (6.5)).

In the first step, we focus on analyzing the BS procedure to bound $cost\,(\boldsymbol{x'},\,\boldsymbol{y'})$ by $cost\,(\boldsymbol{\alpha'},\,\boldsymbol{\beta'},\,\boldsymbol{z'},\theta')$. Suppose APD-4$(\theta,\,2)$ produces the primal solution $(\tilde{\boldsymbol{x}},\,\tilde{\boldsymbol{y}})$ with $\tilde{k}$ open facilities. We let the cost of $(\tilde{\boldsymbol{x}},\,\tilde{\boldsymbol{y}})$ w.r.t. the *original* input instance be $cost\,(\tilde{\boldsymbol{x}},\,\tilde{\boldsymbol{y}}) = \tilde{F} + \tilde{C}$, where in the separate costs $\left(\tilde{F},\,\tilde{C}\right)$, $\tilde{F} = \sum_{i\in\mathcal{F}} f_i\tilde{y}_i$ is the total facility cost and $\tilde{C} = \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}\tilde{x_{ij}}$ is the connection cost. Similarly, w.r.t. the modified instance, the cost is $cost'\,(\tilde{\boldsymbol{x}},\,\tilde{\boldsymbol{y}}) = 2\left(\tilde{F} + \tilde{k}\theta\right) + \tilde{C}$. From the analysis (cf. the paragraph before Theorem 6.4.11) of the factor revealing program of the PD-4 algorithm, for APD-4$(\theta,\,2)$, we get $\forall i \in \mathcal{F}:\ \sum_{j\in\mathcal{C}'}(u_{ij} - 2c_{ij}) \leq 2\,(f_i + \theta)$ where $\mathcal{C}' \subseteq \mathcal{C}$, i.e.,

$$\sum_{j\in\mathcal{C}} \tilde{\alpha_j} - 2\sum_{j\in\mathcal{C}} c_{ij} - \tilde{z_i} - 2\,(f_i + \theta) \leq 0 \tag{6.7}$$

where $\left(\tilde{\boldsymbol{\alpha}},\,\tilde{\boldsymbol{\beta}},\,\tilde{\boldsymbol{z}}\right)$ is the corresponding constructed dual values of $(\tilde{\boldsymbol{x}},\,\tilde{\boldsymbol{y}})$ from the PD-4 algorithm. Further, from Lemma 6.4.3, we have a bound for $cost'\,(\tilde{\boldsymbol{x}},\,\tilde{\boldsymbol{y}})$, i.e.,

$$2\left(\tilde{F} + \tilde{k}\theta\right) + \tilde{C} \leq \sum_{j\in\mathcal{C}} r_j\tilde{\alpha_j} - \sum_{i\in\mathcal{F}} R_i\tilde{z_i}. \tag{6.8}$$

Note that the dual solution $\left(\tilde{\boldsymbol{\alpha}},\,\tilde{\boldsymbol{\beta}},\,\tilde{\boldsymbol{z}}\right)$ is used only in the analysis. Also, because APD-4 only speeds up PD-4 by combining its events, we can use the dual solution produced from PD-4 for analyzing APD-4. If we set $\left(\tilde{\boldsymbol{\alpha'}},\,\tilde{\boldsymbol{z'}},\,\tilde{\theta'}\right) = \left(\frac{\tilde{\boldsymbol{\alpha}}}{2},\,\frac{\tilde{\boldsymbol{z}}}{2},\,\theta\right)$ and $\forall i \in \mathcal{F},\, j \in \mathcal{C}:\ \tilde{\beta'_{ij}} = \tilde{\alpha'_j} - c_{ij}$, the inequality (6.7) then becomes $\sum_{j\in\mathcal{C}} \tilde{\beta'_{ij}} \leq f_i + \tilde{z'_i} + \tilde{\theta'}$, implying $\left(\tilde{\boldsymbol{\alpha'}},\,\tilde{\boldsymbol{\beta'}},\,\tilde{\boldsymbol{z'}},\tilde{\theta'}\right)$ is a feasible dual solution to LP (6.6). Furthermore, (6.8) becomes

$$2\tilde{F} + \tilde{C} \leq 2\left(\sum_{j\in\mathcal{C}} r_j\tilde{\alpha'_j} - \sum_{i\in\mathcal{F}} R_i\tilde{z'_i} - \tilde{k}\tilde{\theta'}\right). \tag{6.9}$$

The analysis here reveals the Lagrangian relation between k-$FTRA$ and $FTRA$ from the dual perspective, whereas the Lagrangian relaxation framework (cf. Section 3.6 of [77]) starts from the primal. Therefore, if $\tilde{k} = k$, $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ is 2-approximation from the inequality (6.9), the bound $cost\,(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}}) < 2\tilde{F} + \tilde{C}$ and the feasibilities of $(\tilde{\boldsymbol{x}}, \tilde{\boldsymbol{y}})$ and $\left(\tilde{\boldsymbol{\alpha}'}, \tilde{\boldsymbol{\beta}'}, \tilde{\boldsymbol{z}'}, \tilde{\theta'}\right)$. However, as mentioned before, we may never encounter the situation $\tilde{k} = k$. Instead, the BS procedure finds θ_1 and θ_2 until $\theta_2 - \theta_1 \leq \epsilon = \frac{c_{min}}{8N^2}$. It then runs APD-4($\theta_1$, 2) to obtain the solution $(\boldsymbol{x_l}, \boldsymbol{y_l})$ with $k_l > k$ and the cost (F_l, C_l) w.r.t. the original instance; and APD-4(θ_2, 2) to get the solution $(\boldsymbol{x_s}, \boldsymbol{y_s})$ with $k_s < k$ and (F_s, C_s). Hence, from (6.9) we have

$$2F_l + C_l \leq 2\left(\sum_{j\in\mathcal{C}} r_j\alpha'_{l,j} - \sum_{i\in\mathcal{F}} R_i z'_{l,i} - k_l\theta_1\right) \tag{6.10}$$

and

$$2F_s + C_s \leq 2\left(\sum_{j\in\mathcal{C}} r_j\alpha'_{s,j} - \sum_{i\in\mathcal{F}} R_i z'_{s,i} - k_s\theta_2\right) \tag{6.11}$$

where $(\boldsymbol{\alpha'_l}, \boldsymbol{\beta'_l}, \boldsymbol{z'_l})$ and $(\boldsymbol{\alpha'_s}, \boldsymbol{\beta'_s}, \boldsymbol{z'_s})$ are constructed in the same way as $\left(\tilde{\boldsymbol{\alpha}'}, \tilde{\boldsymbol{\beta}'}, \tilde{\boldsymbol{z}'}\right)$ to be feasible duals.

Now we are ready to bound $cost\,(\boldsymbol{x'}, \boldsymbol{y'})$ by $cost\,(\boldsymbol{\alpha'}, \boldsymbol{\beta'}, \boldsymbol{z'}, \theta')$. The proof of the following lemma builds on the idea of Lemma 9 in [77] for k-median.

Lemma 6.5.1. *$cost\,(\boldsymbol{x'}, \boldsymbol{y'}) < \left(2 + \frac{1}{N}\right) F' + C' \leq \left(2 + \frac{1}{N}\right) cost\,(\boldsymbol{\alpha'}, \boldsymbol{\beta'}, \boldsymbol{z'}, \theta')$, where $N = \sum_{i\in\mathcal{F}} R_i$, $(\boldsymbol{x'}, \boldsymbol{y'}) = a\,(\boldsymbol{x_s}, \boldsymbol{y_s}) + b\,(\boldsymbol{x_l}, \boldsymbol{y_l})$, $a + b = 1$, $k = ak_s + bk_l$, $F' = \sum_{i\in\mathcal{F}} f_i y'_i = aF_s + bF_l$, $C' = \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x'_{ij} = aC_s + bC_l$, $\boldsymbol{\alpha'} = a\boldsymbol{\alpha'_s} + b\boldsymbol{\alpha'_l}$, $\boldsymbol{\beta'} = a\boldsymbol{\beta'_s} + b\boldsymbol{\beta'_l}$, $\boldsymbol{z'} = a\boldsymbol{z'_s} + b\boldsymbol{z'_l}$ and $\theta' = \theta_2$. Moreover, $(\boldsymbol{\alpha'}, \boldsymbol{\beta'}, \boldsymbol{z'}, \theta')$ is a feasible dual solution to the k-$FTRA$ problem.*

Proof. From the constructions of $(\boldsymbol{\alpha'_l}, \boldsymbol{\beta'_l}, \boldsymbol{z'_l})$ and $(\boldsymbol{\alpha'_s}, \boldsymbol{\beta'_s}, \boldsymbol{z'_s})$, we get $\forall i \in \mathcal{F} : \sum_{j\in\mathcal{C}} \beta'_{s,ij} \leq f_i + z'_{s,i} + \theta_2$ and $\sum_{j\in\mathcal{C}} \beta'_{l,ij} \leq f_i + z'_{l,i} + \theta_1$, then $\sum_{j\in\mathcal{C}} \beta'_{ij} \leq f_i + z'_i + a\theta_2 + b\theta_1 \leq f_i + z'_i + \theta_2$ after multiplying the first inequality by a, the second by b and adding them together. In addition, with the setting $\forall i \in \mathcal{F}, j \in \mathcal{C} : \beta'_{ij} = \alpha'_j - c_{ij}$, we get the feasibility of $(\boldsymbol{\alpha'}, \boldsymbol{\beta'}, \boldsymbol{z'}, \theta')$ to LP (6.6). Next, we aim to derive the following bound

$$\left(2 + \frac{1}{N}\right) F_l + C_l \leq \left(2 + \frac{1}{N}\right)\left(\sum_{j\in\mathcal{C}} r_j\alpha'_{l,j} - \sum_{i\in\mathcal{F}} R_i z'_{l,i} - k_l\theta_2\right) \tag{6.12}$$

from the inequality (6.10). For now, suppose this bound holds, from (6.11), we have

$$\left(2+\frac{1}{N}\right)F_s + C_s \leq \left(2+\frac{1}{N}\right)\left(\sum_{j\in\mathcal{C}} r_j\alpha'_{s,j} - \sum_{i\in\mathcal{F}} R_i z'_{s,i} - k_s\theta_2\right). \quad (6.13)$$

After multiplying (6.12) by b, (6.13) by a and adding them together, we get

$$\left(2+\frac{1}{N}\right)F' + C' \leq \left(2+\frac{1}{N}\right)\left(\sum_{j\in\mathcal{C}} r_j\alpha'_j - \sum_{i\in\mathcal{F}} R_i z'_i - k\theta_2\right). \quad (6.14)$$

This then yields the lemma together with the feasibility of $(\boldsymbol{\alpha'}, \boldsymbol{\beta'}, \boldsymbol{z'}, \theta')$ and $cost\,(\boldsymbol{x'}, \boldsymbol{y'}) = F' + C'$. The last thing left is to verify in the following that (6.12) indeed holds from the inequality (6.10), the termination condition of the algorithm $\theta_2 - \theta_1 \leq \epsilon = \frac{c_{min}}{8N^2}$ and the fact that $C_l = \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x_{l,ij} \geq c_{min}$.

$$\begin{aligned} C_l &\leq 2\left(\sum_{j\in\mathcal{C}} r_j\alpha'_{l,j} - \sum_{i\in\mathcal{F}} R_i z'_{l,i} - k_l\theta_1 - F_l\right) \\ &\leq 2\left(\sum_{j\in\mathcal{C}} r_j\alpha'_{l,j} - \sum_{i\in\mathcal{F}} R_i z'_{l,i} - k_l\theta_2 - F_l\right) + \frac{c_{min}k_l}{4N^2} \\ &\leq 2\left(\sum_{j\in\mathcal{C}} r_j\alpha'_{l,j} - \sum_{i\in\mathcal{F}} R_i z'_{l,i} - k_l\theta_2 - F_l\right) + \frac{C_l k_l}{4N^2}. \end{aligned}$$

For simplicity, let $\triangle = \left(\sum_{j\in\mathcal{C}} r_j\alpha'_{l,j} - \sum_{i\in\mathcal{F}} R_i z'_{l,i} - k_l\theta_2 - F_l\right)$. Because $k_l \leq N$ and $N \geq 1$, we get $C_l \leq \frac{2}{1-\frac{k_l}{4N^2}}\triangle \leq \left(2+\frac{1}{N}\right)\triangle$. Hence, the inequality (6.12) is verified. □

For runtime, our BS procedure totally makes $O\left(L + \log N + \log n\right)$ probes (L is the number of bits of the input costs) over the interval $\left[0, \frac{n_c c_{max}}{2}\right]$ until the interval becomes the size of $\frac{c_{min}}{8N^2}$. Moreover, each probe takes $O\left(n^4\right)$ to invoke the APD-4 algorithm, so the total time is $O\left(n^4\left(L + \log N + \log n\right)\right)$ which dominates the overall runtime of the algorithm PK.

In the next step, we focus on analyzing the GP and RR procedures to bound $cost\,(\boldsymbol{x}, \boldsymbol{y}) = F + C$ with $cost\,(\boldsymbol{x'}, \boldsymbol{y'}) = F' + C'$ where we denote F and C as the expected total facility and connection costs respectively from the randomized procedure RR. This procedure can also be derandomized using the

method of conditional expectation as in [77] for the k-median problem. In the following, we bound F with F' and C with C' separately. With probability 1, RR opens exactly k facilities. Specifically, it randomly opens each facility in $\boldsymbol{y_p}$ with probability b, and each facility in $\bar{\boldsymbol{y}}_{\boldsymbol{p}}$ with probability $\frac{k-k_s}{k_l-k_s}$ which is also b. Since GP properly splits the vector $\boldsymbol{y_l}$ into $\boldsymbol{y_p}$ and $\bar{\boldsymbol{y}}_{\boldsymbol{p}}$ s.t. $\boldsymbol{y_l} = \boldsymbol{y_p} + \bar{\boldsymbol{y}}_{\boldsymbol{p}}$, we can conclude that each facility in $\boldsymbol{y_l}$ is opened with probability b. In addition, RR randomly opens each facility in $\boldsymbol{y_s}$ with probability a, therefore the total expected opening cost is $aF_s + bF_l$ which is F'.

Lemma 6.5.2. *The total expected facility opening cost F satisfies $F = F'$.*

Now we bound C with C'. Suppose two $FTRA$ instances with the solutions $(\boldsymbol{x_s}, \boldsymbol{y_s})$ and $(\boldsymbol{x_l}, \boldsymbol{y_l})$ are produced from the BS procedure. Afterwards, for getting a feasible solution to k-$FTRA$ from these solutions, instead we consider a naive pseudo-polynomial time algorithm. The algorithm first treats the $FTRA$ instances with the solutions $(\boldsymbol{x_s}, \boldsymbol{y_s})$ and $(\boldsymbol{x_l}, \boldsymbol{y_l})$ as equivalent $FTFL$ instances (by naively splitting sites and keeping the clients unchanged) with the transformed solutions $(\check{\boldsymbol{x}}_{\boldsymbol{s}}, \check{\boldsymbol{y}}_{\boldsymbol{s}})$ and $(\check{\boldsymbol{x}}_{\boldsymbol{l}}, \check{\boldsymbol{y}}_{\boldsymbol{l}})$[5] respectively. Then, it uses the matching and rounding procedures [132] on $(\check{\boldsymbol{x}}_{\boldsymbol{s}}, \check{\boldsymbol{y}}_{\boldsymbol{s}})$ and $(\check{\boldsymbol{x}}_{\boldsymbol{l}}, \check{\boldsymbol{y}}_{\boldsymbol{l}})$ to get a feasible solution $(\check{\boldsymbol{x}}, \check{\boldsymbol{y}})$ to k-$FTFL$. Finally, the solution $(\check{\boldsymbol{x}}, \check{\boldsymbol{y}})$ can be easily transformed to a feasible solution $(\boldsymbol{x}, \boldsymbol{y})$ to k-$FTRA$. Now, the important observation is that directly applying GP and RR to these $FTRA$ instances with the solutions $(\boldsymbol{x_s}, \boldsymbol{y_s})$ and $(\boldsymbol{x_l}, \boldsymbol{y_l})$ from BS obtains essentially the *same solution* $(\boldsymbol{x}, \boldsymbol{y})$ (also the same cost) to k-$FTRA$ as the naive algorithm does. It is mainly because for the $FTRA$ instances of size $O(n)$, our designed GP procedure pairs the integer vectors in polynomial time. This is the acceleration of the matching procedure therein [132] applied to the equivalent $FTFL$ instances of size $O\left(\sum_{i\in\mathcal{F}} R_i\right)$. Therefore, *only in the analysis*, we can consider the naive algorithm instead to get the following bound for C. This analysis trick is similar to the trick used for analyzing the algorithm AGA (cf. the paragraph before Theorem 6.4.17).

Lemma 6.5.3. *The total expected connection cost $C \leq (1 + \max(a, b))\, C'$.*

Proof. For the equivalent $FTFL$ instances, we let $\mathcal{F}'$ be the set of split facilities with size $\sum_{i\in\mathcal{F}} R_i$ and use k to index these facilities. After the matching and rounding procedures in [132] on the transformed solutions $(\check{\boldsymbol{x}}_{\boldsymbol{s}}, \check{\boldsymbol{y}}_{\boldsymbol{s}})$ and $(\check{\boldsymbol{x}}_{\boldsymbol{l}}, \check{\boldsymbol{y}}_{\boldsymbol{l}})$, we get the solution $(\check{\boldsymbol{x}}, \check{\boldsymbol{y}})$ to k-$FTFL$. Also from its Lemma 7.2, we

[5] W.l.o.g., the solutions can be easily transformed between $FTRA$ and $FTFL$ as shown in Theorem 7 of [94].

can directly obtain the bound $C_j \leq (1+\max(a,b)) \sum_{k\in\mathcal{F}'} c_{kj} \left(a\check{x}_{s,kj} + b\check{x}_{l,kj}\right)$[6] where $C_j = \sum_{k\in\mathcal{F}'} c_{kj}\check{x}_{kj}$, i.e., the expected connection cost of any client j. Since $(\check{\boldsymbol{x}}_{\boldsymbol{s}}, \check{\boldsymbol{y}}_{\boldsymbol{s}})$, $(\check{\boldsymbol{x}}_{\boldsymbol{l}}, \check{\boldsymbol{y}}_{\boldsymbol{l}})$ and $(\boldsymbol{x}, \boldsymbol{y})$ are transformed from $(\boldsymbol{x}_{\boldsymbol{s}}, \boldsymbol{y}_{\boldsymbol{s}})$, $(\boldsymbol{x}_{\boldsymbol{l}}, \boldsymbol{y}_{\boldsymbol{l}})$ and $(\check{\boldsymbol{x}}, \check{\boldsymbol{y}})$ respectively with the same costs, we have

$$\begin{aligned} C &= \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}x_{ij} = \sum_{j\in\mathcal{C}}\sum_{k\in\mathcal{F}'} c_{kj}\check{x}_{kj} \\ &\leq (1+\max(a,b)) \sum_{j\in\mathcal{C}}\sum_{k\in\mathcal{F}'} c_{kj}\left(a\check{x}_{s,kj} + b\check{x}_{l,kj}\right) \\ &= (1+\max(a,b)) \sum_{i\in\mathcal{F}}\sum_{j\in\mathcal{C}} c_{ij}\left(ax_{s,ij} + bx_{l,ij}\right) \\ &= (1+\max(a,b))\, C', \end{aligned}$$

which concludes the lemma. □

Adding up the separate bounds in the previous lemmas, we get $cost(\boldsymbol{x}, \boldsymbol{y}) = F+C \leq F'+(1+\max(a,b))\,C'$. Relating this bound to the bound $\left(2+\frac{1}{N}\right)F'+C' \leq \left(2+\frac{1}{N}\right) cost(\boldsymbol{\alpha'}, \boldsymbol{\beta'}, \boldsymbol{z'}, \theta')$ in Lemma 6.5.1, we obtain

$$\begin{aligned} cost(\boldsymbol{x}, \boldsymbol{y}) &\leq F' + (1+\max(a,b))\,C' \\ &< \left(2+\frac{1}{N}\right)(1+\max(a,b))\,F' + (1+\max(a,b))\,C' \\ &\leq \left(2+\frac{1}{N}\right)(1+\max(a,b))\, cost(\boldsymbol{\alpha'}, \boldsymbol{\beta'}, \boldsymbol{z'}, \theta') \\ &< 4cost(\boldsymbol{\alpha'}, \boldsymbol{\beta'}, \boldsymbol{z'}, \theta'). \end{aligned}$$

The last inequality is from the fact that $a = \frac{k_l-k}{k_l-k_s} \leq 1-\frac{1}{N}$ (achieved when $k_l = N$ and $k_s = k-1$), and $b = \frac{k-k_s}{k_l-k_s} \leq 1-\frac{1}{k}$ (achieved when $k_l = k+1$ and $k_s = 1$). Therefore, $1+\max(a,b) \leq 2-\frac{1}{N}$ and $\left(2+\frac{1}{N}\right)(1+\max(a,b)) \leq 4-\frac{1}{N^2}$. By the weak duality theorem, the approximation ratio is 4. For runtime, from the algorithm PK, both GP and RR take $O(n^2)$.

Theorem 6.5.4. *The algorithm PK is 4-approximation for the uniform k-FTRA in polynomial time* $O(n^4(L+\log N+\log n))$.

[6]Note that from the proof of the lemma therein, we can easily get the bound coefficient $(1+\max(a,b))$ rather than 2.

6.6 Summary

In this chapter, we studied the $FTRA$ problem. For its general metric case, we first gave a unified LP-rounding algorithm achieving an approximation ratio of 4. Then we showed $FTRA$ reduces to $FTFL$ using an instance shrinking technique. By this reduction, we obtained the ratio of 1.7245 for the metric case and $O\left(\log^2 n\right)$ for the non-metric case. For the uniform $FTRA$ where all r_j's are the same, we provided a 1.52-approximation primal-dual algorithm in $O\left(n^4\right)$ time. The strongly polynomial runtime relies on our carefully designed acceleration heuristics. We also considered the k-$FTRA$ problem. For its uniform case, we gave the first constant-factor approximation algorithm with a factor of 4. The algorithm relies on Lagrangian relaxation and a polynomial time greedy pairing procedure for efficiently splitting sites into paired and unpaired facilities.

Chapter 7

Conclusion

In this thesis, we studied several resource allocation problems that extend the classical facility location problems. The three problems mainly considered in Chapter 4, 5, and 6 respectively are the Unconstrained Fault-Tolerant Resource Allocation ($FTRA_\infty$), Reliable Resource Allocation (RRA), and Constrained Fault-Tolerant Resource Allocation ($FTRA$). For these problems and some of their variants, we developed approximation algorithms with theoretical approximation ratios (solution effectiveness) and runtime bounds (solution efficiency). The techniques we used in the algorithms' design and analysis are extended from linear programming fundamentals (described in Chapter 2) and techniques for approximating the Uncapacitated Facility Location (UFL) problem (described in Chapter 3). In particular, for the uniform case of the problems, we mainly adopted the powerful and versatile primal-dual schema. The framework usually yields fast and analyzable primal-dual algorithms, which are equivalent to some greedy algorithms. The other salient feature of the framework is that the seemingly similar algorithms can solve the problems $FTRA_\infty$, RRA, and $FTRA$, each with a different sets of constraints. For the general case of these problems, we approached the approximate solutions via different types of reductions. In the reductions for $FTRA$, we first extended an LP-rounding algorithm for the Fault-Tolerant Facility Location ($FTFL$) problem, and later showed that $FTRA$ is in general reducible to $FTFL$. In the reductions for RRA and $FTRA_\infty$, we adopted some useful and generic linear programming techniques. The biggest challenge we faced in our work is the analysis of approximation algorithms. A slight change in an algorithm may cause various difficulties in the analysis. Furthermore, there are many different tools for analyzing different types of approximation algorithms, and some of them are rather complicated.

From the practical side, our developed and solved resource allocation models inherited from the facility location models are more general and applicable for optimizing the performances of many contemporary distributed systems. From the theoretical side, our solutions to these models would add new knowledge to the theoretical computer science and operations research fields. Nevertheless, our work is by no means complete. For instance, the approximation results of some resource allocation and facility location problems could still be improved and several interesting open problems still remain. It is also useful to have distributed versions and empirical comparisons of our algorithms. In the rest of this chapter, we discuss some of the future works.

For the non-uniform $FTRA_\infty$ problem, due to its similar combinatorial structure as $FTFL$, it is still unknown whether there exists a constant factor combinatorial algorithm. We have already presented such primal-dual algorithms for the uniform case. It is likely that some clever greedy and local search approximation algorithms would achieve constant ratios, since $FTRA_\infty$ has been shown to be easier than $FTFL$. For the uniform capacitated $FTRA_\infty$ problem, it is possible to improve the obtained approximation ratio of 2.89 to its integrity gap lower bound of 2. Although we have shown that the non-uniform $FTRA$ problem is reducible to $FTFL$, it is still possible for the problem to be approximated better than $FTFL$ for large values of $\boldsymbol{r}$ and $\boldsymbol{R}$. Similar to $FTRA_\infty$, an asymptotic approximation algorithm may exist for $FTRA$. There might also be constant factor LP-rounding algorithms for the k-$FTRA$ problem. For the RRA problem with non-uniform l_{ij}'s, no algorithms with any non-trivial approximation guarantees are known. Better algorithms could possibly exist for the problem with uniform l_{ij}'s. The non-linear RRA problem is still left open to solve. For practical purposes, it is also interesting to study other variants of the resource allocation problems, such as the problems of resource allocation with outliers and online resource allocation.

It is even more challenging to improve the theoretical results of some classical facility location problems. For UFL, the current best approximation ratio is 1.488. Is it still possible to reduce this ratio to the established lower bound of 1.463? Even for the uniform $FTFL$, it is still unknown whether it is more difficult to approximate than UFL. Also, for the general case, is $FTFL$ becoming easier with larger values of $\boldsymbol{r}$? Can it be approximated better than the ratio of 1.7245 using LP-rounding? Is there a constant factor combinatorial algorithm? For the capacitated facility location (CFL) problems, an important open problem is to design a relaxation based approximation algorithm (like LP-rounding and primal-dual) for the hard capacitated case. There is also

room for designing fixed parameter algorithms. For the k-median and k-UFL problems, does a factor 2 approximation algorithm exist? Does k-UFL have the same approximability as k-median? Since the current results for $FTFL$, CFL, and k-median are relatively mature, another future research direction is to solve more complicated facility location models by combining these three. Other facility location variants like the multi-level facility location, universal facility location, and online facility location are also of our interest. Besides, it is interesting to establish relationships between the facility location problems and some graph problems. Furthermore, hardness results are either still missing or can be improved for several facility location problems (and resource allocation problems).

The approximation algorithms for the facility location and resource allocation problems are mostly centralized. However, for many applications it is necessary to have distributed algorithms to work in an environment that lacks of global knowledge. For instance, in a network consisting of sites and clients, the overall algorithm requires a prohibitive communication overhead to collect the whole network topology at a single point before a centralized algorithm can actually run on this point. Several distributed algorithms [117, 60, 119, 55] have already been proposed for UFL. They work by message passing and produce trade-offs of approximation ratios and communications rounds. There is still much need for distributed algorithms for the other problems we have considered. For most facility location and resource allocation problems, empirical studies for comparing different algorithms (especially the LP-rounding algorithms) are still missing. Some initial computer experiments [73, 71, 103] have been done for UFL. The experiments used several sets of benchmark instances to reveal the practical strengths and weaknesses of the algorithms to be compared. It will be fruitful to look into these experiments and conduct similar empirical studies for different facility location and resource allocation problems.

Bibliography

[1] Karen Aardal, Fabian A Chudak, and David B Shmoys. A 3-approximation algorithm for the k-level uncapacitated facility location problem. *Information Processing Letters*, 72(5):161–167, 1999.

[2] EH Emile HL Aarts and Jan K Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 1997.

[3] D.J. Abadi. Data management in the cloud: Limitations and opportunities. *Data Engineering*, page 3, 2009.

[4] Robert Aboolian, Yi Sun, and Gary J. Koehler. A location-allocation problem for a web services provider in a competitive market. *European Journal of Operational Research*, 194(1):64–77, 2009.

[5] Alexander Ageev, Yinyu Ye, and Jiawei Zhang. Improved combinatorial approximation algorithms for the k-level facility location problem. *SIAM Journal on Discrete Mathematics*, 18(1):207–217, 2004.

[6] Alexander A. Ageev and Maxim Sviridenko. An 0.828-approximation algorithm for the uncapacitated facility location problem. *Discrete Applied Mathematics*, 93(2-3):149–156, 1999.

[7] Alexander A Ageev and Maxim I Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

[8] Ankit Aggarwal, Anand Louis, Manisha Bansal, Naveen Garg, Neelima Gupta, Shubham Gupta, and Surabhi Jain. A 3-approximation algorithm for the facility location problem with uniform capacities. *Mathematical Programming*, pages 1–21, 2012.

[9] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows: theory, algorithms, and applications*. Prentice hall, 1993.

[10] Aaron Archer, Ranjithkumar Rajagopalan, and David B. Shmoys. Lagrangian relaxation for the k-median problem: New insights and continuity properties. In *ESA*, pages 31–42, 2003.

[11] Sanjeev Arora. The approximability of np-hard problems. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC)*, pages 337–348. ACM, 1998.

[12] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *Journal of the ACM (JACM)*, 45(1):70–122, 1998.

[13] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristic for k-median and facility location problems. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing (STOC)*, pages 21–29, New York, NY, USA, 2001. ACM.

[14] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM Journal on Computing*, 33(3):544–562, 2004.

[15] Giorgio Ausiello. *Complexity and Approximability Properties: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[16] Michael Bader. Random access machines, September 2013, `http://www5.in.tum.de/lehre/vorlesungen/fundalg/WS02/docs/ram.pdf`.

[17] M. L. Balinski. On finding integer solutions to linear programs. In *Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems*, pages 225–248, 1966.

[18] Manisha Bansal, Naveen Garg, and Neelima Gupta. A 5-approximation for capacitated facility location. In *The 20th Annual European Symposium on Algorithms (ESA 2012)*, pages 133–144. Springer, 2012.

[19] Reuven Bar-Yehuda and Shimon Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981.

[20] Yair Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC)*, pages 161–168. ACM, 1998.

[21] MohammadHossein Bateni and MohammadTaghi Hajiaghayi. Assignment problem in content distribution networks: unsplittable hard-capacitated facility location. In *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pages 805–814, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[22] Babak Behsaz, Mohammad R Salavatipour, and Zoya Svitkina. New approximation algorithms for the unsplittable capacitated facility location problem. In *Algorithm Theory–SWAT 2012*, pages 237–248. Springer, 2012.

[23] Dimitri P Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.

[24] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*. Athena Scientific Belmont, 1997.

[25] Stephen Poythress Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[26] Adriana Bumb. *Approximation Algorithms for Facility Location Problems*. PhD thesis, University of Twente, 2002.

[27] Jaroslaw Byrka. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. In *Proceedings of 10th International Workshop on Approximation, Randomization, and Combinatorial Optimization (APPROX-RANDOM)*, pages 29–43, Princeton, NJ, USA, 20-22 August 2007. Springer-Verlag, Berlin.

[28] Jaroslaw Byrka and Karen Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM Journal on Computing*, 39(6):2212–2231, 2010.

[29] Jaroslaw Byrka, MohammadReza Ghodsi, and Aravind Srinivasan. Lp-rounding algorithms for facility-location problems. Technical report, 2010. http://arxiv.org/abs/1007.3611, CoRR.

[30] Jaroslaw Byrka and Bartosz Rybicki. Improved lp-rounding approximation algorithm for k-level uncapacitated facility location. In *Proceedings of ICALP*, pages 157–169, Warwick, UK, July 9-13 2012. Springer-Verlag, Berlin.

[31] Jaroslaw Byrka, Aravind Srinivasan, and Chaitanya Swamy. Fault-tolerant facility location: A randomized dependent lp-rounding algorithm. In *Proceedings of Integer Programming and Combinatorial Optimization, 14th International Conference (IPCO)*, pages 244–257, Lausanne, Switzerland, 9-11 June 2010. Springer-Verlag, Berlin.

[32] Fangzhe Chang, Jennifer Ren, and Ramesh Viswanathan. Optimal resource allocation in clouds. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 418–425. IEEE, 2010.

[33] Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha. Rounding via trees: deterministic approximation algorithms for group steiner trees and k-median. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC)*, pages 114–123. ACM, 1998.

[34] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 378, Washington, DC, USA, 1999. IEEE Computer Society.

[35] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for facility location problems. *SIAM J. Comput.*, 34(4):803–824, 2005.

[36] Moses Charikar, Sudipto Guha, Éva Tardos, and David B Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing (STOC)*, pages 1–10. ACM, 1999.

[37] Moses Charikar, Samir Khullery, David M. Mountz, and Giri Narasimhanx. Algorithms for facility location problems with outliers. In *Proceedings 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 642–651, Washington, DC, 2001.

[38] Moses Charikar and Shi Li. A dependent lp-rounding approach for the k-median problem. In *The 39th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 194–205. Springer, 2012.

[39] F.A. Chudak and D.B. Shmoys. Improved approximation algorithms for a capacitated facility location problem. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 875–876. Society for Industrial and Applied Mathematics, 1999.

[40] F.A. Chudak and D.P. Williamson. Improved approximation algorithms for capacitated facility location problems. *Mathematical programming*, 102(2):207–222, 2005.

[41] Fabián A Chudak. Improved approximation algorithms for uncapitated facility location. In *Proceedings of the 6th International IPCO Conference on Integer Programming and Combinatorial Optimization*, page 194. Springer-Verlag, 1998.

[42] Fabián A. Chudak and David B. Shmoys. Improved approximation algorithms for the uncapacitated facility location problem. *SIAM J. Comput.*, 33(1):1–25, 2003.

[43] Vasek Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[44] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing (STOC)*, pages 151–158. ACM, 1971.

[45] Gerard Cornuejols, Marshall L. Fisher, and George L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23(8):789–810, 1977.

[46] Gerard Cornuejols, George L Nemhauser, and Lairemce A Wolsey. The uncapacitated facility location problem. Technical report, DTIC Document, 1983.

[47] John Current and Charles Weber. Application of facility location modeling constructs to vendor selection problems. *European Journal of Operational Research*, 76(3):387–392, 1994.

[48] P. Danziger. Turing machines, September 2013, `http://www.math.ryerson.ca/~ddelic/mth714/turing.pdf`.

[49] RG Downey and Michael R Fellows. Fundamentals of parameterized complexity. *Undegraduate Texts in Computer Science, Springer-Verlag*, 2012.

[50] D. Dueck, B.J. Frey, N. Jojic, V. Jojic, G. Giaever, A. Emili, G. Musso, and R. Hegele. Constructing treatment portfolios using affinity propagation. In *Proceedings of the 12th annual international conference on Research in computational molecular biology*, pages 360–371. Springer-Verlag, 2008.

[51] Donald Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26(6):992–1009, 1978.

[52] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[53] Marshall L. Fisher and Dorit S. Hochbaum. Database location in computer networks. *J. ACM*, 27(4):718–735, 1980.

[54] Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008.

[55] Christian Frank and Kay Römer. Distributed facility location algorithms for flexible configuration of wireless sensor networks. In *Proceedings of the 3rd IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 124–141, Santa Fe, NM, USA, 18-20 June 2007. Springer-Verlag, Berlin.

[56] Toshihiro Fujito and Hidekazu Kurahashi. A better-than-greedy algorithm for k-set multicover. In *Proceedings of the Third international conference on Approximation and Online Algorithms (WAOA)*, pages 176–189, Palma de Mallorca, Spain, 6-7 October 2005. Springer-Verlag, Berlin.

[57] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM (JACM)*, 53(3):324–360, 2006.

[58] Michael R Garey, Ronald L Graham, and Jeffrey D Ullman. Worst-case analysis of memory allocation algorithms. In *Proceedings of the fourth annual ACM symposium on Theory of computing (STOC)*, pages 143–150. ACM, 1972.

[59] Michael R Gary and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company, New York, 1979.

[60] Joachim Gehweiler, Christiane Lammersen, and Christian Sohler. A distributed $o(1)$-approximation algorithm for the uniform facility location problem. In *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 237–243, New York, NY, USA, 2006. ACM.

[61] Hadi Goudarzi and Massoud Pedram. Energy-efficient virtual machine replication and placement in a cloud computing system. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 750–757. IEEE, 2012.

[62] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–73, 2008.

[63] Sudipto Guha. *Approximation Algorithms for Facility Location Problems*. PhD thesis, Stanford, USA, 2000.

[64] Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 649–657. Society for Industrial and Applied Mathematics, 1998.

[65] Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31:228–248(21), April 1999.

[66] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. Improved algorithms for fault tolerant facility location. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 636–641, Washington, DC, USA, 7-9 January 2001. ACM/SIAM, Philadelphia, PA.

[67] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. A constant factor approximation algorithm for the fault-tolerant facility location problem. *J. Algorithms*, 48(2):429–440, 2003.

[68] Dorit S Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.

[69] Dorit S. Hochbaum. Heuristics for the fixed cost median problem. *Mathematical Programming*, 22(1):148–162, 1982.

[70] Dorit S Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1996.

[71] M. Hoefer. Experimental comparison of heuristic and approximation algorithms for uncapacitated facility location. In *Experimental and Efficient Algorithms: Second International Workshop, WEA 2003, Ascona, Switzerland, May 26-28, 2003. Proceedings*, pages 622–622. Springer, 2003.

[72] Q.S. Hua, D. Yu, F.C. Lau, and Y. Wang. Exact algorithms for set multicover and multiset multicover problems. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 34–44, Honolulu, Hawaii, USA, 16-18 December 2009. Springer-Verlag, Berlin.

[73] Kamal Jain, Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay V. Vazirani. Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP. *Journal of the ACM*, 50(6):795–824, 2003.

[74] Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing (STOC)*, pages 731–740, Montreal, Quebec, Canada, 19-21 May 2002. ACM, New York, NY.

[75] Kamal Jain and Vijay V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 2–13, 1999.

[76] Kamal Jain and Vijay V. Vazirani. An approximation algorithm for the fault tolerant metric facility location problem. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 177–183, Saarbrücken, Germany, 5-8 September 2000. Springer-Verlag, Berlin.

[77] Kamal Jain and Vijay V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

[78] David S Johnson. Approximation algorithms for combinatorial problems. *Journal of computer and system sciences*, 9(3):256–278, 1974.

[79] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[80] Andreas Klose and Andreas Drexl. Facility location models for distribution system design. *European Journal of Operational Research*, 162(1):4–29, 2005.

[81] S.G. Kolliopoulos. Approximating covering integer programs with multiplicity constraints. *Discrete applied mathematics*, 129(2):461–473, 2003.

[82] Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer-Verlag, Berlin, Third Edition, 2006.

[83] Madhukar R. Korupolu, C. Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 1–10, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[84] Jakob Krarup and Peter Mark Pruzan. The simple plant location problem: survey and synthesis. *European Journal of Operational Research*, 12(1):36–81, 1983.

[85] Ravishankar Krishnaswamy and Maxim Sviridenko. Inapproximability of the multi-level uncapacitated facility location problem. In *Proceedings of the Twenty-Third Annual ACM-SIAM Sympkosium on Discrete Algorithms (SODA)*, pages 718–734. SIAM, 2012.

[86] Alfred A Kuehn and Michael J Hamburger. A heuristic program for locating warehouses. *Management science*, 9(4):643–666, 1963.

[87] N. Lazic, I. Givoni, B. Frey, and P. Aarabi. Floss: Facility location for subspace segmentation. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 825–832. IEEE, 2010.

[88] Charles E Leiserson, Ronald L Rivest, Clifford Stein, and Thomas H Cormen. *Introduction to algorithms*. The MIT press, 2001.

[89] Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. In *Proceedings of ICALP*, pages 77–88, Zurich, Switzerland, 4-8 July 2011. Springer-Verlag, Berlin.

[90] Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Information and Computation*, 2012.

[91] Shi Li and Ola Svensson. Approximating k-median via pseudo-approximation. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing (STOC)*, pages 901–910. ACM, 2013.

[92] Yu Li, Donglei Du, Naihua Xiu, and Dachuan Xu. Improved approximation algorithms for the facility location problems with linear/submodular penalty. In *Proceedings of the 19th annual international conference on computing and combinatorics (COCOON)*, pages 292–303, Hangzhou, China, June 21-23 2013. Springer-Verlag, Berlin.

[93] Kewen Liao and Hong Shen. Fast fault-tolerant resource allocation. In *Proceedings of 12th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 231–236, Gwangju, Korea, October 20-22 2011. IEEE.

[94] Kewen Liao and Hong Shen. Unconstrained and constrained fault-tolerant resource allocation. In *Proceedings of the 17th annual international conference on computing and combinatorics (COCOON)*, pages 555–566, Dallas, Texas, USA, 14-16 August 2011. Springer-Verlag, Berlin.

[95] Kewen Liao and Hong Shen. Approximating the reliable resource allocation problem using inverse dual fitting. In *Proceedings of the Eighteenth Computing: The Australasian Theory Symposium (CATS)*, page Vol. 128, Melbourne, Australia, January-February 2012. ACS, Sydney.

[96] Kewen Liao and Hong Shen. Lp-based approximation algorithms for reliable resource allocation. *The Computer Journal*, 57(1):154–164, 2014.

[97] Kewen Liao, Hong Shen, and Longkun Guo. Improved approximation algorithms for constrained fault-tolerant resource allocation. In *Fundamentals of Computation Theory (FCT) - 19th International Symposium*, pages 236–247, Liverpool, UK, August 19-21 2013. Springer-Verlag, Berlin (extended version invited to the special issue of Theoretical Computer Science).

[98] Kewen Liao, Hong Shen, and Longkun Guo. Constrained fault-tolerant resource allocation. *Theoretical Computer Science*, submitted in December 2013, at `http://arxiv.org/abs/1208.3835`, currently under review.

[99] Jyh-Han Lin and Jeffrey Scott Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, 44(5):245–249, 1992.

[100] Jyh-Han Lin and Jeffrey Scott Vitter. ϵ-approximations with minimum packing constraint violation. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing (STOC)*, pages 771–782, Victoria, British Columbia, Canada, 4-6 May 1992. ACM, New York, NY.

[101] A. Liotta, C. Ragusa, and G. Pavlou. Near-optimal service facility location in dynamic communication networks. *Communications Letters, IEEE*, 9(9):862–864, Sep 2005.

[102] C. Greg Plaxton Madhukar R. Korupolu and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of Algorithms*, 37(1):146–188, 2000.

[103] Mohammad Mahdian. *Facility Location and the Analysis of Algorithms through Factor-Revealing Programs*. PhD thesis, MIT, USA, 2004.

[104] Mohammad Mahdian, Evangelos Markakis, Amin Saberi, and Vijay Vazirani. A greedy facility location algorithm analyzed using dual fitting. In *Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 127–137, Berkeley, CA, USA, 18-20 August 2001. Springer-Verlag, Berlin.

[105] Mohammad Mahdian and Martin Pál. Universal facility location. In *Algorithms-ESA 2003*, pages 409–421. Springer, 2003.

[106] Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 229–242, Rome, Italy, 17-21 September 2002. Springer-Verlag, Berlin.

[107] Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. A 2-approximation algorithm for the soft-capacitated facility location problem. In *6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 129–140. Springer, 2003.

[108] Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Approximation algorithms for metric facility location problems. *SIAM J. Comput.*, 36(2):411–432, 2006.

[109] Olvi L Mangasarian. Mathematical programming in data mining. *Data mining and knowledge discovery*, 1(2):183–201, 1997.

[110] Alan S Manne. Plant location under economies-of-scale?decentralization and computation. *Management Science*, 11(2):213–235, 1964.

[111] M.T. Melo, S. Nickel, and F. Saldanha da Gama. Facility location and supply chain management - a review. *European Journal of Operational Research*, 196(2):401–412, 2009.

[112] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[113] Ramgopal R Mettu and C Greg Plaxton. The online median problem. In *In Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, 2000.

[114] Ramgopal R Mettu and C Greg Plaxton. The online median problem. *SIAM Journal on Computing*, 32(3):816–832, 2003.

[115] Adam Meyerson. Online facility location. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on (FOCS)*, pages 426–431. IEEE, 2001.

[116] Pitu B. Mirchandani and Richard L. Francis, editors. *Discrete Location Theory*. John Wiley, New York, 1990.

[117] Thomas Moscibroda and Roger Wattenhofer. Facility location: distributed approximation. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 108–117, New York, NY, USA, 2005. ACM.

[118] George L Nemhauser and Laurence A Wolsey. *Integer and combinatorial optimization*, volume 18. Wiley New York, 1988.

[119] Saurav Pandit and Sriram Pemmaraju. Return of the primal-dual: distributed metric facilitylocation. In *Proceedings of the 28th ACM symposium on Principles of distributed computing (PODC)*, pages 180–189, New York, NY, USA, 2009. ACM.

[120] Christos H Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *Journal of computer and system sciences*, 43(3):425–440, 1991.

[121] L. Qiu, V.N. Padmanabhan, and G.M. Voelker. On the placement of web server replicas. In *Proceedings of IEEE Infocom*, pages 1587–1596, Anchorage, AK, 22-26 April 2001. IEEE, New York, NY.

[122] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing (STOC)*, pages 475–484. ACM, 1997.

[123] Bartosz Rybicki and Jaroslaw Byrka. Improved approximation algorithm for fault-tolerant facility placement. Technical report, 2013. http://arxiv.org/abs/1311.6615, CoRR.

[124] Alexander Schrijver. *Theory of linear and integer programming*. Wiley. com, 1998.

[125] Hong Shen and Shihong Xu. Approximation algorithms for fault tolerant facility allocation. *SIAM Journal on Discrete Mathematics*, 27(3):1584–1609, 2013.

[126] David B. Shmoys. Approximation algorithms for facility location problems. In *APPROX '00: Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 27–33, London, UK, 2000. Springer-Verlag.

[127] David B Shmoys. The design and analysis of approximation algorithms: facility location as a case study. In *Trends in Optimization, AMS Proceedings of Symposia in Applied Mathematics*, volume 61, pages 85–97, 2004.

[128] David B Shmoys and Éva Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1-3):461–474, 1993.

[129] David B. Shmoys, Eva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 265–274, El Paso, Texas, USA, 4-6 May 1997. ACM, New York.

[130] John F Stollsteimer. A working model for plant numbers and locations. *Journal of Farm Economics*, 45(3):631–645, 1963.

[131] Maxim Sviridenko. An improved approximation algorithm for the metric uncapacitated facility location problem. In *Proceedings of the 9th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 240–257, Cambridge, MA, USA, 27-29 May 2002. Springer-Verlag, Berlin.

[132] Chaitanya Swamy and David B. Shmoys. Fault-tolerant facility location. *ACM Trans. Algorithms*, 4(4):1–27, 2008.

[133] Hamdy A. Taha. *Operations Research: An Introduction*. Pearson/Prentice Hall, 2007.

[134] Mikkel Thorup. Quick k-median, k-center, and facility location for sparse graphs. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 249–260. Springer-Verlag, 2001.

[135] Mikkel Thorup. Quick and good facility location. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 178–185. Society for Industrial and Applied Mathematics, 2003.

[136] Mikkel Thorup. Quick k-median, k-center, and facility location for sparse graphs. *SIAM Journal on Computing*, 34(2):405–432, 2005.

[137] Alan M Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 42(2):230–265, 1936.

[138] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, 2001.

[139] Vijay V Vazirani. Primal-dual schema based approximation algorithms. In *Theoretical aspects of computer science*, pages 198–207. Springer, 2002.

[140] J. Vygen. Approximation Algorithms for Facility Location (lecture notes), Report No. 05950-OR, Research Institute for Discrete Mathematics, University of Bonn. 2005.

[141] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge University Press, 2011.

[142] D.P. Williamson. The primal-dual method for approximation algorithms. *Mathematical Programming*, 91(3):447–478, 2002.

[143] Georg Wittenburg and Jochen Schiller. A survey of current directions in service placement in mobile ad-hoc networks. In *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PERCOM)*, pages 548–553, Hong Kong, 17-21 March 2008. IEEE Computer Society, Washington, DC.

[144] Shihong Xu. *Replica Placement Algorithms for Efficient Internet Content Delivery*. PhD thesis, The University of Adelaide, Australia, 2009.

[145] Shihong Xu and Hong Shen. The fault-tolerant facility allocation problem. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 689–698, Honolulu, Hawaii, USA, December 16-18 2009. Springer-Verlag, Berlin.

[146] Shihong Xu and Hong Shen. QoS-oriented content delivery in e-learning systems. In *Information Technology in Medicine and Education, IEEE International Symposium on (ITME)*, Jinan, China, 2009.

[147] Li Yan. *Approximation algorithms for the Fault-Tolerant Facility Placement Problem*. PhD thesis, The University of California Riverside, USA, 2013.

[148] Li Yan and Marek Chrobak. Approximation algorithms for the fault-tolerant facility placement problem. *Information Processing Letters*, 111(11):545 – 549, 2011.

[149] Li Yan and Marek Chrobak. New results on the fault-tolerant facility placement problem. Technical report, 2011. http://arxiv.org/abs/1108.5471, CoRR.

[150] Li Yan and Marek Chrobak. Lp-rounding algorithms for the fault-tolerant facility placement problem. Technical report, 2012. http://arxiv.org/abs/1205.1281, CoRR, conference version appeared in CIAC 2013.

[151] Jiawei Zhang. Approximating the two-level facility location problem via a quasi-greedy approach. *Math. Program.*, 108(1):159–176, 2006.

[152] Jiawei Zhang, Bo Chen, and Yinyu Ye. A multi-exchange local search algorithm for the capacitated facility location problem. In *Integer Programming and Combinatorial Optimization (IPCO)*, pages 219–233. Springer, 2004.

[153] Peng Zhang. A new approximation algorithm for the k-facility location problem. *Theoretical Computer Science*, 384(1):126–135, 2007.