Tel Aviv University
The Raymond and Beverly Sackler Faculty of Exact Sciences
The Blavatnik School of Computer Science

Resource Placement in Cloud Computing and Network Applications

by

Yuval Rochman

A thesis submitted in partial fulfillment for the degree of
Doctor of Philosophy

This work was carried out under the supervision of
Prof. Hanoch Levy

Submitted to the Senate of Tel-Aviv University

October 2017

To my loving family and friends.

Abstract

In this thesis we address the problem of resource allocation in geographically distributed systems, in particular cloud computing. We consider a large-scale system and regionally distributed demands for various resources of multiple types. The system operator aims at placing the resources across regions to maximize the profit, thus needing to address the problem of how to place the resources in response to the demand. The system is faced with an arbitrary, multi-dimensional, stochastic demand, thus the operator should optimize the profit while taking into account the full demand distributions. Cloud computing and online services, utilizing regional datacenters and facing the problem of where to place various servers, fall under this paradigm.

Our problems are developed under a very wide cost model, thus allowing the accommodation of many systems. These problems have various formulations based on the systems they represent. We provide efficient algorithms running in a fast polynomial time[1].

Our algorithms and solutions can be used for:

1. Exact system optimization;
2. Deriving lower bounds for heuristic based analysis, and;
3. Sensitivity analysis.

Our solutions are based on analytic techniques utilizing stochastic analysis as well as graph theory methodologies that can be applied to other optimization/combinatorial problems.

[1] Polynomial in the number of resources the operator places or reposition.

Acknowledgements

First of all I would like to express my deepest gratitude to my advisor Professor Hanoch Levy. He has been a constant source of ideas, advice, and support. His knowledge on presenting and telling a good story has contributed to my personal life and improved my self-marketing. Today I feel that I am a better researcher, a more engaging instructor and even a more appealing salsa teacher. He has generously dedicated much of his time and resources to my learning process, and helped me improve.

I can hardly find the words to express the wealth of knowledge I have gained from him.

I would also like to thank my collaborator Dr. Eli Brosh for contributing his industrial and academic knowledge. He helped me overcome new obstacles and had fruitful discussions with me. His contribution was essential to my work. I would like to thank Dr. Gail Gilboa-Freedman for collaborating with us on the work we provided in Chapter 9. Thanks to Prof. Noga Alon, Prof. Refael Hassin and Prof. Arie Tamir for fruitful discussions and for their helpful advice on my research. I would like to thank an anonymous referee who proposed the algorithm described in Section 5.8.

Finally, I would like to thank my family and friends for supporting me and providing me with the help I needed while working on this thesis.

Contents

Symbols

Symbol	Name	Explanation (if needed)
k	Number of regions (areas).	
m	Number of resource types.	
B_i	Resource capacity.	The maximum number of requests a type i resource can serve.
L_i^j	Number of type i resources in region j.	
L_i	Number of type i resources.	$L_i = \sum_{j=1}^{k} L_i^j$
L^j	Number of resources in region j.	$L^j = \sum_{i=1}^{m} L_i^j$
L	The placement.	$L = \{L_i^j\}$.
D_i^j	The demand for type i resources in region j.	The random variable of the stochastic demand for type i resources in region j.
D	The demand.	$D = \{D_i^j\}$.
d_i^j	The demand realization for type i resources in region j.	The actual number of requests for type i resources in region j.
R_i	Global revenue constant.	The revenue gained by satisfying globally (by a local or a remote region) a type i request.
R_{glo}	Global revenue constant.	In a simple profit function, the global revenue constants R_i are all equal to each other, i.e., $R_i = R_{glo}$ for every type i.
R_i^j	Local revenue constant.	The revenue gained by satisfying locally (by the local region) a type i request in region j.

R_{loc}	Local revenue constant.	In a simple profit function, the local revenue constants R_i^j are all equal to each other, i.e., $R_i^j = R_{glo}$ for every type i and region j.
$C_i(), C_j(), C_i^j()$	The operational cost functions.	The general non-negative cost functions.
$P(D, L)$	The profit	The profit given demand D and placement L.
$\zeta_i(), \zeta^j(), \zeta_i^j()$	The marginal (profit) functions.	The functions composing the profit. See Eqs. (3.5)-(3.8).
$g_i(), g^j(), g_i^j()$	The conditional-marginal functions.	Given a demand D, we define for every $n \in \mathbb{N}$, $g_i(n) = \zeta_i(n, D), g^j(n) = \zeta^j(n), g_i^j = \zeta_i^j(n, D)$. These functions are concave.
$\Delta g_i(), \Delta g^j(), \Delta g_i^j()$	The marginal-differential functions.	The differential of the conditional functions, i.e., $\Delta g(n) = g(n) - g(n-1)$ for $n \geq 1$. These functions are monotonically non-increasing.
$L(t)$	The placement at period t.	The given placement at previous period.
$D(t+1)$	The demand at period $t+1$.	The given demand predication of the current period.
$L(t+1)$	The placement at period t.	The placement we should find and place for the current period.
r	The reposition bound.	The maximum number of allowed repositions between periods t and $t+1$.
L_{opt}	The optimal placement	An optimal solution, which maximizes the profit.
s^j	The storage value of region j.	Only in capacity-constrained systems. The maximum number of resources allowed in region j, i.e., $L^j \leq s^j$
s	The total storage value.	Only in capacity-constrained systems. $s = \sum_{j=1}^{k} s^j$.

List of Problems

Static Placement Problems

Prob.	Cap.	Hom.	S. reg	Sim.	Algo.	Comp.	Chap.
SPP-1	Y	Y	Y	Y	MaP	$O((s+m)\cdot\log m)$	4
SPP-2	Y	Y	N	Y	MuRMaP	$O((s+m)\cdot\log m)$	5
SPP-3	Y	N	N	Y	BG	$O(smk(m+k))$	6
SPP-4	N	N	N	N	G-BG	$O(\|L_{opt}\|f(m,k))$	7

Constrained-Reposition Placement Problems

Prob.	S. type	Opt.	Algo.	Comp.	Chap.
ConRePP-1	Y	Y	U&Me	$O((r+k)\cdot\log k)$	9
ConRePP-2	N	N	SCC	$O(rmk(k+m)^2)$	10
ConRePP-2	N	N	Hybrid	$O(r_{max}mk(k+m)^2)$	10
ConRePP-2	N	N	SCO	$O(r\min(k,m)^3(k+m))$	10

Glossary of abbreviations

Cap.	Capacity-constrained system
Hom.	Homogeneous-region system
S. reg	Single-region system
Sim.	Simple profit
Algo.	Algorithm name (Abbreviation)
Comp.	Time complexity
Chap.	Chapter
S. type	Single-type system
Opt.	The solution is optimal
$f(m,k)$	$km+(m+k)\log(m+k)$.

Chapter 1

Introduction

The Internet has witnessed a surge in the usage of its services, featuring large volumes of content and demands. In these services, the service provider replicates content and places resources across multiple locations for better performance and availability. For example, popular cloud computing platforms like Amazon EC2 [2] and Microsoft Azure [3] organize a shared pool of (virtual) servers[1] in geographically distributed datacenters to enable on-demand delivery of computer resources at scale. Differently from a centralized design, the service provider can place server resources at geographical areas close to its users to provide an adequate level of service quality, e.g., low response times and better resilience.

In such systems, the service provider needs to balance two main factors: (a) the revenue from serving a demand, where it is typically better to serve a demand by a resource located in the same area rather than by one located remotely; (b) the costs of placing and operating the resources. For example, Amazon EC2 bills server instance usage according to a pay-per-use plan where the price varies depending on the instance type and the datacenter location. Thus, the service provider deals with regionally distributed demands for various resources by aiming to find a high profit placement of resources at the regions (areas).

Another example of an application that falls under this paradigm is a peer-assisted Video-on-Demand (VoD). A provider of such service aims to serve video content to users who are located across various geographical regions and are each equipped with a set-top box. Serving all requests from a central server can lead to bottlenecks, limiting the system's scalability. It can therefore make use of the users' boxes to store video content and serve it, using their upload channel, to other users. Serving videos within a region is more profitable than across regions. The provider faces the problem of where

[1] Virtual servers are also called instances or virtual machines in the cloud computing community.

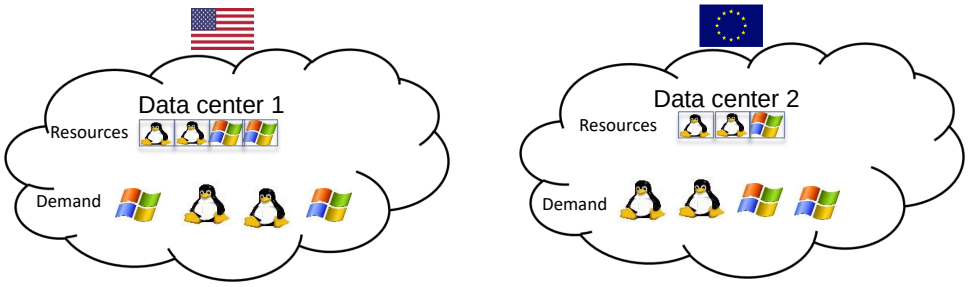


FIGURE 1.1: A basic example of a cloud system, where users are requesting servers with different operating systems.

to place copies of the various movies to minimize the expected cost (or equivalently, maximize the profit) of servicing video requests. A variety of other applications that fall under this formulation is outlined in Section 3.3.

In this thesis, we address the general problem of how to place resources over geographically distributed regions in response to stochastic demands. To this end, we consider a variety of formulations and models and propose solutions to these placement problems.

1.1 Model framework

This work includes a development of a variety of models that incorporate the major cost parameters affecting the design of the resource placement systems. For the sake of presentation, in Chapter 3 we present a generalized framework that can be used to formulate all our placement problems.

The systems considered comprise multiple regions (areas) and multiple types of resources. The total number of resources that can be placed in each region can be restricted. The provider is given a probabilistic forecast of the stochastic demand $D = \{D_i^j\}$, where D_i^j is a random variable that represents the demand for resources of type i in region j. The demand D is given under general settings, i.e., the distribution of each random variable D_i^j is arbitrary, and these random variables are neither necessarily identically distributed nor independent of each other. It should be noted that a probabilistic demand prediction derives more accurate results than a deterministic one. The prediction can be extracted using demand traces as done in the industry [4].

Based on the demand, the provider's objective is to optimally place (under some constraints) the resources in the regions to maximize profit. The profit is equal to the revenue of granting the demand requests minus the operational costs of operating the system. The revenue accounts for geographical distances, so that granting a request

from the same region where the request was made has a higher profit than granting a request from a remote region; not granting a request incurs zero or negative profit. The operational costs are arbitrary convex functions and can represent regional, energy, cloud rental and other costs.

The generality of the demand and the operational costs described above allows us to address a variety of scenarios:

1. The demand and operational costs vary across resource types and regions. For example, in cloud providers such as Amazon EC2 and Microsoft Azure, the demand and prices change across different platforms and regions [2, 3, 5]. In addition, in VoD applications, the demand can change drastically across movies (and the servers containing them); while some are very popular, the others may be quite esoteric. In fact, studies show a large diversity of usage patterns exhibited by existing VoD services such as NetFlix, IPTV, and YouTube [6, 7].

2. The demand has high variance and is unpredictable. High-variance demand may be caused, for example, by flash crowds or by some network events (e.g., an unpredicted datacenter failure, leading to all requests being granted by a remote region). It has been observed that the demand in these systems can exhibit large variations and it can be unpredictable [8]. For settings with low-variance demand, one may consider dynamic placement strategies with (nearly) deterministic inputs, such as [9, 10]. These works are complementary to ours.

3. The stochastic demand varies over time, changing significantly (by an order of magnitude) over the course of a day due to, for example, changes in the number of available users [5]. To obtain a cost-effective operation of the system, the provider needs to dynamically adapt to demand changes, and periodically reposition allocated resources.

Our placement model framework, which is based on the full demand distributions, may deviate from prior approaches focused on the mean of the distribution. The use of a simple scheme like proportional mean placement may be quite inefficient[2] as demonstrated in the following simple example: Consider a single-region system that can store up to n resources. Suppose there are two types of resource to be placed, and every resource can grant one request. The first resource type has a deterministic demand of n requests and the second one has a stochastic demand of nk^2 requests w.p $\frac{1}{k}$ and zero requests w.p $1 - \frac{1}{k}$. The objective of efficient placement is to maximize the expected number of requests granted (the system's profit). The optimal algorithm will place n resources of the first type, yielding a revenue of n granted requests. In contrast, a proportional mean

[2]Proportional mean placement has been shown to be optimal under some conditions, see e.g., [11].

strategy which tunes the number of replicas to the average number of requests, will place $\frac{n}{k+1}$ resources of the first type and $\frac{nk}{k+1}$ of the second type, yielding a drastically smaller revenue of $\frac{2n}{k+1}$. As k approaches infinity, the number of requests granted by the proportional mean placement approaches to zero, while the number of requests granted by the optimal placement is equal to n. A further applicative example is discussed in Chapter 6.4.

Remark 1.1. One might consider optimizing of proportional mean by truncating the demand distributions, i.e., the demand cannot be larger than the system capacity. Even under that version, proportional mean might provide an allocation which is away from the optimal. Consider a single-region system that has $\frac{n}{2}$ resource types, denoted by $t_1, t_2, .., t_{\frac{n}{2}}$, with deterministic demand of a single request. In addition, the system contains $\frac{n}{2}$ resource types (denoted by $t_{\frac{n}{2}+1}, , .., t_n$), in each of which the demand has probability of $\frac{1}{\sqrt{n}}$ to have n requests, and probability of $1 - \frac{1}{\sqrt{n}}$ to have zero requests. Suppose the system contains n resources. The optimal placement allocates one resource for resource types $t_1, t_2, .., t_{\frac{n}{2}}$, and allocates $\frac{n}{2}$ resources from one of the other resource types. The optimal profit, equals to $\frac{n}{2} + \frac{\sqrt{n}}{2} = \theta(n)$. Proportional mean, however, allocates $\frac{1}{\frac{n}{2}+\frac{n}{2}\cdot\sqrt{n}} \cdot n = \frac{2}{1+\sqrt{n}}$ resources for resource types $t_1, t_2, .., t_{\frac{n}{2}}$, and $\frac{\sqrt{n}}{\frac{n}{2}+\frac{n}{2}\cdot\sqrt{n}} \cdot n = \frac{2\sqrt{n}}{1+\sqrt{n}}$ for resource types $t_{\frac{n}{2}+1}, , .., t_n$. Its profit is equal to $\frac{2}{1+\sqrt{n}} \cdot \frac{n}{2} + \frac{2\sqrt{n}}{1+\sqrt{n}} \cdot \frac{n}{2} \cdot \frac{1}{\sqrt{n}} = \frac{2n}{\sqrt{n}+1} = \theta(\sqrt{n})$. Thus, proportional mean is far by a factor of $\theta(\sqrt{n})$ from the optimal value.

Although we focus our presentation on a cloud environment, the generality of our model allows it to serve as a building block in a variety of dynamic resource placement problems, spanning geo-distributed inventory problems to personnel allocation in multi-contact center settings.

1.2 Static Placement Problems

The first part of our work deals with several placement problems where the service provider optimizes the profit at a specific time period. These problems are called **Static Placement Problems (SSPs)** as opposed to dynamic placement problems where the placement is changed over time. All these problems are formulated similarly: given a demand $D = D(t)$ at period t, the goal is to find a placement L that maximizes the profit $P(L, D)$, and place these resources. The formulation of every SPP problem is shown below:

A Static Placement Problem (SPP)	
Input:	A probabilistic forecast of the demand $D = D(t)$ at period t.
Problem:	Find the placement L that maximizes the profit $P(L, D)$.

Our work solves a variety of static placement problems. These problems are appropriate for systems and applications with different properties, which are corresponding to different time complexity. For example, we have solved static placement problems for homogeneous systems, where the demand of requests is symmetric across the regions, and we have solved static placement problems for heterogeneous systems. The optimal solution for the homogeneous system is faster and simpler than that for the heterogeneous system, while most practical systems are heterogeneous and thus a solution for heterogeneous systems is more applicative.

After deriving the optimal placement L_{opt}, the operator needs to place the resources in the different regions. We present efficient algorithms to these problems that are polynomial in the number of resources the operator places (i.e., the number of resources in the optimal placement), in the number of resource types and in the number of regions.

Remark 1.2. Note that since the operator must physically place the resources (not only compute the quantities of the number of resources in every region), the time complexity of every algorithm that solves a Static Placement Problem should be at least polynomial in the number of resources the operator places. This is why our proposed solutions are efficient. Further if one has to compute the demand distribution (or just read it) of an arbitrary demand, one has to invest $O(s)$ work on the demand input alone, where s is an upper bound over the number of resources a placement can have ($|L| = \sum_i \sum_j L_i^j \leq s$). If we assume that the operator goal is only to compute the quantities of the number of resources in every region and an oracle is given that can provide probability values of the demand, then our proposed algorithms run in pseudo-polynomial[3] in the size of the thin input.

A summary of these problems is given next, and their mathematical formulation is provided at the beginning of each chapter, based on the model given in Chapter 3.

1.2.1 Single-region system (SPP-1) and multi-region homogeneous system (SPP-2) [12]

In Chapters 4 and 5, we study the first static placement problems (published in [12]). For these problems, we make two significant assumptions: 1) the system comprises

[3] A **pseudo-polynomial algorithm** is one that its running time is polynomial in the numeric value of the input, but is exponential in the length of the binary-encoded input.

homogeneous regions, i.e., the demand is identically distributed across regions, each region has the same capacity for resource placement, etc. 2) the profit function P is a simplistic revenue-only function. A formal definition of the simplistic profit function can be found in Section 3.4. In Chapter 4 we focus on solving the problem for a single-region system; this is used to expose the reader to the problem formulation and principles of the analysis. In Chapter 5 we solve the problem for a multi-region system with homogeneous regions. Our main result is that the optimal placement in these systems is of a *max percentile* nature; that is, it is based on the tail distributions of the demand. This is in contrast to past results where various solutions were based on the proportional mean (we discuss their performance in detail in Section 5.7). We show that the solution possesses a *balancing* property which narrows the state space of the solution to that of symmetrically-full balanced placements.

We derive combinatorial algorithms for these problems and demonstrate that they are of a relatively low complexity and thus very practical. The optimality of the solution and the algorithms can be used either to achieve optimal operation, as a benchmark, or to provide operational guidelines for the design of these systems. We use a numerical analysis and simulation to validate our results and study the system's behavior. We demonstrate how to evaluate the performance of alternative placement strategies relative to that of the optimal one.

Most of the Chapter 5 proofs are presented in Appendix a.

1.2.2 Heterogeneous-region system (SPP-3) [13]

In Chapter 6 we extend the problems of Chapters 4 and 5 to include dealing with a heterogeneous system. These results have also been published in [13]. The analysis techniques presented for a single- and homogeneous-region system (SPP-2, SPP-1) are not powerful enough to address a general asymmetric setting.

We demonstrate that one can optimize the problem by transforming the placement problem into min-cost flow problem over a large eight-layer graph. The min-cost problem is traditionally solved by a min-cost flow algorithm, such as the SSP (Successive Shortest Paths) algorithm. Owing to the fact that the eight-layer graph is large, the SSP algorithm runs in high time complexity. To solve this problem, we develop an optimized version of the SSP algorithm tailored to our problem called **Bipartite Graph (BG)**. It provides a low-complexity optimal algorithm, and thus it is suitable for handling large-scale systems. BG can serve to derive upper bounds for heuristic approaches, and can be used for sensitivity analysis. We use a numerical analysis to validate our results and

to demonstrate that alternative mean-based analysis may achieve, in certain cases, a performance that is drastically worse than the optimal one.

The majority of the chapter's proofs are presented in Appendix b.

1.2.3 The general Static Placement Problem: unconstrained-capacity system with a generalized profit function (SPP-4)

In Chapter 7 we further extend the analysis provided in Chapter 6 (SPP-3) by enriching the model and the profit function by adding a variety of parameters important for cloud computing applications. These parameters include server rental costs, server capacity and resource type dependency; the profit function presented in the previous problems (SPP-1, SPP-2, and SPP-3) does not take these parameters into account. Thus, the current problem (SPP-4) is harder than the previous problems. To solve the current problem we propose new solution techniques, such as Successive-Shortest-Path-Negative-Edges combined with node-potentials. These results are to be reported in [14].

In addition, the algorithm presented for this problem addresses one more difficulty posed by the algorithm of the previous problem (SPP-3): the time complexity of the algorithm that solves SPP-3 depends on the maximum number of resources placed in a system. In some cloud applications this number is large; for example, Microsoft's data centers contain an order of ten thousands servers [15]. Thus, we introduce and solve the *unconstrained-capacity* placement problem where there is no restriction on the number resources allowed in an area, given that there exists an optimal solution. The time complexity of the algorithm that solves this problem depends on the size of the placement and not on the maximal number of resources placed in a system. In addition, the unconstrained-capacity problem is useful for capacity planning – it allows one to determine the total quantity of resources (and their locations) that optimizes the profit, i.e., when the marginal profit of placing a resource approaches zero.

To derive the solution, we prove a new theoretical result called the **SSP convexity theorem**. The theorem can be applied in other optimization problems (e.g., [13, 16, 17]) where an algorithm called Successive Shortest Path (SSP) is used to solve a min-cost flow problem.

By using these techniques, we yield a low-complexity algorithm that is significantly **faster** than the one proposed in SPP-3.

1.3 Sensitivity of Optimal Placements [1]

In Chapter 8 we start dealing with a dynamically changing (stochastic) demand. A naive strategy to adapt in case of dynamically changing conditions (demand) is to allocate an optimal placement in response to any change. However, this strategy might be expensive in the number of repositions, and it might be infeasible to implement due to limitations of operating new servers or turning them off. Moreover, in Chapter 8 (based on the results published in [1]), we analyze the sensitivity of optimal placements to changes in the (stochastic) demand. Our results reveal that when seeking an optimal placement, even small demand fluctuations can pose a relatively large reposition of the resources. On the contrary, these small demand deviations result in a minor effect on the profit of the optimal placement. Therefore, a practical treatment of the placement problem should account for reposition cost[4] and should possibly limit the number of resources that can be repositioned.

The analysis presented in Chapter 8 is based on mapping the problem to a min-cost flow problem (in graph theory) we used when solving the unconstrained-capacity static placement problem (SPP-4).

1.4 Constrained-Reposition Placement Problems

Addressing the difficulties introduced by large repositions, we define the *Constrained-Reposition Placement Problems (ConRePPs)*. These problems are in response to changing demands as they found an optimal placement under a constraint of allowing up to a fixed number of additions or removals of resources. The need for reposition rises due to some system change, such as a change in the demand, in the cost structure (e.g., resource price fluctuations[5]), or in the network topology due to a site failure. These problems are formulated as follows: given a placement $L(t)$ at period t what is an optimal placement with respect to a new demand $D(t+1)$ that can be achieved under a constraint of allowing up-to r unit operations[6]? The formulation of each problem is presented as follows:

[4] In a cloud environment, the reposition cost can capture the overhead of setting up new server instances and tearing down old ones.

[5] The price of spot instances in Amazon's cloud is dynamically determined by the demand and supply of instances [2].

[6] A unit operation is either a single *addition* or a single *subtraction* of a resource.

A Constrained-Reposition Placement Problem (ConRePP)	
Input:	Demand $D(t+1)$ at period $t+1$, placement $L(t)$ at period t, reposition bound r.
Problem:	Find a placement $L(t+1)$ that maximizes $P(L(t+1), D(t+1))$ under the reposition constraint $\sum_{i=1}^{m}\sum_{j=1}^{k}\|L_i^j(t) - L_i^j(t+1)\| \leq r$, where $L_i^j(t)$ is the number of type i resources at region j in period t.

These problems are challenging; not only do they need to deal with arbitrary stochastic demands that can vary over time, but we have also established theoretically evidence that some of these problems are hard[7], as opposed to the static placement problems that are solved in polynomial time.

After deriving the optimal placement L_{opt}, the operator needs to reposition the resources in the different regions. We present efficient algorithms to these problems that are polynomial in the number of resources the operator reposition, in the number of resource types and in the number of regions.

Remark 1.3. Note that since the operator must physically reposition the resources, the time complexity of every algorithm that solves a Constrained-Reposition Placement Problem should be at least polynomial in the number of resources the operator place. This is why our proposed solutions are efficient.

1.4.1 The single-type Constrained-Reposition Placement Problem (ConRePP-1)

In Chapter 9 we solve the Constrained-Reposition Placement Problem given a single-type system, based on a submitted article [18]. We propose an effective greedy algorithm for the repositioning problem, called the *Unary and Move (U&Me)* that gives an optimal solution. The time complexity of the algorithm is almost linear by the reposition bound r. The algorithm uses a special greedy approach whereby it starts by greedily applying *unary operations* (i.e., one addition or one subtraction of a resource from any region) and ends by greedily applying a *move (binary) operation* (i.e., move a resource from one region to another). The U&Me algorithm defines a sequence of unary and move operations, such that conducting the sequence operations leads the U&Me solution. The sequence is called the **U&Me sequence**.

Despite the simplicity of the algorithm, we need a non-trivial multi-step analysis to prove the optimality of the algorithm. The correctness of the algorithm is based on

[7] In our work, a problem is called "hard" if a polynomial time solution for the problem implies the correctness of a well-known open problem, such as the P versus NP problem.

three major threads: 1) The U&Me sequence does not contain canceling operations (i.e., U&Me does not add and remove a resource from the same region). 2) For every valid repositioning of the resources, there exists a sequence of operations corresponding to the repositioning. The sequence possesses properties similar to a U&Me sequence and thus it is called a **U&Me-like sequence**. 3) We prove that conducting the operations of the U&Me sequence is more profitable than every other repositioning, based on the properties of the U&Me-like sequences.

We conduct a numerical study to explore the applicability of the algorithm. We demonstrate that near optimal allocations can be obtained, even for systems with a rigid constraint of the number of repositions they can carry; we show that by repositioning a small portion of the maximum number of repositions (e.g., 60% of the reposition limit) can yield a near-optimal placement (90% of the optimal profit). This allows operators to reach a near-optimal placement without having to sustain a significant overhead due to repositions.

1.4.2 The multi-type Constrained-Reposition Placement Problem (ConRePP-2) [1]

In Chapter 10 we describe the multi-type Constrained-Reposition Placement Problem (ConRePP-2), based on the results published in [1]. We prove that this problem is hard. Specifically, we show that if there is an optimal polynomial algorithm for the Constrained-Reposition Problem[8] – then there is a polynomial algorithm to the Exact Perfect Matching Problem[9]. The question whether the Exact Perfect Matching problem has a polynomial time solution has been considered to be an old open problem in graph theory for almost 30 years ([19–21]). One may, therefore, conjecture that there is no polynomial time solution for ConRePP-2.

We therefore develop three polynomial heuristic algorithms to solve the problem in polynomial time: the first is called the Shortest Cycle-Canceling (SCC) algorithm, which is based on the min-cost flow reduction presented for the unconstrained-capacity static placement problem (SPP-4). SCC is a variant of the well-known Cycle-Canceling methodology, which can be used to solve the min-cost flow problem. The second is called the Hybrid algorithm that uses SCC. The algorithm performs small resource repositioning when the demand changes are small and conducts large resource repositioning when the changes are large. Finally, we present an algorithm called SCO that guarantees

[8] Polynomial in the reposition constant r, the number of resource types m and the number of regions k.

[9] Given a graph where edges are colored in red or blue and a parameter k, the Exact Perfect Matching Problem asks for a perfect matching that has exactly k red edges; a perfect matching is a set of edges covering all vertices such that no two edges share a vertex.

to find the optimal solution in several important special cases. We show that one can implement SCO in polynomial time, using techniques in graph theory and min-cost flow.

We use a numerical analysis to evaluate our algorithms under practical workloads and conditions, i.e., using Amazon's EC2 on-demand image costs and realistic demand arrival rates. Our results demonstrate that the proposed Hybrid algorithm can achieve a near-optimal placement (its profit is at least 98.7% of the unconstrained optimal placement profit), while maintaining a low reposition cost[10]. In particular, the reposition cost of the proposed Hybrid algorithm is significantly lower than that of the optimal placement (more than 65%) as well as that of a proportional mean-based placement scheme [11] (wherein the number of servers is proportional to the average demand).

1.5 Model Extensions [1]

In Chapter 11 we present further model extensions, such as placement problems where the reposition cost is integrated into the profit function, the parameterized placement problem and others. We show that these problems can be solved using the algorithms proposed in this thesis. The results were published in [1].

1.6 List of Papers

In this work, we published two articles in peer-reviewed proceeding conferences [12, 13], one article in a peer-reviewed journal [1], and one submitted article [18]. In addition, this thesis contains unpublished results [14, 22, 23] that will be submitted to peer-review journals.

For the sake of presentation, a list of model symbols can be found at page x, and a list of problems can be found at page xii.

[10] The reposition cost is defined as the number of repositions made.

Chapter 2

Related Work

The problem of resource placement in distributed systems depends highly on three settings: the application, the modeling and the objective function one considers optimizing. Generalized problems that are suitable for modeling multiple mathematical applications are not common. The generality of our formulation, in contrast, allows us to compare our work to different dynamic resource placement problems spanning from virtual machine to VoD P2P placement problems.

When comparing the time complexity of our work to other studies in the field, it is important to emphasize that while our proposed algorithms run in polynomial time by the number of resources the operator places or reposition, it might be pseudo-polynomial[1] in the size of the thin input. Resource placement problems that are stochastic, goe-distributed, optimizing a general convex function are usually strongly NP-hard problems[2], and thus they are solved using heuristics or approximations, as opposed to our optimal effective solutions. Note that in Remarks 1.2, 1.3 in the introduction chapter (Chapter 1), we explain why our algorithms are considered to be efficient, although they are pseudo-polynomial.

2.1 Cloud-related Problems- VM Placement

Early work on VM placement e.g., [24, 25], models the problem as a bin packing problem, namely, for every service there is an estimate of a deterministic demand. The deterministic problem can be generalized into the Stochastic Bin-Packing (SBP) problem, where

[1] A **pseudo-polynomial algorithm** is one that its running time is polynomial in the numeric value of the input, but is exponential in the length of the binary-encoded input.

[2] A problem is called **strongly NP-hard** if it is proven that it cannot be solved by a pseudo-polynomial time algorithm unless $P = NP$.

the problem is formulated in different ways. For example, [26, 27] formulated the problem as follows: given a set of random items, find the minimum number of bins such that each bin overflow probability will not exceed a given value. In addition, Shabtai at el. [28] suggested the Stochastic Packing with Minimum Expected Deviation (SP-MED) and Stochastic Packing with Minimum Overflow Probability (SP-MOP), giving optimal fractional solutions to these problems. As the deterministic bin packing problem is NP-hard (and even strongly NP-hard [29]), so are stochastic studies on that matter [26–28]. For more information on VM placement we refer to Raz et al. [30], which studies the placement of VM that implements services over the physical infrastructure, trying to understand what makes a placement scheme better than others in the overall utilization of the various resources. The solutions proposed in those works are heuristics or approximations that might be not be optimal in some scenarios, while our work suggests optimal solutions[3].

2.2 Cloud-related Problems: Deterministic Geo-distributed Problems

With the growth of cloud computing and large-scale dynamic services, the problem of dynamic server placement in a geo-distributed environment (by service provides) has received increasing attention. For example, [9] focused on dynamically optimizing service placement while ensuring performance requirements; and [10] studied algorithms for dynamic scaling of social media applications. Other works looked at the problem from a cloud provider's viewpoint of how to assign a newly provisioned virtual machines to distributed datacenters [31, 32]. These works typically assume that demands are approximately deterministic, and develop an optimization problem (based on deterministic inputs), which is solved periodically. As the demand in geo-distributed settings can be highly variable, the deterministic approach can lead to inaccuracies in the placement and operating costs. In contrast, we provide algorithms that inherently account for the full demand distributions and for repositioning costs, yielding a cost-effective solution better geared for modern distributed settings.

Other deterministic geo-distributed studies have focused on minimizing the energy load of virtual machines hosts by using live migration and detecting hosts with high and low utilization [33–36]. Similar to VM placement problems, the solutions proposed in these works are heuristics that might not be optimal [36], while our work suggests optimal solutions.

[3] Except for the heuristics provided in Chapter 10.

2.3 Cloud-related problems: Stochastic Geo-distributed Problems

There are several works in virtual network embedding, i.e., allocating network resources to given stochastic virtual network requests in a geo-distributed network [37–41]. Similar to our works, the goal is to grant as many requests as possible to maximize a complex revenue-based function, and they are using flow algorithms in their solution. However, these problems are strongly NP-hard problems by a reduction to the Unsplittable Flow Problem [41, 42]. These problems are solved via heuristics, such as coveting a mixed integer programming problem by a linear programming problem [40]. In addition, these works ignore migration and reposition costs, as opposed to our works in Chapters 8-11.

Other policies for server allocation look at the long-term horizon given a predicted stochastic demand [43, 44], and stochastic load balancing of VMs in datacenters [45]. They use heuristics to compute an efficient (in time) solution. Further, [43, 44] do not capture the reposition cost overhead, and thus allow for a large number of repositions, a costly task in practice. In addition, [45] did not capture different service revenues or operational costs of placing these VMs, as we do.

Similar models to ours were used to optimize server and task scheduling in cloud settings [46–48]. In these works, every server (or task) execution has a start and finish time. In [46, 48] the objective function is to ensure the stability of the system given the stochastic load, while in our case our goal is to find a solution that maximizes the excepted profit. While [47] considered finding a solution that minimizes the energy cost, it proposed heuristics, and does not capture the reposition cost overhead.

Finally, Sherzer et al [49], instead of considering a deterministic supply, assumes that not all placed supply can be assigned due to malicious attacks, or occasional failures. As it turns out, although the supply is now stochastic, under some supply distributions[4], both the placement and the assignment problems can be solved similarly as in this thesis. They provide a sufficient and necessary condition for these distributions. Yet, if the condition is violated, then, it is possible to solve the placement problem via dynamic programming in higher time (can be exponential in the number of resource types, while ours is polynomial in the number of resource types).

[4]By supply distribution we mean the number of resources that survived an attack or a failure, given the number of placed resources.

2.4 Peer-to-Peer (P2P) Works

Other works studied resource replication in P2P systems. For example, Tewari and Kleinrock [11] suggested an exponentially expanding topology for file sharing systems. It proved the optimality of proportional replication i.e., one based on the mean demand, under the assumption of abundant storage and upload capacity where all possible requests are always served. In contrast, our model allows for restricting the number of resources (files). Others, like Zhou et al. [50] proposed an optimal P2P VoD replication algorithm across peers, called RLB. Most of these works have focused on static allocations that account mainly for service costs, while others did consider dynamic settings (e.g., [51]), but paid little attention to resource placement and reconfiguration overhead. In addition, [50] focused on small-scale networks, and do not consider geo-distributed topologies as we do. In Section 5.7 we compare the performance of our optimal placement to the performance of the policies of [11, 50].

Other relevant works such as [52] proposed placement framework for large-scale VoD service based on a mixed integer program. While their model accounts for arbitrary demand pattern and network structure, it assumes deterministic demand, whereas we consider stochastic one. Other works in this domain often focus on different goals. For example, [53] optimized file availability when peers are infrequently online, while [54] aimed to minimize the number of access failures under random assignment policy. Consequently, they establish different replication rules compare to ours, such as log proportional and square root.

2.5 Other Resource Placement Problems

The problem of resource placement in distributed systems often falls under facility location theory [55, 56]. This area has received significant attention from the viewpoint of both analysis and algorithmic solutions. While the many traditional facility location problems are formulated given a deterministic demand (such as [57, 58]), others consider stochastic demands (see a survey of these problems at [59]). Most of the stochastic facility location problems are (strongly) NP-hard since they often generalize classical facility location problems, which are NP-hard [59] themselves. In addition, in this domain "very few exact algorithms are available for general problems on general networks; most apply only to specially structured problems with limited potential for direct application" [59]. Our problem, on the other hand, solves a generalized and stochastic problem in relevantly low polynomial time.

Our problem is also applicable in a wider operation research studies. Herer et al [60] presented the multi-location transshipment problem, which uses a modeling similar to ours. They suggested a supply chain, which comprises several retailers and one supplier was considered. They analyze the benefits of transshipment considering several retailers, which can differ in their cost and demand parameters. Within successive time periods, it is possible to move items from one retailer to another in order to answer the stochastic demand. They proved that in order to minimize the expected long-run average cost for this system, an optimal replenishment policy is for each retailer to follow an order-up-to S policy. However, finding a solution that accounts for all demand distributions is a hard task, and thus they have used Monte Carlo-based heuristics; in our case, we use optimal solution account for all demand distributions. Other works on the multi-location transshipment problem have used heuristics and not optimal solutions (such as [61–63]).

The problems of product allocation are related to resource placement in cloud computing. For example, [64] considered the problem of which products are to be built at which plants, given a stochastic future product demand. They proposed several principles on how to use benefits of process flexibility; that means, what different types of products a manufacturing plant should create. Their objective function differs from ours, and the problem they present is hard to solve analytically in polynomial time. The problem of static bicycle repositioning problem (SBRP), presented by Raviv et al. [65] considers the problem of repositioning bicycles in bike-sharing systems. SBRP incorporated reposition costs into a convex penalty cost that incorporates the stochastic demand (thus, it is similar to our problem solved in Section 11.1), under different constrains. They suggested two different mixed integer linear program (MILP) formulations for the problem. Since the mixed integer linear problem is (strongly) NP-hard, they used heuristics enhancements to solve the problem. In our case, we suggest optimal solutions.

2.6 Resource Placement Problems: Stochastic and Geo-distributed Problems, a (Pseudo-)Polynomial Time Solution

To our knowledge, there are few resource placement works where (1) a convex cost function (or concave profit function) incorporating stochastic demand is considered, and (2) a (pseudo-)polynomial solution is given. A very recent work, Freund et al. [66], which is independent of ours [18] (presented in Chapter 9), considers the problem of how to allocate dock-capacity in bike-sharing systems, by minimizing the expected number of out-of-stock events. The algorithm they proposed is similar to our work in Chapter 9.

There three major differences between the works, as follows: (1) [66] have used a different reposition constraint (which they have called *operational constraints*), which is measured differently. (2) [66] have capacity bounds over the number of resources (in their case, bikes and docks) in place in every region, bounds over the number of bikes and docks totaled over all regions, and bounds over the number of bikes totaled over all regions. In our work (in Chapter 9), we do not assume these assumptions, although we can capture these capacity bounds in our modeling (See Example 3.3 in Chapter 3). Moreover, the time complexity of the algorithm proposed in [66] depends on these capacity bounds, and in particular, if the capacity bound approaches to infinity, our algorithm run faster. (3) The algorithm they proposed have use only move operations, while in our case we have used in addition unary operations. We think that in some particular cases, our problem and solution can be used to solve the problem they have proposed.

Chapter 3

The Model Framework

In this chapter, we present the model framework used in our thesis. Although we focus our examples on a SaaS provider that rents servers from a cloud provider, the generality of our model allows solving a variety of resource placement problems spanning from peer-to-peer based (P2P) VoD systems (as done in [12], whose results are published in Chapters 4, 5) to personnel placement in call centers. A list of model symbols can be found at page x, and a list of problems can be found at page xii.

3.1 The Model

We consider a provider that aims to maximize the profit from servicing requests by placing resources in k **regions** (also called **areas**) indexed by $1, 2, \ldots, k$. The resources have different **types** indexed by $1, 2, \ldots, m$, and a type i resource can only grant up to B_i type i requests. Requests for these resources can be granted locally by a resource in the same region or granted remotely from a different region.

The demand (number of requests) is stochastic and follows an arbitrary (non-negative integer) random variable. The demand is denoted by a multi-dimensional random variable $D = \{D_i^j\}$, where D_i^j is a random variable that represents the number of requests made at region j for type i resources for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, k$. We do not assume any correlation between the random variables D_j^i, neither make any assumptions regarding their distributions. The provider's resource placement is denoted by $L = \{L_i^j\}$ where L_i^j is the number of type i resources placed in region j for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, k$.

The profit is a function composed of the revenue of granting demand minus the cost associated with placing and operating the resources. The profit functions are determined by

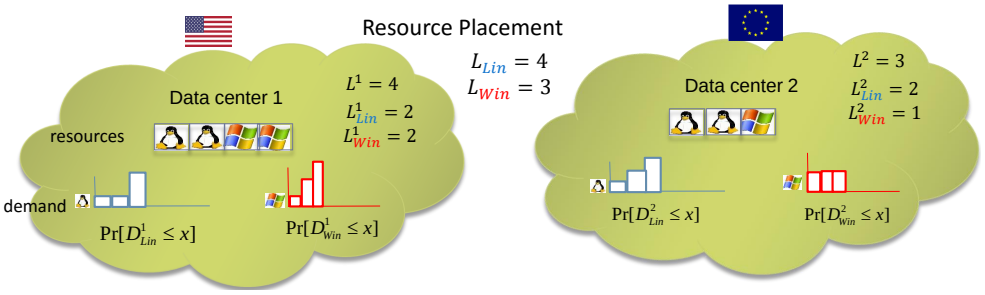


FIGURE 3.1: An example of the system. The placement quantities are $L^1_{Lin} = L^2_{Win} = 2$, $L^2_{Lin} = 2$, $L^2_{Win} = 1$ etc.

the given arbitrary demand D and the provider's resource placement L for $i = 1, 2, \ldots m$ and $j = 1, 2, \ldots k$. An example of the system topology and placement (of Windows and Linux resources) in response to user demand is depicted in Figure 3.1.

A major challenge of the profit function is to capture both the revenue differentiation between serving local and serving remote requests, as well as the complex costs associated with placing and operating the resources in these regions. Fortunately, as we will show below, we can model the cost differential between servicing remote and local requests (as well as optimizing this service); further, for any realization d of D, we can use a variable transformation that makes the analysis feasible through a semi-separable function. The function form is

$$P(L, d) = \sum_{i=1}^{m} \sum_{j=1}^{k} \zeta_i^j(L_i^j, d_i^j) + \sum_{i=1}^{m} \zeta_i(L_i, d_i) + \sum_{j=1}^{k} \zeta^j(L^j), \tag{3.1}$$

where $d_i = \sum_{j=1}^{k} d_i^j$, $L_i = \sum_{j=1}^{k} L_i^j$ are respectively the number of type i requests and type i resources from all regions. The notation $L^j = \sum_{i=1}^{n} L_i^j$ denotes the number of region j resources of all types. The functions ζ_i, ζ^j and ζ_i^j are called the marginal profit functions, or, in short, **the marginal functions**.

The function ζ_i represents the basic profit composed of the revenue of serving type i requests, regardless of serving them locally or by a remote resource (i.e., the requests are served **globally**), minus the cost of operating or manufacturing type i-resources. We assume that the profit gained by satisfying a type i request is a constant $R_i \ge 0$, called the **global revenue constant**. The number of satisfied type i requests is the minimum between the demand d_i of type i requests and the supply, i.e., the total number of requests that can be granted by type i resources, $L_i \cdot B_i$ (B_i – the number of requests a type i resource can serve concurrently). This follows a common structure of the profit as the difference between the revenue and the non-negative **operational cost function** $C_i : \mathbb{N} \to \mathbb{R}^+$, as shown by the following equation:

$$\zeta_i(L_i, d_i) = R_i \cdot \min(L_i \cdot B_i, d_i) - C_i(L_i). \tag{3.2}$$

The function ζ_i^j is the additional local profit derived from allocating resources in a specific region, representing the avoidance of extra bandwidth and latency costs of remote service or any other region-specific benefits and expenses. It is composed of the additional revenue function of granting requests locally, and the (non-negative) additional cost inflicted by placing a resource in a specific region. It follows a similar formula to Eq. (3.2):

$$\zeta_i^j(L_i^j, d_i^j) = R_i^j \cdot \min(L_i^j \cdot B_i, d_i^j) - C_i^j(L_i^j), \tag{3.3}$$

where $R_i^j \geq 0$ is the profit of granting a type i request in region j, called the **local revenue constant**.

Finally, the function ζ^j represents only the (non-negative) regional costs C^j associated with the placement, such as physical limitations over the number of placed resources at a region and regional taxes. The function form is expressed as follows:

$$\zeta^j(L^j) = -C^j(L^j). \tag{3.4}$$

As stated above, the combination of Eqs. (3.1), (3.2), (3.3) and (3.4) allows for modeling the cost differentiation between local and remote service of requests and capturing the different costs. While the cost depends on the placement and is not affected by demand, the revenues are affected. It leads us to the next challenge of defining a profit function for this setting.

We must capture the fact that the profit from any allocated resource is stochastic, since the demand is stochastic. For this purpose, we describe the profit as the expectation of P (from Eq. (3.1)) where the expectation is over a demand distribution D. Substituting this into in Eqs. (3.2), (3.3), (3.4) the profit function in Eq. (3.1) gets the following form:

$$P(L, D) = \sum_{i=1}^{m} \sum_{j=1}^{k} \zeta_i^j(L_i^j, D_i^j) + \sum_{i=1}^{m} \zeta_i(L_i, D_i) + \sum_{j=1}^{k} \zeta^j(L^j), \tag{3.5}$$

where,

$$\zeta_i^j(L_i^j, D_i^j) = R_i^j \cdot E_{D_i^j}[\min(L_i^j \cdot B_i, D_i^j)] - C_i^j(L_i^j), \tag{3.6}$$

$$\zeta_i(L_i, D_i) = R_i \cdot E_{D_i}[\min(L_i \cdot B_i, D_i)] - C_i(L_i), \tag{3.7}$$

$$\zeta^j(L^j) = -C^j(L^j), \tag{3.8}$$

are the marginal profit functions.

Remark 3.1. Equations (3.1)-(3.4) correspond to the profit formulation with respect to a realization (deterministic) demand, and (3.5)-(3.8) to any (stochastic) demand.

3.1.1 List of problems

As recalled from the introduction (Chapter 1), the Static Placement Problems and the Constrained-Reposition Placement Problems are defined as follows:

A Static Placement Problem (SPP)	
Input:	A probabilistic forecast of the demand $D = D(t)$ at period t.
Problem:	Find the placement L that maximizes the profit $P(L, D)$.

The static placement problems are discussed in Chapters 4–7.

A Constrained-Reposition Placement Problem (ConRePP)	
Input:	Demand $D(t+1)$ of time $t+1$, placement $L(t)$ of time t, reposition bound r.
Problem:	Find the placement $L(t+1)$ that maximizes $P(L(t+1), D(t+1))$ under the reposition constraint $\sum_{i=1}^{m}\sum_{j=1}^{k} \lvert L_i^j(t) - L_i^j(t+1)\rvert \leq r$.

An example of resource reposition in response to the demand change is given in Figure 3.2. ConRePP is solved in Chapters 9, 10.

3.1.2 Model extensions and other problems

In Chapter 11, we extend our modeling and present two additional problems: 1) we formulate the placement problem where reposition costs are part of the profit function. 2) We present an extension where un-served demands incur a positive cost on the profit. For the sake of presentation, these problems are formulated in the appropriate chapter.

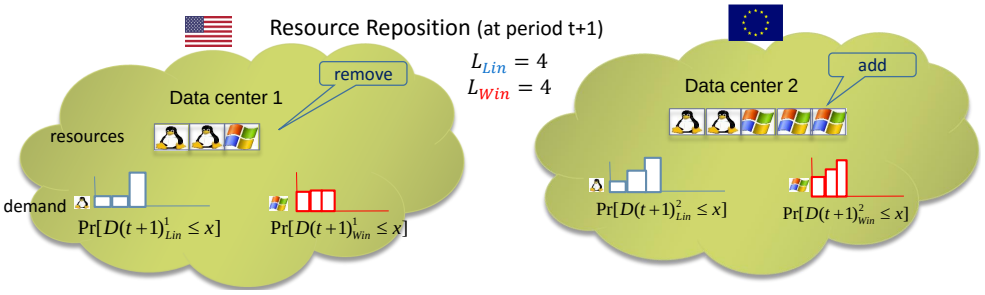


FIGURE 3.2: Reposition of resources in response to the demand change: Remove one Windows resource from region 1 and add two Windows resources in region 2. The initial placement $L(t)$ is given in Figure 3.1.

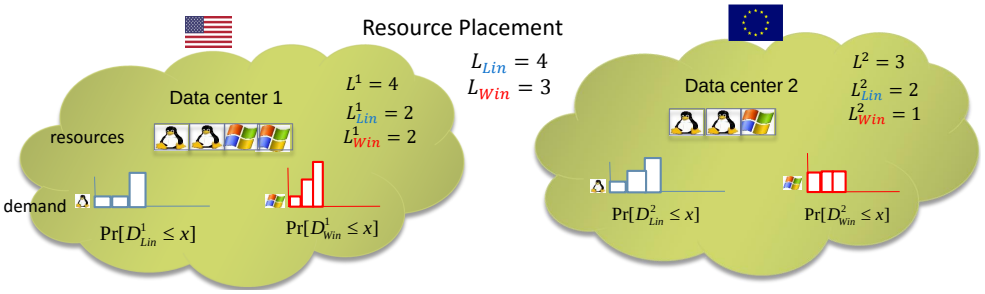


FIGURE 3.1: The initial placement $L(t)$ (repeated from page 19).

3.2 Model Assumption – Convexity of Marginal Functions

Apart from the above general profit function structure, we assume that given a demand D, the **conditional-marginal functions** $g_i(n) = \zeta_i(n, D_i^j), g_i^j(n) = \zeta^j(n, D_i^j)$ and $g^j(n) = \zeta^j(n)$ are all concave functions in n. That means, the differential of these functions, $\Delta g(n) = g(n) - g(n-1)$, which are called the **marginal-differential** functions, are monotonically non-increasing for $n \geq 1$.

This is a reasonable assumption as the profit may express diseconomy of scales; this mainly results from the fact that communications and control may become more difficult to handle as the number of resources grows. In addition, the revenue parts of the profit, i.e., the expected value of the minimum multiplied by a positive constant (see left terms in the right hand side of Eqs. (3.6), (3.7)) is concave, as we next show:

Theorem 3.1. *Let n, b be non-negative integers, and let D be a non-negative integer valued random variable. Then we have:*

$$E_D(\min(D, n \cdot b)) = \sum_{i=1}^{n \cdot b} \Pr(D \geq i). \tag{3.9}$$

Therefore, the function $f(n) = E_D(\min(D, n \cdot b))$ is concave.

Proof. The cumulative distribution function of the minimum is presented below:

$$\Pr[\min(D, n \cdot b) \geq k] = \begin{cases} \Pr(D \geq k \cdot b) & k \cdot b \geq n \\ 0 & n \cdot b < k. \end{cases} \tag{3.10}$$

We know that the expected value of a non-negative integer valued random variable Y is equal to $E(Y) = \sum_{i=1}^{\infty} \Pr(Y \geq i)$ (A proof can be found in [67]). Substituting Y by $\min(D, n \cdot b)$ in this equality and using Eq. (3.10) yields Eq. (3.9).

To show that $f(n)$ is concave, we note that $f(n) - f(n-1) = \sum_{i=(n-1)\cdot b+1}^{n \cdot b} \Pr(D \geq i)$ is a sum of b elements that are decreasing as n is increasing. Thus, $f(n) - f(n-1)$ is monotonically decreasing, and $f(n)$ is concave.

□

Thus, the assumption of convexity holds in cases where the operational costs $C()$ are convex, and in particular are linear (as we will see later in Example 3.2).

3.3 Applications of our Modeling

The generality of our modeling enables us to capture a variety of different system settings. Examples are given next:

Example 3.1 (Systems with generalized demand setting). Our model is given an arbitrary demand distribution, and does not assume any correlation between the arrivals of the requests. This allows us to express real-life systems where the demand distribution is not necessarily taken from a family of well-known distributions (such as Poisson or Binomial).

Example 3.2 (Systems with linear operational costs). Our model can capture a variety of costs such as cloud rental plans. For example, suppose the on-demand server pricing of cloud providers such as EC2 can be captured as $p_i^j \geq 0$, a fixed price for running type i server in region j, yielding an operational cost function of $C_i^j(L_i^j) = L_i^j \cdot p_i^j$. A more complicated example can incorporate software licensing costs and region-specific charges. Let $r_i \geq 0$ be the licensing cost associated with type i server and $h^j \geq 0$ be an add-on provisioning cost for region j. Then, the operational cost functions are:

$$\begin{aligned} C_i^j(L_i^j) &= p_i^j \cdot L_i^j \\ C_i(L_i) &= r_i \cdot L_i, \\ C^j(L^j) &= h^j \cdot L^j. \end{aligned}$$

These operational cost functions are all linear functions. Since each marginal profit function $\zeta_i, \zeta_i^j, \zeta^j$ is equal to a concave function (the expected value of a minimum) minus a linear function, then these functions are all concave, regardless of the demand distribution D.

Example 3.3 (Capacity-constrained systems). Our model system allows us to capture **capacity-constrained systems** where the number of resources placed in a region is bonded, due to physical limitations. That means that there exists a well-known **storage value** s^j such that the operator can place up to $L^j = \sum_{i=1}^{m} L_i^j \leq s^j$ resources in region j. To do this, we set the region j operational cost function to be $C^j(x) = \infty$ for $x \geq s^j + 1$.

3.4 Simplistic Profit Function

The problems presented later at Chapters 4-6 consider a simplistic profit function $P()$ that satisfies the following properties: 1) The type and area+type operational cost functions $C_i(), C_i^j()$ are set to zero. 2) The area operational cost function C^j is set to represent capacity-constrained systems (see Example 3.3). Formally, for every region j, there is a storage value s^j such that $C^j(x) = 0$ for $x \leq s^j$ and $C^j(x) = \infty$ for $x > s^j$. 3) The resource capacity for every resource type is set to one (i.e., $B_i = 1$ for every resource type i). 4) All global and local revenue constants are equal to one another (i.e., there exist constants R_{loc}, R_{glo} such that $R_i^j = R_{loc}$ and $R_i = R_{glo}$).

We define a placement L to be a **valid** placement in a capacity-constrained system, if for every region j, the number of resources that L contains in that region is less than the corresponding storage value, i.e., $L^j \leq s^j$. For every demand D, the profit for every valid placement L equals

$$P(L, D) = \sum_{i=1}^{m} \sum_{j=1}^{k} R_{loc} \cdot E_{D_i^j}[\min(L_i^j, D_i^j)] + \sum_{i=1}^{m} R_{glo} \cdot E_{D_i}[\min(L_i, D_i)],$$

and for every invalid placement the profit is $-\infty$. For sake of simplicity, the problems at Chapters 4-6 consider only valid placements.

Note that the profit function is composed of marginal convex functions, according to Theorem 3.1.

Chapter 4

The Single-Region System Static Placement Problem (SPP-1)

In this chapter, we solve the placement problem over a simplistic system; this is used to expose the reader to the problem formulation and the principles of our analysis. That is, solving the Static Placement Problem given a capacity-constrained single-region system, optimizes the simple profit. The problem was published and solved in [12].

4.1 Problem Formulation

Let $D = \{D_i\}$ be the demand for type i resources in the region. Since we are interested in the placement problem over capacity-constrained systems, we assume that the operator can place up to s resources in the region. A placement L that contains up to s resources is called a **feasible** placement. Formally, our goal is to find a placement $L = \{L_i\}_{i=1}^{m}$ under the constraint $\sum_{i=1}^{m} L_i \leq s$ that maximizes the simple profit, which in a single-region system is equal to

$$P(L, D) = \sum_{i=1}^{m} R_{loc} \cdot E_{D_i^1}[\min(L_i^1, D_i^1)] + \sum_{i=1}^{m} R_{glo} \cdot E_{D_i}[\min(L_i, D_i)]$$
$$= (R_{loc} + R_{glo}) \cdot \sum_{i=1}^{m} E_{D_i}[\min(L_i, D_i)], \quad (4.1)$$

where $R_{loc}, R_{glo} \geq 0$ are non-negative revenue constants, under the constraint $\sum_{i=1}^{m} L_i \leq s$. Formally, the problem is as follows:

THE STATIC PLACEMENT PROBLEM FOR A SINGLE-REGION SYSTEM (SPP-1)	
Assumptions:	Single-region, capacity-constrained system optimizing simple profit.
Input:	The demand $D = \{D_i\}$ for type i resources, the total storage value s.
Problem:	Find the placement $L_{opt} = \{L_i\}_{i=1}^{m}$ that maximizes $\sum_{i=1}^{m} E_{D_i}[\min(L_i, D_i)]$ under the capacity constraint $\sum_{i=1}^{m} L_i \leq s$.

4.2 Max Percentile Solution

Our key result is the observation that the expected value $E_{D_i} \min(L_i, D_i)$ can be transformed into a simple sum of probabilities using Eq. (3.9):

$$E(\min(L_i, D_i)) = \sum_{j=1}^{L_i} \Pr(D_i \geq j). \tag{4.2}$$

The objective of the placement problem is therefore to maximize the function

$$\sum_{i=1}^{m} E_{D_i}[\min(L_i, D_i)] = \sum_{i=1}^{m} \sum_{j=1}^{L_i} \Pr(D_i \geq j), \tag{4.3}$$

by selecting the set $L = \{L_i | 1 \leq i \leq m\}$ under the constraint that $\sum_{i=1}^{m} L_i^1 \leq s$, i.e., one places up to s resources in the region.

The structure of this equation serves as a key for the analysis as well as to its interpretation. The key property of this equation is that adding a resource of type i to placement L given in the above equation will increase the expected profit by $\Pr(D_i \geq L_i + 1)$. We thus may define a "delta" function $\delta_i()$ whose values are $\delta_i(l) = Pr(D_i \geq l)$. As such, the objective function is to select a placement L that maximizes $\sum_{i=1}^{m} \sum_{j=1}^{L_i^1} \delta_i(j)$ under the constraint $\sum_{i=1}^{m} L_i^1 = s$.

Having this view in mind, we can now turn to describe our Max Percentile algorithm, whose description may be assisted by Figure 4.1, which depicts an example of three resource types (and therefore three $\delta()$ vectors). The (Percentile) algorithm is carried out by going over the $\delta_i()$ vectors and using a greedy approach to select their highest values. This is easily done by observing that $E(\min(L_i, D_i))$ is concave and each $\delta_i()$ vector is monotonically non-increasing (as in Figure 4.1). Thus, the algorithm is carried out by keeping three types of $\delta_i()$ variables: 1) already selected variables (dark background, on the left hand side); 2) candidates for selection – one element for every resource type i (light background); and 3) non-examined variables (white background, on the right

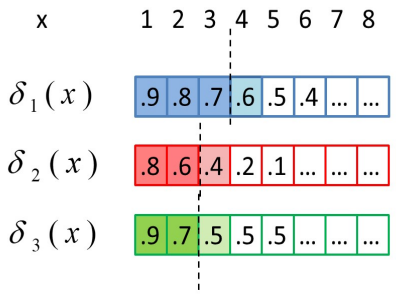


FIGURE 4.1: The Max Percentile "delta" vectors

hand side). Using this classification the $\delta_i()$ selection algorithm progresses by selecting the maximal value element from the set of candidates (4^{th} element in first vector in the example) and then adding the element appearing to the right of the selected candidate (5^{th} in the first vector) to the candidate set. The algorithm terminates when the number of placed resources equals the total storage value, s.

Theorem 3.1 implies that $\delta_i()$ is a monotonically non-increasing vector, and thus one can easily prove that the percentile algorithm indeed selects the s highest values of the $\delta_i()$ vectors. This completes the Max Percentile algorithm, which will return, as a solution to the placement problem, the selected set L where L_i^1 is equal to the number of elements selected from the $\delta_i()$ vector.

4.2.1 Implementation and complexity

We discuss the implementation of the max-percentile in the next chapter. It can be implemented using a maximum priority queue in $O(s \log m)$ time, where s is the total storage and m is the number of resource types.

4.2.2 Performance Evaluation

We use numerical analysis and simulations to evaluate a single-region system later in Section 5.7, Figures 5.1, 5.2.

Chapter 5

The Homogeneous-Region System Static Placement Problem (SPP-2)

In this chapter, we extend the analysis from a single-region system (Chapter 4) and solve the problem given a capacity-constrained and homogeneous-region system, optimizing a simple profit function. The solution in this chapter, called Multi-Region Max Percentile (MuRMaP), was published in [12] and uses the max percentile principle described in the single-region system solution.

5.1 Problem Formulation

Let $D = \{D_i^j\}$ be the demand for type i resources in region j. In this chapter, we solve the placement problem given a homogeneous-region capacity-constrained system. Thus, the system is assumed to have the following properties: 1) The system can hold up to $\frac{s}{k}$ resources in every region, where s is the total storage divisible by the number of regions k. 2) The regional demands are statistically identical. That is, for every resource type i the random variables $D_i^1, D_i^2, \ldots, D_i^k$ are identically distributed; let $\tilde{D}_i$ denote a generic variable having the same distribution.

A placement $L\{L_i^j\}_{i=1}^m$ is called a **feasible** placement if it represents a valid resource placement of the system, i.e., $L^j = \sum_{i=1}^m L_i^j \leq \frac{s}{k}$ in every region j. To this end, we define our placement problem as finding a feasible placement L that maximizes the

simple profit $P(L, D)$, which is equal to

$$P(L,D) = \underbrace{\sum_{i=1}^{m}\sum_{j=1}^{k} R_{loc} \cdot E_{D_i^j}[\min(L_i^j, D_i^j)]}_{\text{Local}} + \underbrace{\sum_{i=1}^{m} R_{glo} \cdot E_{D_i}[\min(L_i, D_i)]}_{\text{Global}}, \tag{5.1}$$

where $R_{loc}, R_{glo} \geq 0$ are non-negative revenue constants. The placement problem in this chapter is formulated as follows:

PLACEMENT PROBLEM FOR A HOMOGENEOUS-REGION SYSTEM (SPP-2)	
Assumptions:	Homogeneous-region system, capacity-constrained system, optimizing simple profit.
Input:	A demand $D = \{D_i^j\}$ where $D_i^1, D_i^2, \ldots, D_i^k$ are all identically distributed for every resource type i, the total storage value s (where s is divisible by the number of regions k).
Problem:	Find an optimal placement $L_{opt} = \{L_i^j\}$ that maximizes the profit in Eq. (5.1) under the capacity constraint $L^j = \sum_{i=1}^{m} L_i^j \leq \frac{s}{k}$.

5.1.1 The VoD P2P application

The original problem in [12] was formulated as a VoD P2P problem, where the resources are peer-servers. The peer-servers store a single movie, and can upload this movie to clients requesting it. For the sake of presentation, we use the terminology used in the modeling chapter (Chapter 3); the terminology refers to the peer-servers as resources, movies as resource types, and a peer containing movie i is equivalent to a type i resource.

5.2 Symmetrically-full Placements

The first key machinery we use in our analysis (of finding an optimal feasible placement) is to narrow down the space of placements to the space to *symmetrically-full placements*, as we defined next:

Definition 5.1. Let $L = \{L_i^j | \, 1 \leq i \leq m, \, 1 \leq j \leq k\}$ be a placement. If the number of resources in every region is equal to $\frac{s}{k}$ (i.e., for every region j the number of resources is equal to $L^j = \sum_{i=1}^{m} L_i^j = \frac{s}{k}$) then L is called a *symmetrically-full placement*.

To this end, we show that an optimal feasible solution must be a symmetrically-full placement:

Claim 5.1. *Let L be a feasible placement. Then there exists a symmetrically-full placement L' whose profit is no smaller than that of L, $P(L', D) \geq P(L, D)$.*

Proof of Claim 5.1. One can easily transform L into a symmetrically-full placement by adding $s/k - \sum_{i=1}^{m} L_i^j$ resources in each region j. It is easy to see that the profit cannot decrease due to adding resources. Thus, the claim follows. □

Thus, the optimal placement can be found within the class of symmetrically-full placements. On its face value, finding the optimal placement seems to be a difficult since the number of symmetrically-full placements is exponential, which may lead to exponential complexity. Fortunately, as derived in the next section, the optimal placement within the set of symmetrically-full placements possesses an important balancing property that we refer as *the balance principle*. This will allow us to drastically reduce the complexity of searching for an optimal policy from within a reduced set of placements, named *balanced placements*.

5.3 Balanced Placements

5.3.1 Introduction

As stated above, balanced placements aimed to serve a key role in reducing the search for the optimal placement. We define balanced placements as follows:

Definition 5.2. Let L be a placement. Then L is called a *balanced placement* if for every resource type i and every two regions j_1 and j_2 we have $|L_i^{j_1} - L_i^{j_2}| \leq 1$.

A striking result (proved in Theorem 5.2) is that for every arbitrary symmetrically-full placement L there exists a symmetrically-full balanced placement L_B, which is as good as L. This will serve a key result since it allows us to conduct the whole optimization over the (narrow) space of symmetrically-full balanced placements.

Theorem 5.2 (Optimality Theorem). *Given the total storage value s and the number of regions k, for every arbitrary symmetrically-full placement L there exists a balanced and symmetrically-full placement L_B with a higher or equal profit.*

Note that the above theorem is true regardless of the probability distribution of the demand, (D_i^j).

Before proving this theorem, we need to prove several properties of placements. The proof of Theorem 5.2 will be completed at the end of Section 5.3.3.

5.3.2 Properties of balanced placements

To prove Theorem 5.2 and demonstrate fundamental properties regarding balanced placements, we use the following definitions:

Definition 5.3. Let $L = \{L_i^j | \, 1 \leq i \leq m, \, 1 \leq j \leq k\}$ be a placement. Let $L_i := \sum_{j=1}^{k} L_i^j$ be the number of type i resources across all regions. The vector $\hat{L} = (L_1, L_2, \ldots L_m)$ is called the *quantity vector*. If the quantity vector satisfies $\sum_{i=1}^{m} L_i = s$ then $\hat{L}$ is called a *full quantity vector* and L is called a *full placement*.

The following two claims regarding balanced placements and their quantity vectors will be useful in this section and in Section 5.4:

Claim 5.3. *Let L be a balanced placement, whose quantity vector is $\hat{L} = (L_1, L_2, \ldots L_m)$. Then, the following properties hold:*

1. *In every region j either $\left\lfloor \frac{L_i}{k} \right\rfloor$ or $\left\lceil \frac{L_i}{k} \right\rceil$ resources of resource type i are allocated for $1 \leq i \leq m$ and $1 \leq j \leq k$.*
2. *The number of regions with $\left\lfloor \frac{L_i}{k} \right\rfloor + 1$ resources is equal to $L_i \bmod k$.*
3. *The profit $P(L, D)$ is equal to:*

$$\sum_{i=1}^{m} [R_{glo} E(\min(L_i, D_i)) + R_{loc} \sum_{j=1}^{r_i} E(\min(\left\lceil \frac{L_i}{k} \right\rceil, D_i^j)) + R_{loc} \sum_{j=r_i+1}^{k} E(\min(\left\lfloor \frac{L_i}{k} \right\rfloor, D_i^j))], \tag{5.2}$$

where $r_i = L_i \bmod k$.

Claim 5.4. *Given a quantity vector $\hat{L} = (L_1, L_2, \ldots L_m)$, the following properties hold:*

1. *There exists a balanced placement of quantity vector $\hat{L}$.*
2. *Every two balanced placements with the same quantity vector $\hat{L}$, have the same profit.*

The proofs of these claims are straightforward and presented in Appendix a. The next two claims are also important. We will use them in Section 5.3.3 and in Section 5.4:

Claim 5.5 (The tail-sum formula for expected minimum). *Let C be an integer constant, and let X be a non-negative integer valued random variable. Then we have:*

$$E(\min(X, C)) = \sum_{i=1}^{C} \Pr(X \geq i). \tag{5.3}$$

Proof of Claim 5.5. This claim follows immediately from Theorem 3.1 by setting $b = 1$. □

Claim 5.6. *Let L be a placement. Let j_1 and j_2 be regions such that $L_i^{j_1} \geq L_i^{j_2} + 2$. Let L' be a placement produced from L by moving a resource of type i from region j_1 to region j_2 (i.e., setting $L'^{j_1}_i \leftarrow L_i^{j_1} - 1$ and $L'^{j_2}_i \leftarrow L_i^{j_2} + 1$). Then the profit of L' cannot be smaller than the profit of L', i.e., $P(L', D) \geq P(L, D)$.*

Remark 5.1. Note that this claim *does not* directly imply Theorem 5.2 since the theorem deals with **symmetrically-full** placements. Suppose we use Claim 5.6 over a symmetrically-full unbalanced placement L. Then, after moving a resource of type i from region j_1 to region j_2, the resulting placement will be a placement that is not symmetrically-full. This is because the number of resources in regions j_1 and j_2 are $\frac{s}{k} - 1$ and $\frac{s}{k} + 1$, respectively.

Proof of Claim 5.6. Observe the profit function in Eq. (5.1). The operation (of moving the resource) does not change the quantity vector $\hat{L}$, as well as the global part of the profit. The operation changes the local part of the profit at the elements corresponding to resource type i in regions j_1, j_2 (i.e., $E(\min(L_i^{j_1}, D_i^{j_1}))$ and $E(\min(L_i^{j_2}, D_i^{j_2}))$). The change in the profit is equal to:

$$\begin{aligned}
&P(L', D) - P(L, D) \\
=&R_{loc}[E(\min(L_i^{j_1} - 1, D_i^{j_1})) + E(\min(L_i^{j_2} + 1, D_i^{j_2})) - E(\min(L_i^{j_1}, D_i^{j_1})) - E(\min(L_i^{j_2}, D_i^{j_2}))] \\
\underbrace{=}_{\text{Change order of summation}}&R_{loc}[\underbrace{E(\min(L_i^{j_1} - 1, D_i^{j_1})) - E(\min(L_i^{j_1}, D_i^{j_1}))}_{a} + \underbrace{E(\min(L_i^{j_2} + 1, D_i^{j_2})) - E(\min(L_i^{j_2}, D_i^{j_2}))}_{b}] \\
=&R_{loc}(a + b).
\end{aligned} \tag{5.4}$$

Using the tail-sum formula for expected minimum, i.e., Eq. (5.3), we derive that

$$a = E(\min(L_i^{j_1} - 1, D_i^{j_1})) - E(\min(L_i^{j_1}, D_i^{j_1})) = -\Pr(D_i^{j_1} \geq L_i^{j_1}), \tag{5.5}$$

and

$$b = E(\min(L_i^{j_2} + 1, D_i^{j_2})) - E(\min(L_i^{j_2}, D_i^{j_2})) = \Pr(D_i^{j_2} \geq L_i^{j_2} + 1). \tag{5.6}$$

According to our problem formulation, the random variables $D_i^{j_1}$ and $D_i^{j_2}$ are both identically distributed to a generic random variable $\tilde{D}_i$. It is given that $L_i^{j_1} \geq L_i^{j_2} + 1$, and therefore

$$a + b \underbrace{=}_{\text{Eqs. (5.5), (5.6)}} \Pr(D_i^{j_2} \geq L_i^{j_2} + 1) - \Pr(D_i^{j_1} \geq L_i^{j_1})$$
$$= \Pr(\tilde{D}_i \geq L_i^{j_2} + 1) - \Pr(\tilde{D}_i \geq L_i^{j_1})$$
$$= \Pr(L_i^{j_1} > \tilde{D}_i \geq L_i^{j_2} + 1) \geq 0. \quad (5.7)$$

Thus, from Eq. (5.4), $P(L', D) - P(L, D) = R_{loc}(a + b) \geq 0$, i.e., $P(L', D) \geq P(L, D)$ and the claim is follows.

□

5.3.3 Quantity-Equivalent placements

To prove Theorem 5.2 we will define and use a quantity-equivalence relation between placements.

Definition 5.4. Let L and L' be placements. We call them *quantity-equivalent placements* if they have the same quantity vector, i.e., if for every resource type i the placements L and L' contain the same number of type i resources, namely $L_i = L'_i$.

It is easy to see that the relation defined is an equivalence relation. The following lemmas will lead directly to the proof of Theorem 5.2:

Lemma 5.7. *For every arbitrary full placement (not necessarily symmetrically-full) L, there exists a balanced and symmetrically-full placement L_B, which is quantity-equivalent to L.*

Proof of Lemma 5.7. We use a constructive approach and propose the Balanced Spread algorithm that produces L_B, as follows:

Algorithm 1 Balanced Spread algorithm

Require: A full quantity vector, i.e., $\hat{L} = (L_1, L_2, \ldots L_m)$, where $\sum_{i=1}^{m} L_i = s$.

Ensure: A symmetrically-full balanced placement.

1: **for all** resource type i **do**
2: Place $\left\lfloor \frac{L_i}{k} \right\rfloor$ resources of type i in every region j
3: Set $r_i \leftarrow L_i - \left\lfloor \frac{L_i}{k} \right\rfloor \cdot k$ (Notice that $r_i = L_i \, mod \, k$).
4: **end for**
5: Place one additional resource of type 1 in each of regions $1, 2 \ldots, r_1$. Then, place one additional resource of type 2 in each of the regions $r_1 + 1, r_2 + 2 \ldots, r_1 + r_2$ and so on. Note that all region index counting is done modulo k, guaranteeing circular placement.

Denote L_B as the placement returned from a run of the Balanced Spread algorithm on the quantity vector of L. Below we present a proof that L_B is a symmetrically-full balanced placement, which is quantity-equivalent to L.

1. L_B *is symmetrically-full*– Step 2 places the resources evenly among the regions. As for Step 5, it spreads $\sum_{i=1}^{m} r_i = \sum_{i=1}^{m} L_i - \sum_{i=1}^{m} \left\lfloor \frac{L_i}{k} \right\rfloor \cdot k$ resources in a circular fashion among the regions. It starts allocating the first resource in region 1. Further, since the number of resources is $\sum_{i=1}^{m} L_i = s$ (L is a full placement), and s is divided by k, then the last resource placed by Step 5 will be in the k^{th} region. Thus, the Balanced Spread algorithm allocates the same number of resources evenly among the regions, which equals $\frac{s}{k}$.

2. L_B *is balanced* – The Balanced Spread in Step 2 algorithm places the resources evenly among the regions. In Step 5 an additional resource of type i is allocated to $r_i = L_i \, mod \, k < k$ regions. Therefore, for each resource type i, the number of type i resources of two different regions is differ from one another by one resource at most, i.e., L_B is balanced.

3. L_B *is quantity-equivalent to* L – The number of type i resources is equal to $r_i + k \cdot \left\lfloor \frac{L_i}{k} \right\rfloor = L_i$, which is exactly as in L.

□

Remark 5.2. The construction of the symmetrically-full balanced placement depends only on the quantity vector and not on how the original placement L places its resources in any region. The only requirement of the input is that it represents a quantity vector of a full placement, i.e., $\sum_{i=1}^{m} L_i = s$.

Lemma 5.8. *The profit of L_B (as defined in Lemma 5.7), is not smaller than that of L. That is $P(L_B, D) \geq P(L, D)$.*

A full-detailed proof can be and is in Appendix a. We use Claim 5.6 to show that for every unbalanced placement L there exists a balanced placement L' satisfying two properties: 1) L and L' are quantity-equivalent and 2) The profit of L' is not smaller than the profit of L, i.e., $P(L', D) \geq P(L, D)$. The placements L' and L_B, which are both balanced and quantity-equivalent, and thus they have the same profit ($P(L_B, D) = P(L', D)$) according to Claim 5.4. This implies $P(L_B, D) = P(L', D) \geq P(L, D)$ as required.

We finally complete the proof of the key theorem of this chapter:

Proof of Theorem 5.2. Given a symmetrically-full placement L, its symmetrically-full balanced placement, L_B, has a higher (or equal) profit than L, as stated in Lemma 5.8. □

5.4 The Multi-Region Max Percentile Algorithm

In this section, we describe the *Multi-Region Max Percentile algorithm* (in short, the *MuRMaP* algorithm), and prove that the algorithm constructs the placement with the maximum profit.

As shown in the previous section (Theorem 5.2), within the class of symmetrically-full placements the balanced placements are better than the unbalanced ones. Therefore, the search for an optimal placement is carried out in the (narrow) space of symmetrically-full balanced placements. The MuRMaP algorithm will therefore operate by finding the placements with the highest profit among these placements.

The next claim will calculate the increase of the profit when adding a new resource to the placement:

Claim 5.9. *Given placement L, inserting a resource of type i_0 to region j_0, will increase the profit by:*

$$R_{glo} \cdot \Pr(D_{i_0} \geq L_{i_0} + 1) + R_{loc} \Pr(D_{i_0}^{j_0} \geq L_{i_0}^{j_0} + 1).$$

The claim is straightforward and is in Appendix a.

According to Claim 5.4, we know that for every given quantity vector $\hat{L} = (L_1, L_2, \ldots L_m)$ there exists a symmetrically-full balanced placement L_B with a quantity vector $\hat{L}$. The profit of L_B, which is unique to the quantity vector, will be called the *balanced profit of*

the quantity vector $\hat{L}$ and be denoted by $P_B(\hat{L})$. By adding a resource to the quantity vector, we increase its balanced profit, as the next claim states:

Claim 5.10. *Let* $\hat{L} = (L_1, L_2, \dots L_m)$ *be a quantity vector and let* $\hat{L}'$ *be a quantity vector such that* $L'_{i_0} = L_{i_0} + 1$ *and* $L'_j = L_j$ *for all* $j \neq i_0$. *Then the increase of the balanced profit, i.e.,* $P_B(\hat{L}') - P_B(\hat{L})$, *is*

$$R_{glo} \cdot \Pr(D_{i_0} \geq L_{i_0} + 1) + R_{loc} \Pr(\tilde{D}_{i_0} \geq \left\lfloor \frac{L_{i_0}}{k} \right\rfloor + 1). \tag{5.8}$$

A proof of Claim 5.10 is presented in Appendix a, based on Claim 5.3.

Remark 5.3. Eq. (5.8) can be viewed as the marginal increment in the balanced profit of the quantity vector due to an increase in L_{i_0}. Thus, it can be written as a function $\delta_i(L_i)$ which expresses the contribution of resource i to the balanced profit of L_i. Eq. (5.8) becomes

$$\delta_{i_0}(L_{i_0} + 1) = R_{glo} \cdot \Pr(D_{i_0} \geq L_{i_0} + 1) + R_{loc} \Pr(\tilde{D}_{i_0} \geq \left\lceil \frac{L_{i_0} + 1}{k} \right\rceil), \tag{5.9}$$

where we used $\left\lceil \frac{L_i+1}{k} \right\rceil = \left\lfloor \frac{L_i}{k} \right\rfloor + 1$.

Corollary 5.11. *The balanced profit of quantity vector* $\hat{L} = (L_1, L_2, \dots L_m)$ *is identical to the integration of the marginal contribution* δ. *That is*

$$P_B(\hat{L}) = \sum_{i=1}^{m} \sum_{j=1}^{L_i} \delta_i(j). \tag{5.10}$$

This can be proven by a simple induction on $\sum_{i=1}^{m} L_i$.

According to the previous claim, we can construct a quantity vector to have the highest balanced profit among all the full quantity vectors. The vector, which is called the *MuRMaP quantity vector*, is constructed by the following greedy algorithm:

Algorithm 2 Deriving the MuRMaP quantity vector

1: Initiate a new minimum priority queue Q.
2: Initiate a quantity vector $\hat{L} = (L_1, L_2, \ldots L_m)$ such that $L_i = 0$ for every resource i.
3: **for all** resource i **do**
4: Insert to Q the value of $\delta_i(1)$.
5: **end for**
6: **repeat**
7: Take resource i with the maximal value in Q.
8: $L_i \leftarrow L_i + 1$.
9: Insert to Q the value $\delta_i(L_i + 1)$.
10: **until** $\sum_{i=1}^{m} L_i = s$
11: **return** the quantity vector, $\hat{L} = (L_1, L_2, \ldots L_m)$

In Lemma 5.7, we proved that given a full quantity vector, we can construct a symmetrically-full balanced placement (See also Remark 5.2). Therefore, constructing a symmetrically-full balanced placement from the MuRMaP quantity vector is presented in the following algorithm:

Algorithm 3 The max percentile inter region (MuRMaP) algorithm

1: Run Algorithm 2 for finding the MuRMaP quantity vector $\hat{L}$.
2: Run the balanced spread algorithm (Algorithm 1) on the quantity vector to yield the MuRMaP placement.

We will prove that the placement constructed by the MuRMaP algorithm, which is called the *MuRMaP placement*, is the solution for the placement problem (SPP-2):

Theorem 5.12. *The MuRMaP placement obtains the highest expected profit among all the symmetrically-full placements.*

Proof of Theorem 5.12. By Theorem 5.2, it is sufficient to prove that the MuRMaP placement has a profit at least as the profit of all symmetrically-full balanced placements.

By Corollary 5.11, the profit of every symmetrically-full balanced placement (which is equal to the balanced profit of its quantity vector) is a sum of s marginal contributions $\delta_i(j)$. Therefore, it is sufficient to prove that the profit of the MuRMaP placement is the sum of the s **largest** elements in the collection of vectors $\delta_i(), i = 1, \ldots, m$.

This result follows from the non-increasing monotonicity property of each of the $\delta_i()$ vectors. Based on this property, the queue in algorithm 2 contains the largest non-selected element of vector $\delta_i()$ for each resource type $i = 1, \ldots, m$. Thus, in the j^{th} stage it selects the j^{th} largest element in the vector collection, and Algorithm 2 selects the s **largest** elements.

□

Corollary 5.13. *Running the MuRMaP algorithm (algorithm 4) yields an optimal placement profit.*

5.5 Efficient Implementation of Algorithms

The MuRMaP algorithm can be implemented efficiently as stated next:

Claim 5.14 (Efficient Implementation of MuRMaP). *The MuRMaP algorithm (as well as the max-percentile algorithm for the single-region system) can be implemented efficiently and has time complexity of* $O((s+m)\cdot \log m)$.

Remark 5.4. We assume that computing the c.d.f values $\Pr(D \leq x)$ for every random variable D and integer x takes $O(1)$ time. Otherwise, if it takes $O(e)$ time to compute the c.d.f values, then the complexity of the algorithm is $O((s+m)\cdot(e+\log m))$.

Proof of Claim 5.14. In the MuRMaP quantity vector algorithm (Algorithm 2) the implementation of the maximum priority queue is done by a heap. The loop at Step 3 and the loop at Step 6 will each be carried out m and s times at most, respectively. The heap size equals the number of resource types $O(m)$ and inserting a value to the heap will take $O(\log m)$ operations. Therefore, the overall time complexity of the MuRMaP quantity vector algorithm is $O((s+m)\log m)$. In the case of a single-region system, the MuRMaP quantity vector algorithm is the max-percentile algorithm, as described in the previous chapter.

The MuRMaP algorithm (Algorithm 3) will calculate the MuRMaP quantity vector $\hat{L} = (L_1, L_2, \ldots, L_s)$ (Step 1 of Algorithm 3). Then it will transform the vector into a quantity vector of resource types containing at least one resource (i.e., $\hat{L} = (L_{i_1}, L_{i_2}, \ldots, L_{i_t})$ where $L_{i_j} \geq 1$). The transformation is done in $O(s)$. Lastly, it will perform the Balanced Spread Algorithm: Since in every step of that algorithm we spread at least one resource, the algorithm takes $O(s)$ time. Thus, in total, the time complexity of the MuRMaP algorithm is $O((s+m)\log m)$.

□

5.6 Resource Serves Multiple Request Types

The analysis given so far dealt with the case where a resource can serve only requests of a single request type. That means, if a single resource can serve requests of type i, then it cannot serve a type j request, for $i \neq j$. This is not the case in many applications, such as P2P VoD system (See Section 5.1.1) where the peer-servers (i.e., the resources) can store multiple movies, and thus can grant requests for the movies they contain (i.e., multiple request types).

This leads to examine systems where the resources can grant multiple request types. Suppose that the operator can add resources that can serve c request types (i.e., in the P2P VoD application it is equivalent to adding peer-servers containing c different movies). We assume that each resource can serve only one of these c request types, and the operator can choose which request types the resource can grant (i.e., in the P2P VoD application it is equivalent to what movies each peer-server contains).

To this end, we propose to use a heuristic algorithm. For the sake of explanation, we use the P2P VoD application to describe the algorithm. The algorithm operates as follows: First, the number of movie replicas in a region is determined by the Max Percentile algorithm operating on $s \cdot c$ peer-servers, where c is the number of copies a peer-server can hold. Then, the $s \cdot c/k$ movie replicas destined for a region are listed by their 'marginal contribution' to the profit, namely, in the order they were selected by the MuRMaP algorithm. Placement of these $s \cdot c/k$ movie replicas in a region is roughly done in an elevator type approach: Initially, it places the first s/k largest replicas in a descending marginal contribution order, starting with peer-server 1 of the region and ending with server s/k of the region. Then, it places the next largest s/k replicas, starting with server s/k and ending at peer-server 1. Then, it continues placing from peer-server 1 to peer-server s/k, and so on. This will lead to a relatively balanced spread of marginal contributions over the s/k servers.

5.7 Performance Evaluation

We use numerical analysis and simulations to evaluate a P2P VoD system performance (explained in Section 5.1.1) and compare it to other VoD works. We study the video servicing profit (revenue) and the fraction of requests that are granted as a function of the system parameters. The diversity of movies is a crucial parameter for the system performance; the larger the number of movies and the larger the weight of esoteric movies in this population, the harder it is on the system to satisfy the demand. Motivated by previous analytical works [11, 68] and empirical studies on the usage patterns in

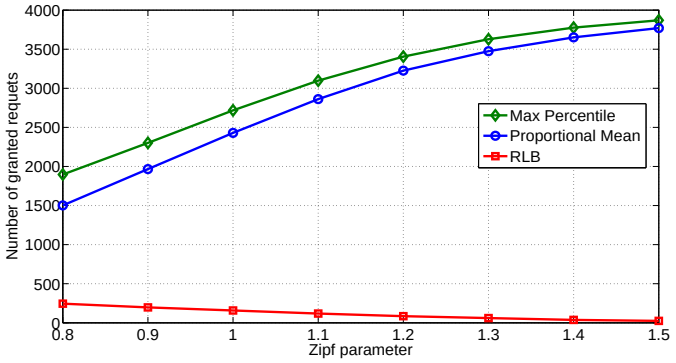


FIGURE 5.1: Performance of a single-region system with n=4,000, s=5,000, m=60,000.

VoD [6, 69], we assume that the movie demand follows a Zipf distribution. Since it is unclear how future demand will behave (the prior references mentioned above used a Zipf parameter of values between 0.56 and 1.5, depending on the study), we consider a wide range of Zipf distributions and vary the Zipf parameter from 0.5 to 1.5. We assume that the aggregate demand comprises n requests, where each request picks a movie i with probability p_i, $\sum_i p_i = 1$, and p_i follows a Zipf distribution.

Single-region. Figure 5.1 shows the number of requests granted by the P2P network as a function of the Zipf parameter for a single-region P2P setting, given a catalog of $m = 60,00$ movies, $s = 5000$ servers, and aggregate demand of $n = 4000$ requests. The number of granted requests is equal to $\sum_{i=1}^{m} E_{D_i}[\min(L_i, D_i)]$. The figure demonstrates that at low values of the Zipf parameter the system can grant about 50% of the requests; this is due to the fact that there are many esoteric movies and the system simply cannot hold all of them. In this case, the P2P system will require a significant support from the central server. When the Zipf parameter is large, the system can handle almost all the requests since there are not many esoteric movies. For the sake of comparison, we also plot the performance of the proportional mean placement [11] and of the RLB placement [50]. As shown, both are not as efficient as the Max Percentile placement. We repeat the analysis for a larger catalog of $m = 100,000$ movies and observe similar results (not shown).

Figure 5.2 demonstrates the impact of the system load on performance (the number of granted requests) for a large-scale single-region P2P setting consisting of a catalog of $m = 60,000$ movies and $s = 50,000$ servers. The number of requests is $30,000$ and $40,000$, corresponding to loads of 0.6 and 0.8, respectively. As expected, we see that the number of granted requests increases with the number of submitted ones. We also show the performance of proportional and RLB placements for the load of 0.8 and observe similar results in the smaller-scale network in Figure 5.1.

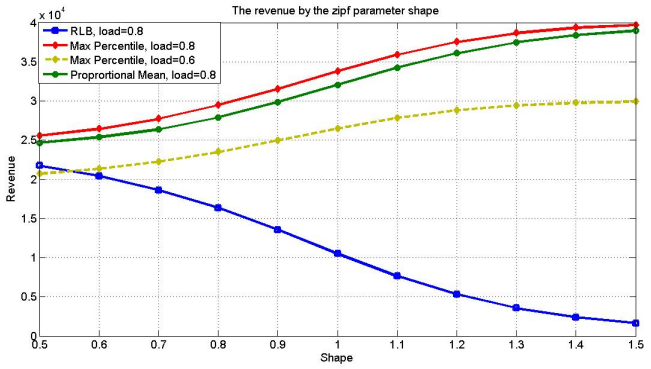


FIGURE 5.2: Performance of a large-scale single-region network with s=50,000, m=60,000, n=40,000 (load 0.8) and 30,000 (load 0.6).

Max percentile and proportional mean. As shown in Figure 5.1 and 5.2, the performance gap between max percentile and proportional mean is largest when the Zipf parameter is small. This happens because the settings include a large number of esoteric movies for which the variability of the demand is high, resulting in large inaccuracies of proportional mean (and RLB) placements. The gap will be large in other scenarios as well. For example, consider a two-movie system with n peer-servers. The first movie has a deterministic demand of n requests and the second one has a stochastic demand of nk^2 requests w.p $1/k$ and 0 requests w.p $1 - 1/k$[1]. Max percentile will place n replicas for the first movie, yielding a maximum revenue (profit) of n (number of peers). However, proportional mean will place $\frac{n}{k+1}$ and $\frac{nk}{k+1}$ replicas for the first and second movies, respectively. Thus, the revenue of proportional mean is equal to $\frac{2n}{k+1}$. Note that as k approaches infinity, the performance ratio approaches zero.

Remark 5.5. One might consider optimizing of proportional mean by truncating the demand distributions, i.e., the demand cannot be larger than the system capacity. Even under that version, proportional mean might provide an allocation which is away from the optimal. Consider a single-region system that has $\frac{n}{2}$ movies, denoted by $t_1, t_2, .., t_{\frac{n}{2}}$, with deterministic demand of a single request. In addition, the system contains $\frac{n}{2}$ movies (denoted by $t_{\frac{n}{2}+1}, , .., t_n$), in each of which the demand has probability of $\frac{1}{\sqrt{n}}$ to have n requests, and probability of $1 - \frac{1}{\sqrt{n}}$ to have zero requests. Suppose the system contains n movies. The optimal placement allocates one peer-server for movies $t_1, t_2, .., t_{\frac{n}{2}}$, and allocates $\frac{n}{2}$ replicas from one of the other movies. The optimal profit, equals to $\frac{n}{2} + \frac{\sqrt{n}}{2} = \theta(n)$. Proportional mean, however, allocates $\frac{1}{\frac{n}{2}+\frac{n}{2}\cdot\sqrt{n}} \cdot n = \frac{2}{1+\sqrt{n}}$ peer-servers for movies $t_1, t_2, .., t_{\frac{n}{2}}$, and $\frac{\sqrt{n}}{\frac{n}{2}+\frac{n}{2}\cdot\sqrt{n}} \cdot n = \frac{2\sqrt{n}}{1+\sqrt{n}}$ for movies $t_{\frac{n}{2}+1}, , .., t_n$. Its profit is equal to $\frac{2}{1+\sqrt{n}} \cdot \frac{n}{2} + \frac{2\sqrt{n}}{1+\sqrt{n}} \cdot \frac{n}{2} \cdot \frac{1}{\sqrt{n}} = \frac{2n}{\sqrt{n}+1} = \theta(\sqrt{n})$. Thus, proportional mean is far by a factor of $\theta(\sqrt{n})$ from the optimal value.

[1]This can represent the case where the demand depends on the review a movie is about to receive.

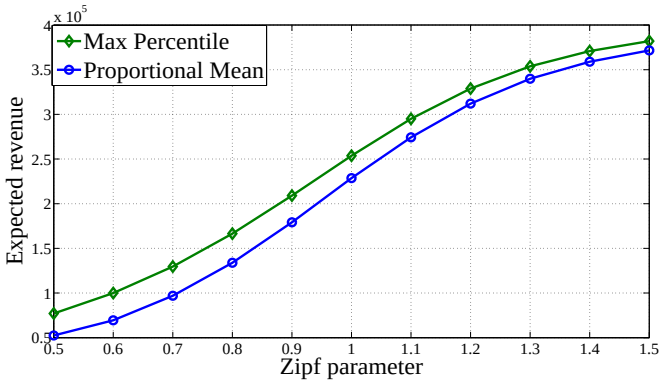


FIGURE 5.3: Performance of a 20-region network with n=40,000, s=50,000, m=60,000, R_{glo}=1, R_{loc}=9.

Multi-region. Next, we study the performance of a multi-region system. We consider a setting with $k = 20$ regions, local revenue of $R_{loc} = 9$, global revenue of $R_{glo} = 1$, and remaining parameters as before, $m = 60000$, $s = 50,000$ and $n = 40000$. Figure 5.3 shows the revenue computed using Eq. (5.1) as a function of the Zipf parameter. The revenue of Max-Percentile is compared to that of proportional mean when it is applied locally to each region. We observe similar behavior in the single-region case. In a multi-region setting, the number of locally served requests increases with the Zipf parameter. For large parameter values, the majority of the requests are served locally, and the gap between max-percentile and proportional is small. The gap is larger for small values of the parameter since many requests are served from remote regions.

Resource serves multiple request types . Next, we examine systems where resources can serve multiple request types. To this end, we use the heuristic algorithm in Section 5.6. That is, the number of replicas in a region is determined by the max percentile algorithm, operating on a capacity of $s \cdot c$ resources, where c is the number of request types a resource can serve; in our case $c = 5$. Then, replicas are sorted in a descending order of popularity and placed on the individual peers (resources) of the region, using an elevator-type placement. We use a simulation over the same setting as used previously to evaluate the performance. After computing the heuristic placement once, we generate the demand multiple times and using an optimal matching algorithm we derive the expected profit (revenue). We compute the proportional mean placement in a similar fashion, and show the performance of both in Figure 5.4. When the Zipf parameter is large, the demand variability is low, and Proportional Mean becomes a good predictor for the distribution's tail. Hence, both algorithms produce similar results. The performance gap is larger when the Zipf parameter is around 1. When the Zipf parameter is small, both algorithms face the same heuristic decision whether or not to place a copy of an esoteric movie, resulting in similar performance.

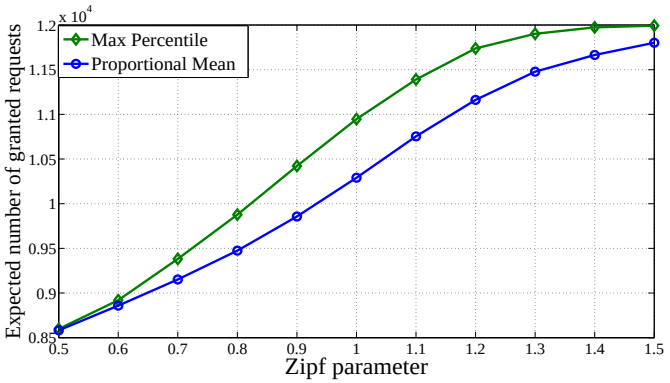


FIGURE 5.4: Performance of single-region setting where each resource can serve 5 request types and n=12,000, s=15,000, m=75,000.

5.8 Further improvement: An implementation which is sublinear in the number of resources

Remark 5.6. I would like to thank an anonymous referee who proposed the basic ideas for the algorithm described in this section.

The MurMap algorithm presented in this chapter runs at $O((s+m)\cdot\log m)$ time, where s is the total storage and m is the number of resource types. While it is assumed that the operator must place the resources in the different regions (See Remark 1.2), thus leading to an algorithm that must be linear in the number of resources (i.e., in the total storage s), it is possible to implement the algorithm differently, if the assumptions change. That means, if the operator does not place the resources, and the CDF values $\Pr(D_i \geq j)$ are given as an oracle (do not need $\Omega(s)$ for pre-processing computation). In such a case, we show that the MurMap algorithm can be implemented in sublinear time in the number of resources, i.e., polynomial in the thin input.

First, we show how to solve the single region system ($k = 1$) problem: Let L_i be the number of resources for type i. Using the transformation done in Section 4.2, the optimal placement $L = \{L_1, L_2, \ldots, L_m\}$ maximizes

$$\sum_{i=1}^{m}\sum_{j=1}^{L_i}\Pr(D_i \geq j), \tag{5.11}$$

under the constraint $L_1 + L_2 + \ldots + L_m = s$.

Consider the arrays $a_i = [\Pr(D_i \geq 1), \Pr(D_i \geq 2), \ldots \Pr(D_i \geq s)]$ for $i = 1, 2, \ldots, m$. The goal is to find an element $a_i[j] = \Pr(D_i \geq j)$ (among the $k \cdot s$ elements) such that the total number of elements in these arrays with value larger than or equal to $a_i[j]$ is

exactly s, i.e., the s^{th} largest element. Thus, the single region problem can be reduced to the following problem:

THE t^{th} LARGEST ELEMENT IN m SORTED ARRAYS

Input: A vector $(n_1, n_2, \ldots, n_m)$ of positive integers. Monotonically decreasing arrays $a_i = [a_i[1], a_i[2], \ldots, a_i[n_i]]$ for $i = 1, 2, \ldots, m$ (i.e., a_i has n_i elements). The total number of elements $N = \sum_{i=1}^{m} n_i$. A positive integer t.

Problem: Find the t^{th} largest element, i.e., an element $a_i[j]$ such that the total number of elements in these arrays with value larger than or equal to $a_i[j]$ is exactly t.

We will show a recursive algorithm that runs at logarithmic time in the number of the array elements (i.e., $\log n_i$). In every iteration, we compute the maximal and the minimal medians of every array. Suppose $a_{i_{min}} = \arg\min_i a_i[\frac{n_i}{2}]$ and $a_{i_{max}} = \arg\max_i a_i[\frac{n_i}{2}]$ are the arrays with the maximal and the minimal medians, respectively. We observe that the maximal and minimal medians are respectively greater than and smaller than $\frac{N}{2}$ elements in these arrays.

Suppose that $\frac{N}{2} \geq t$. Then, the maximal median is greater than the t^{th} largest element. Thus, at least half of the elements in the maximal median array $a_{i_{max}}$ are larger than the t^{th} largest element and can be "thrown away"; that means, the array $a_{i_{max}}$ is cut to half, and our recursive algorithm will continue to find the t^{th} element among the remaining elements.

Alternatively, and in a similar way, if $\frac{N}{2} < t$, then at least half of the elements in the minimal median array $a_{i_{min}}$ are smaller than the t^{th} largest element and can be "thrown away". Since the t^{th} largest element is larger than $\frac{n_{i_{min}}}{2}$) elements in $a_{i_{min}}$, our recursive algorithm will continue to find the $(t - \frac{n_{i_{min}}}{2})^{th}$ largest element among the remaining elements.

The algorithm runs for $O(\sum_{i=1}^{m} \log n_i)$ iterations. Finding the minimal and maximal medians in every iteration takes $O(\log m)$ by using a min and a max heap, respectively. Thus, the complexity of this algorithm over the single region problem, where $t = n_i = s$, is $O(m \log m \log s)$.

By using such implementation, we can show that MurMap can be implemented in a $O(mk + m \log m \log s)$ time algorithm: The MuRMaP quantity vector algorithm (Algorithm 3) is implemented as done in this section at $O(m \log m \log s)$. For every resource type $1 \leq i \leq m$, the Balanced Spread Algorithm will derive in $O(k)$ time the number of resources of resource type i for every region $1 \leq j \leq k$. Thus, the Balanced Spread

Algorithm takes $O(km)$ time. The MurMap, which runs the MuRMaP quantity vector algorithm and the Balanced Spread Algorithm, takes $O(mk + m \log m \log s)$.

Remark 5.7 (applicability). In practice it is expected that in most cases the operator places the resources and/or has to derive the full distribution data, in which case only treating the data takes $\Omega(s)$ and thus a sub-linear algorithm is of little value. The value of the sublinear algorithm is in cases where the demand distribution is either given by a formula or by an Oracle, and the operator computes only the placement quantities.

Chapter 6

The Heterogeneous-Region System Static Placement Problem (SPP-3)

In this chapter, we extend the problem from homogeneous-region system (presented in Chapter 5) to a heterogeneous-region system. The problem posed by this system is that of optimizing the simple profit function under capacity constraints (and with no symmetry between the regions). The solution, which is called the *Bipartite Graph (BG)* algorithm, is using graph theory techniques, and it differs from the max-percentile solutions presented in Chapters 4, 5. The work in this chapter is based on a published article [13].

6.1 Problem Formulation

Let $D = \{D_i^j\}$ be the demand for type i resources in region j. In this chapter, we solve the placement problem over a capacity-constrained system. Thus, the system can hold up to s^j resource in region j. In addition, the system comprises heterogeneous regions, meaning that neither the storage values $\{s^j\}_{j=1}^k$ nor the regional demands $\{D_i^j\}_{j=1}^k$ are necessarily identical across regions or statistically identical (see Chapter 5), respectively.

A placement $L\{L_i^j\}_{i=1}^m$ is called a **feasible** placement if it represents a valid resource placement of the system, i.e., $L^j = \sum_{i=1}^m L_i^j \leq s^j$ in every region j. To this end, we define our placement problem to find a feasible placement L that maximizes the simple

profit $P(L, D)$, which is equal to

$$P(L,D) = \sum_{i=1}^{m}\sum_{j=1}^{k} R_{loc} \cdot E_{D_i^j}[\min(L_i^j, D_i^j)] + \sum_{i=1}^{m} R_{glo} \cdot E_{D_i}[\min(L_i, D_i)], \tag{6.1}$$

where $R_{loc}, R_{glo} \geq 0$ are non-negative revenue constants. The placement problem in this chapter is formulated as follows:

THE STATIC PLACEMENT PROBLEM FOR A HETEROGENEOUS-REGION SYSTEM (SPP-3)	
Assumptions:	Capacity-constrained system optimizing simple profit.
Input:	The demand $D = \{D_i^j\}$, the storage values $\{s^j\}$.
Problem:	Find the placement $L_{opt} = \{L_i^j\}$ that maximizes the simple profit in Eq. (6.1) under the constraint $L^j = \sum_{i=1}^{m} L_i^j \leq s^j$.

We denote by $|L| = \sum_{j=1}^{k}\sum_{i=1}^{m} L_i^j$ the number of resources in L.

6.2 A Solution for the Placement Problem (SPP-3)

The placement problem as defined above seems to be highly complex and therefore challenging. The reason is that one must deal with a very large input data (m distributions, where each could be represented by $O(s)$ data elements; in some applications, s and m can reach values up to 10^4).

Fortunately, we are able to utilize the powerful properties of arbitrary distributions as we derived earlier (e.g., Chapter 3, Eq. (3.9)), and thus transform the problem into the one presented next in Theorem 6.1. Using this transformation will later be utilized to devise a reduction to the well-known **min-cost flow** problem, and derive efficient algorithms for the placement problem.

6.2.1 Transformation to a cost function

To solve the placement problem, we use Eq. (6.1) and convert a profit maximization problem into a cost minimization problem. We define the *cost function* of placement L in the following formula:

$$E(C^L) = \underbrace{R_{loc}\sum_{j=1}^{k}\sum_{i=1}^{m}\sum_{n=1}^{L_i^j}(1 - \Pr(D_i^j \geq n))}_{\text{Local}} + \underbrace{R_{glo}\sum_{i=1}^{m}\sum_{n=1}^{L_i}(1 - \Pr(D_i \geq n))}_{\text{Global}} \tag{6.2}$$

We can use this cost function to solve the placement problem, as follows:

Theorem 6.1. *Let L be a feasible placement that minimizes $E(C^L)$ among feasible placements with s resources (i.e., $|L| = s$). Then L is an optimal placement (i.e., solves SPP-3).*

We prove the theorem by the fact that every discrete non-negative random variable X and every constant C satisfy $E(\min(X, C)) = \sum_{k=1}^{C} \Pr(X \geq k)$ (i.e., Eq. (3.9)). A full proof of the theorem can be found in Appendix 6.

6.2.2 Preliminaries: The min-cost flow problem

To find an optimal solution to the placement problem, we present the *min-cost flow problem*, which is a generalization of the notable max flow problem. In the problem, one considers a directed graph $G = (V, E)$ where every edge $e \in E$ has a non-negative integer *capacity* $c(e)$ and a real-value *weight* $w(e)$ (also called *cost*). The graph must contain two different nodes: a source node x and a sink node y. An *x-y-flow* $f : E \to R^+$ is defined on the graph edges $(v, v') \in E$ in the same way as defined in the max-flow problem. That means, the flow must satisfy the following properties: 1) *Capacity constraint:* for each edge e, we have $0 \leq f(e) \leq c(e)$. 2) *Conservation of flows:* for every vertex $v \in V \setminus \{x, y\}$ we have $\sum_{(v',v)\in E} f(v', v) = \sum_{(v,v')\in E} f(v, v')$. In addition to the standard definitions, we define the *flow in node* $v \neq x, y$ as the income flow (and by conservation of flows, the outcome flow) to (from) node v. We denote it by $f^{in}(v)$, which equals $f^{in}(v) = \sum_{(v,v')\in E} f(v, v')(= \sum_{(v',v)\in E} f(v', v) = f^{out}(v))$. The *flow value* of f, as defined in the max-flow problem, is $|f| = \sum_{(x,v)\in E} f(x, v) = \sum_{(v,y)\in E} f(v, y)$. The *weight (or cost) of flow* f is $w(f) = \sum_{e\in E} f(e)w(e)$.

Given a parameter n, the classic *minimum-cost flow* problem is to find a flow f_{opt} of value n that has minimum weight among all flows of value n. This means that for every flow f' where $|f'| = |f_{opt}| = n$ we have $w(f_{opt}) \leq w(f')$. It is well-known that if the capacities $c(e)$ of a min-cost flow network are integers, then there exists a min-cost flow f where the flow in every edge is integer; a proof can be found in [70], Theorem 9.10.

In Section 6.3.2, we present the Successive Shortest Path algorithm for solving a min-cost flow problem on a general graph $G = (V, E)$ with non-negative weights. This algorithm was studied in [71]. Its time complexity is $O(|f||E||V|)$ where $|f|$ is the required flow value.

In the next subsection, we will reduce the placement problem into a min-cost flow problem. To this end, we define a directed graph $G = (V_1 \bigcup V_2 \bigcup \ldots V_k, E)$ to be a *k-layer*

graph if the following holds: 1) The vertex sets $V_1, V_2, \ldots V_k$ are pairwise disjointed. 2) The vertices in V_i, which are called *layer-i* vertices, can be connected only to vertices in successive layers (i.e., if $(u, v) \in E$ then there is $1 \leq i \leq k$ such that $u \in V_i$ and $v \in V_{i+1}$). Note that a two-layer graph definition is the equivalent definition of a bipartite graph.

6.2.3 Transformation to a min-cost flow problem

We use Theorem 6.1 to reduce the placement problem into a min-cost flow problem in a **seven-layer graph** G^7. That means, we show the following properties: 1) For every placement L, there is a **corresponding flow** f_L, where the cost function of L is equal to the weight of f_L, i.e, $E(C^L) = w(f_L)$. 2) Finding the min-cost flow f_{opt} in G^7 derives the optimal placement L_{opt} that solves the placement problem (SPP-3).

The reduced graph $G^7 = (V_1' \bigcup V_2' \ldots \bigcup V_7', E')$ (see Figure 6.1) contains the source node x in the first layer ($V_1 = \{x\}$) and contains the sink y in layer-7 ($V_7 = \{y\}$). We set the required flow value to be the total storage value over all regions (i.e., $|f| = s = \sum_{i=1}^{k} s^k$, where s^j is the bound on the number of resources that can be placed in region j). Layer-2 contains *region nodes* denoted as $a^1, a^2, \ldots, a^k$. The flow f_L in node a^j (i.e., $f_L^{in}(a^j)$) represents the total resources placed in region j (i.e., $f_L^{in}(a^j) = L^j$). The edges between the source x and region node a^j have capacity s^j, reflecting a bound on the number of resources to be placed in region j. The cost of these edges is 0. Layer-3 contains pairs of *(region, type) nodes* such that the flow in node (a^j, t^i) represents the storage in region j of resource type i (i.e., $f_L^{in}(a^j, t^i) = L_i^j$). We connect between layer-2 region nodes (a^j) and layer-3 (region, type) nodes (a^j, t^i) for all resource types $1 \leq i \leq m$ and all regions $1 \leq j \leq 1$. The capacity of these edges is infinity (the flow through the nodes is unlimited), and the cost is 0.

To represent the local part of the cost function $E(C^L)$, the layer-4 nodes are the triples (a^j, t^i, r) called *(region, type, # resources) nodes* for regions $1 \leq j \leq k$, resource types $1 \leq i \leq t$ and an integer $1 \leq r \leq s$. Positive flow in node (a^j, t^i, r) means that the number of type i resources from region j is larger than or equal to r (in other words, if $L_i^j \geq r$, then $f_L^{in}(a^j, t^i, r) = 1$, otherwise $f_L^{in}(a^j, t^i, r) = 0$). To allow the flow values of 0 or 1, the capacity of the edge connecting (a^j, t^i) and (a^j, t^i, r) is equal to 1. Note that $L_i^j \geq r$ iff the number $R_{loc}(1 - \Pr(D_i^j \geq r))$ appears in the summation of Eq. (6.2). Thus, we set the cost of the edge from (a^j, t^i) to (a^j, t^i, r) to be $R_{loc}(1 - \Pr(D_i^j \geq r))$.

Layer-5 nodes are called *resource type nodes* or simply *type nodes*. The value $f_L^{in}(t^i)$ represents the number of type i resources from all regions (i.e., $f_L^{in}(i) = L_i$). For all $1 \leq j \leq k, 1 \leq r \leq s$, we connect layer-4 nodes (a^j, t^i, r) to resource type node t^i.

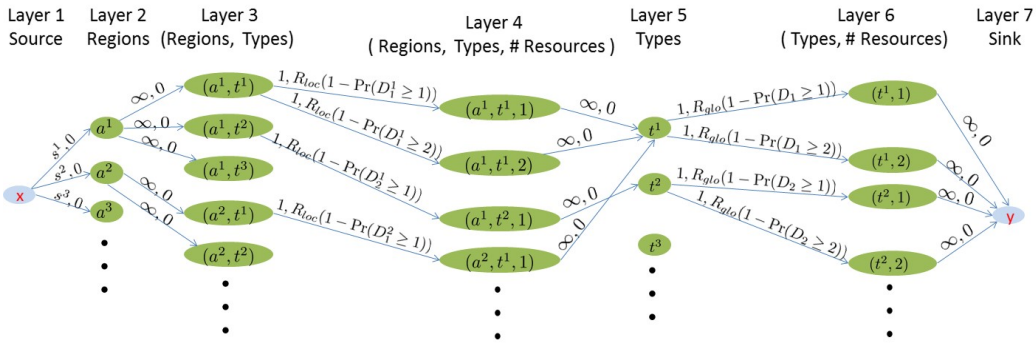


FIGURE 6.1: The seven-layer graph

The capacity of these edges is infinity[1], and the cost of these edges is 0. Layer-6 nodes represent pairs (t^i, r) of *(type, # resources) nodes*. Positive flow in node (t^i, r) indicates whether the number of type i resources is larger than (or is equal to) r.

Similar to the edges between layer-3 and layer-4, the edges between t^i and (t^i, r) represent the global part of the cost function $E(C^L)$, and they have capacity 1 and cost $R_{glo}(1 - \Pr(D_i \geq r))$. Finally, we connect all (resource type, # resources) nodes to the sink node y in layer-7. The cost of these edges is 0 and the capacity is infinity.

In the following lemma, we show how to transform a placement L into a **corresponding flow** f_L in the seven-layer graph G^7:

Lemma 6.2. *Let L be a feasible placement with n resources. Then there exists an integer flow f_L (i.e., $f_L(e)$ is integer for all edges $e \in E$) called the **corresponding flow of** L with flow value $|f_L| = n$ that obeys the following conditions: (1) ${f_L}^{in}(a^j, t^i) = L_i^j$, ${f_L}^{in}(t^i) = L_i$ for all area j and resource type i. (2) The flow f_L sends 1 unit of flow on the edges between layer-3 $((a^j, t_i))$ and layer-4 $((a^j, t_i, r))$ of costs $R_{loc}(1 - \Pr(D_i^j \geq r))$ for $1 \leq r \leq L_i^j$. (3) The flow f_L sends 1 unit of flow on the edges between layer-5 (t_i) and layer-6 $((t_i, r))$ of costs $R_{glo}(1 - \Pr(D_i \geq r))$ for $1 \leq r \leq L_i$. Moreover, the cost of f_L is equal to the alternative cost of L, i.e., $w(f_L) = E(C^L)$.*

We provide a formal proof of Lemma 6.2 in Appendix 6. The technique used to create the flow can be well understood from the following example.

Example 6.1. Consider a system with two regions denoted by a^1, a^2, and two resource types denoted by t^1, t^2. Suppose that there are two resources in area a^1 of different resource types, i.e., $L_1^1 = L_2^1 = 1$. In area a^2 there is a single resource of type t_1, i.e., $L_1^2 = 1, L_2^2 = 0$.

We depict in Figure 6.2 the flow f_L, which is corresponding to the placement L. The flow on solid-line blue edges, dotted-line red edges, and the black edge (x, a^1) are 0, 1,

[1]It can be equivalently set to one, as the flow in (a^j, t^i, r) is either 0 or 1.

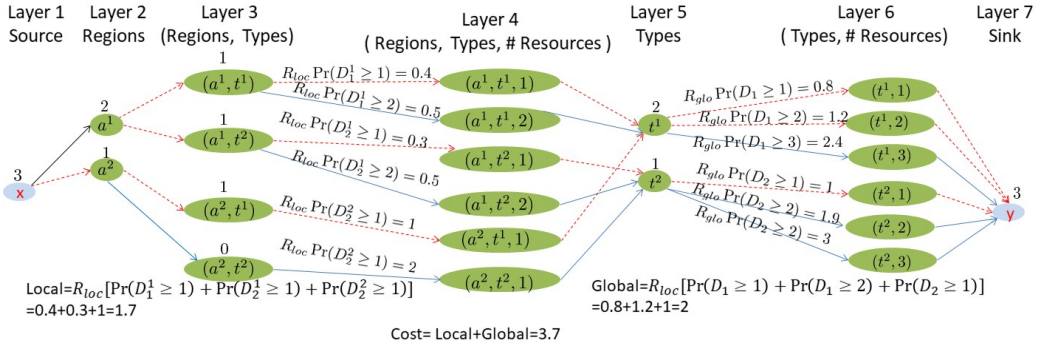


FIGURE 6.2: The flow f_L corresponding to the placement of Example 6.1, i.e., $L_1^1 = L_2^1 = 1$ and $L_1^2 = 0, L_1^2 = 1$. The flow on solid-line blue edges, dotted-line red edges, and the black edge (x, a^1) are 0, 1, and 2, respectively. The flow weight $w(f_L)$, which equals the cost function of the placement $E(C^L)$, is the sum of red edge costs.

and 2, respectively. The flow in node values $f_L^{in}(v)$ is shown above the vertices v of layers $1, 2, 3, 5, 7$. The cost of edges with non-zero cost is depicted above the edges. The storage values of regions a, b are respectively $s^a = 2$ and $s^b = 1$.

The flow weight $w(f_L)$, which equals the cost function of the placement $E(C^L)$, is the sum of red edge costs. The local part of the cost function $E(C^L)$ is equal to the sum of red edge costs between vertices of layer 3 and layer 4, i.e., is equal to $0.4+0.3+1 = 1.7$. Similarly, the global part of the cost is the sum of red edge costs between vertices of layer 5 and layer 6, i.e., is equal to $0.8 + 1.2 + 1 = 2$. The cost of the flow (and the alternative cost function) is equal to the sum of the local and global parts, i.e., $E(C^L) = 1.7 + 2 = 3.7$.

Remark 6.1. Note that if G^7 contains edges with identical weights, then there can be multiple corresponding flows for a given placement. For example, consider a system with one region ($k = 1$) and one resource type ($m = 1$). Figure 6.3 depicts two flows, denoted by f and f', corresponding to the placement that contains a single resource i.e., $L_1^1 = 1$. The edges incoming to vertices $(a^1, t^1, 1)$ and $(a^1, t^1, 2)$ have weights of $R_{loc}(1-\Pr(D_1^1 \geq 1)) = R_{loc}(1 - \Pr(D_1^1 \geq 2)) = 0.5$ (this can be possible if $(\Pr(D_1^1 \geq 1) = \Pr(D_1^1 \geq 2))$. Both flows have cost of $R_{loc}(1 - \Pr(D_1^1 \geq 1)) + R_{loc}(1 - \Pr(D_1 \geq 1)) = 0.5 + 0.6 = 1.1$.

Note that the edge costs $g(r) = R \cdot (1 - \Pr(D \geq r))$ increases as the number of resources r increases. This leads to the correctness of the reduction, as stated next:

Lemma 6.3. *Let f be an integer flow in G^7. Let L be a placement such that $L_i^j = f^{in}(a^j, t^i) = f_i^j$. Then the cost of f is not smaller than the cost of a corresponding flow of L, i.e., $w(f) \geq w(f_L)$. Moreover, $w(f) = w(f_L)$ iff f is a flow corresponding to L.*

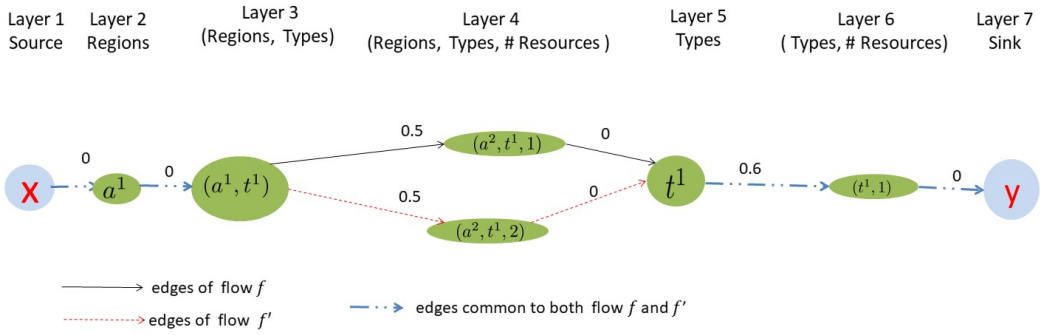


FIGURE 6.3: The flows f, f' corresponding to the placement $L_1^1 = 1$.

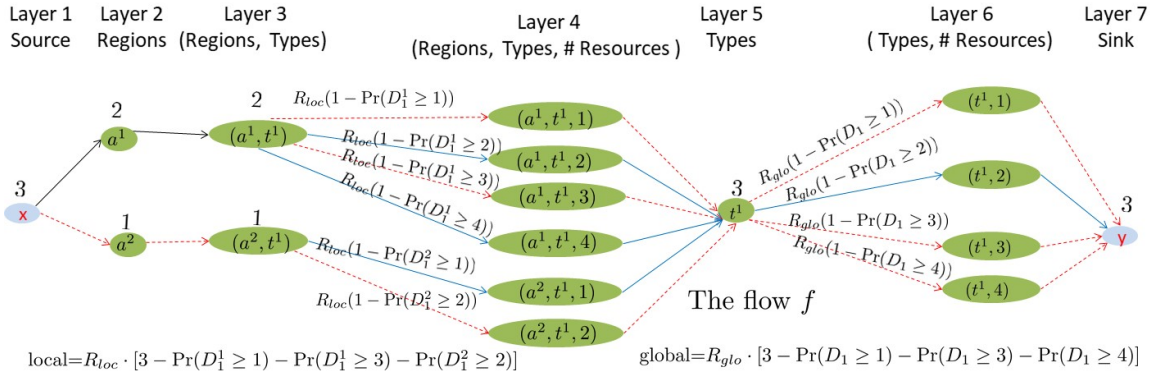


FIGURE 6.4: A given flow f. The dotted-line red edges have 1 unit of flow, the solid-line blue edges are with 0 units of flow, and solid-line black edges have 2 units of flow.

The important point of Lemma 6.3 is that the optimal flow passes through the first $f_i^j = f^{in}(a^j, t^i)$ edges emanating (a^j, t^i) node (that is the first f_i^j edges with minimum cost), thus reflecting a consistent cost of placing f_i^j resources of type i in region j.

A rigorous proof of Lemma 6.3 is provided in Appendix 6. To demonstrate the correctness of the lemma, suppose the system contains two regions $k = 2$ and one resource type $m = 1$. Let f be a flow presented in Figure 6.4. The unique flow f_L corresponding to $L_i^j = f_i^j$, which passes through the upper f_i^j edges, is depicted in Figure 6.5. One can verify that the cost of f_L is equal to $E(C^L)$, and that the cost of f_L is not larger than the cost f, as the edge costs $g(r) = R \cdot (1 - \Pr(D \geq r))$ are monotonically increasing.

Thus, by Theorem 6.1 and Lemmas 6.3, 6.2 we can draw the following conclusions:

Corollary 6.4. *Let f_{opt} be an integer min-cost flow of G^7. Let L be the placement created from the values of the flow in (region, type) nodes $L_i^j = f_{opt}^{in}(a^j, t^i)$. Then the flow f_{opt} is corresponding to L.*

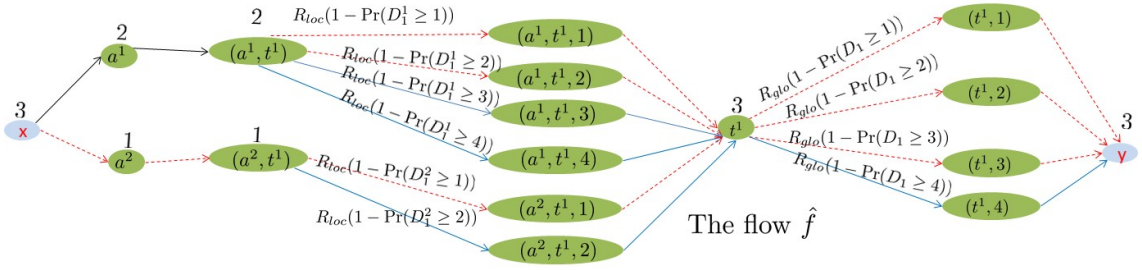


FIGURE 6.5: The flow f_L corresponding to $L_i^j = f_i^j$.

Corollary 6.5. *Let f_{opt} be an integer min-cost flow of G^7 with flow value $|f_{opt}| = s$. Let L be the placement of the flow in (region, type) nodes $L_i^j = f_{opt}^{in}(a^j, t^i)$. Then L solves the placement problem.*

Note that the number of (region, type, # resources) nodes is $O(smk)$. Thus, the number of vertices and the number of edges in G^7 is $O(smk)$. Since G^7 contains edges with non-negative weight, we can use the well-known Successive Shortest Path (SSP) algorithm. Running SSP on $G^7 = (V', E')$ takes $O(|f||V'||E'|)$, where $|f| = O(s)$ is the required flow value. Expressing the time complexity of SSP in terms of total storage, resource types and regions (denoted as s, k, m) is $O(s^3m^2k^2)$. This time complexity is quite high, and therefore in the next section we will further optimize this algorithm to yield a more efficient one, the Bipartite Graph (BG) algorithm.

6.3 Bipartite Graph (BG) Algorithm

To develop an efficient algorithm, we propose the *Bipartite Graph* (in short *BG*) algorithm, which reduces the time complexity of SSP over the seven-layer graph G^7. To do so – we first have to present the technicalities of SSP. In Section 6.3.1, we describe the residual graph that SSP uses. In Section 6.3.2, we use the well-known Successive Shortest Path (SSP) algorithm. In Section 6.3.3, we describe the *bipartite-like graph* G_f^B, which shrinks the seven-layer graph G^7 graph and removes unnecessary paths, so that running the BG algorithm over G_f^B is equivalent to running SSP on G^7. Finally, we present the implementation of the Bipartite Graph (BG) algorithm in Section 6.3.4.

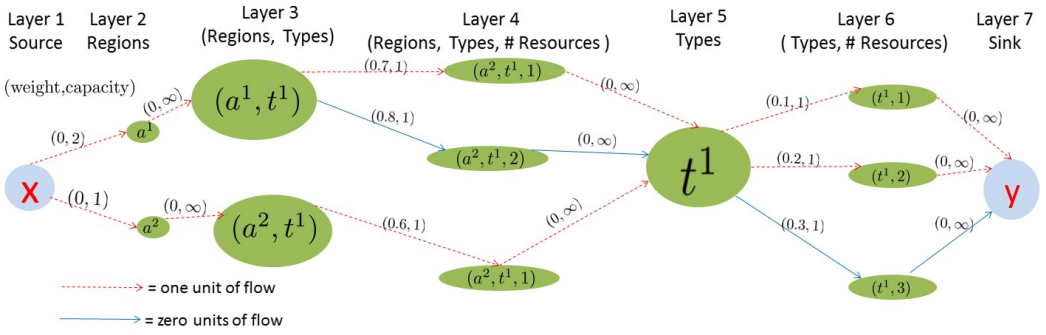


FIGURE 6.6: The flow f_L corresponding to the placement of $L_1^1 = L_2^1 = 1$.

6.3.1 Preliminaries: The residual graph

The SSP algorithm is a well-known algorithm that solves the min-cost flow problem for graphs where all edges have a non-negative cost. Running examples, as well as a full description of the algorithm, can be found in [70].

Given a flow f on a graph $G = (V, E)$, the Successive Shortest Path (SSP) uses the well-known **residual graph** $G_f = (V, E_f)$. Every edge in the residual graph edges is associated with weight w_f and capacity c_f (also called the *residual capacity*). The residual graph G_f is constructed from G and f by the following steps: 1) Add to G_f edges from G, such that every edge $(v, v') \in E$ will have a weight of $w_f(v, v') = w(v, v')$ and a capacity of $c_f(v, v') = c(v, v') - f(v, v')$. 2) Add the reverse edges of G. That means, if $(v, v') \in E$, then add edge (v', v) to G_f with a weight of $w_f(v', v) = -w(v, v')$ and a capacity of $c_f(v', v) = f(v, v')$. Note that for every edge e in G_f we have $c(e) \geq 0$. 3) Set the weight of every edge $e \in E_f$ with zero residual capacity ($c_f(e) = 0$) to $w_f(e) = \infty$.

Example 6.2. Suppose the system contains two regions ($k = 2$) and one resource type ($m = 1$). Let L be a placement that contains one resource in every region ($L_1^1 = L_2^1 = 1$). Suppose that the storage values of regions 1 and 2 are respectively $s^1 = 2$ and $s^2 = 1$. We denote by f_L the flow corresponding to L, which is depicted in Figure 6.6. The dotted-line red edges have 1 unit of flow, the solid-line blue edges are with 0 units of flow. We depict above every edge its capacity and weight.

The residual graph of f_L, denoted by $G_{f_L}^7$, is depicted in Figure 6.7. We indicate above every edge its weight. Dotted-line edges have zero residual capacity, and thus they have infinite weight.

It is vital to be familiar with shortest path algorithms, since they are used by the SSP algorithm. One notable algorithm to calculate the shortest paths from one source v to all other vertices, is the *Bellman-Ford* algorithm. The running time of Bellman-Ford

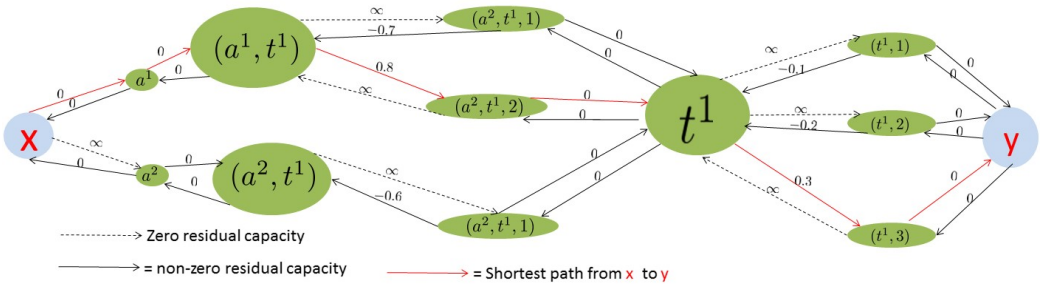


FIGURE 6.7: The residual graph of f_L. Dotted-line edges have zero residual capacity, and thus they have an infinite weight. The red edges are the shortest path between x and y, and thus SSP augments 1 unit of flow along the path.

on graph $G = (V, E)$ is $O(|E| \cdot |V|)$. More information on Bellman-Ford can be found in [72].

6.3.2 Preliminaries:The Successive Shortest Path (SSP) algorithm (without node potentials)

Finally, we present a description of the well-known SSP algorithm on a general graph G with required flow n. In the initial step, the algorithm assigns the zero flow $f := 0$ (i.e., $f(e) = 0$ for all $e \in E$) with flow value $|f| = 0$ and constructs the residual graph G_f. The algorithm works iteratively, and in the i^{th} iteration it calculates a minimum-cost flow of flow value larger than or equal to i. The algorithm runs the following steps in each iteration: 1) Check if the flow value $|f|$ is equal to the required flow n. If so, then f is the optimal flow and the algorithm terminates. 2) Calculate the shortest paths from source x on G_f with respect to the weight function w_f. SSP retrieves the shortest path p between x and y, which is called the *augmenting path*. If the shortest path's weight is equal to infinity (i.e., $w_f(p) = \infty$) – then the maximum flow value of the graph G is strictly less than n, and the algorithm returns an error. 3) SSP augments $\delta = \min(n - |f|, \min\{c(e)|e \in p\}) > 0$ units of flow through p. This means that if $(v, v') = e \in p$ is in the original graph (i.e., $e \in E$), then SPP updates $f(e) \leftarrow f(e) + \delta$, and if the reverse edge in G (i.e., $(v', v) \in E$), then SSP updates $f(v', v) \leftarrow f(v', v) - \delta$. 4) SSP creates a new residual graph G_f, according to the updated flow f.

In [71, 73] the correctness of SSP is proven as follows:

Theorem 6.6. *Suppose that f is a min-cost flow in a graph G. Then, after augmenting $\delta = \min(k - |f|, \min\{c(e)|e \in p\}) > 0$ units of flow through a shortest path p in the residual graph G_f, the new returned flow f' is a min-cost flow. Moreover, SSP computes a min-cost flow in each iteration, and it returns a min-cost flow with flow value $|f| = n$.*

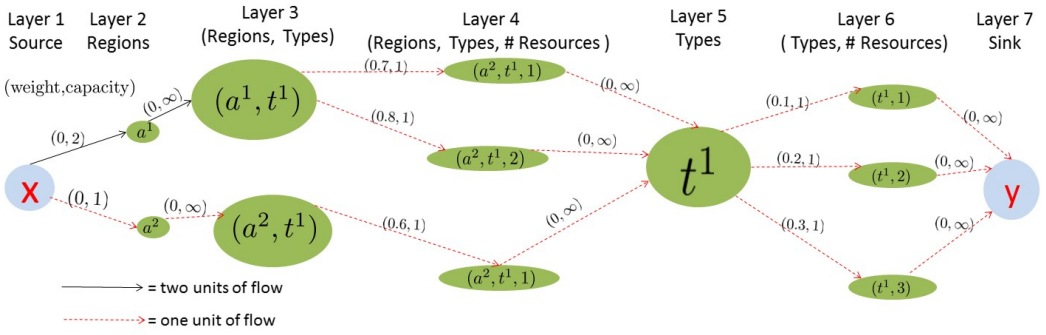


FIGURE 6.8: The flow $f_{L'}$, after augmenting 1 unit of flow along the shortest path depicted by red edges in Figure 6.7. The flow is corresponding to a placement L' with two resources in region 1 and a single resource in region 2, i.e., $L'^1_1 = 2, L'^2_1 = 1$.

Remark 6.2. An alternative algorithm for finding the shortest path is the *Dijkstra's algorithm*. Although Dijkstra's algorithm is faster than the Bellman-Ford algorithm, it requires the edge weights to be non-negative. Since the residual graph might contain negative edges, we must use the Bellman-Ford algorithm. However, if one uses the *node potentials* technique to optimize the SSP algorithm, then the edge weights will be non-negative. Thus, by using node potentials, one can use Dijkstra's algorithm and reduce the time complexity. This method is described later in Section 7.5.3, where we solved the general Static Placement Problem (SPP-4).

Running the SSP algorithm over the seven-layer graph G^7 will find a shortest path between source node x and sink y in each iteration. As all edges between (region, type) nodes (a^j, t^i) and (region,type, # resource) nodes have unit capacities ($c(e) = 1$), their residual capacity is 1 at most (i.e., $c_f(e) \leq c(e) = 1$) and thus SSP augments 1 unit of flow along the shortest path in every iteration.

Example 6.3. Suppose we run a single iteration of SSP over the residual graph depicted in Figure 6.7. SSP augments the flow by sending 1 unit of flow along the shortest path, which is depicted by the red edges. As a result, the flow f_L is changed to the one depicted in Figure 6.8, which is corresponding to a placement L' with two resources in region 1 and a single resource in region 2, i.e., $L'^1_1 = 2, L'^2_1 = 1$.

6.3.2.1 SSP in the context of the resource placement problem

Corollary 6.4 and Theorem 6.6 imply that in each iteration SSP computes a flow corresponding to some placement, as stated next:

Corollary 6.7. *Let f be a flow SSP computes in some iteration. Then f is a min-cost flow corresponding to a placement L, where $L^j_i = f^j_i$.*

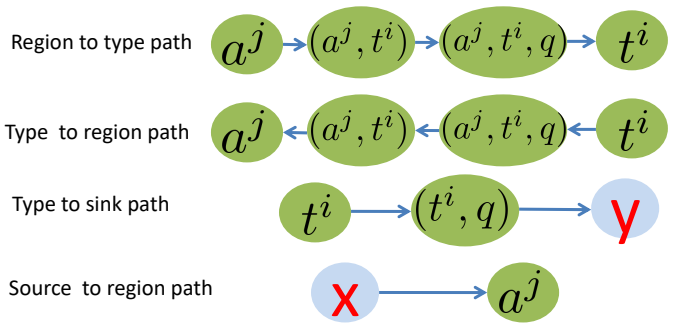


FIGURE 6.9: Definitions of monotone paths in G_f^7.

We will see later (formally in Lemma 6.10) that the augmentation of the flow f_L by the shortest path found by SSP is equivalent to finding the best possibility for adding a resource to a region in L, and then moving resources between different regions.

6.3.2.2 Complexity of SSP

The SSP algorithm on the seven-layer graph G^7 (presented in Section 6.2.3) changes the flow f in each iteration until f is a min-cost in G^7. Therefore, by Corollary 6.4 the placement representing the flow in region+type nodes $L_i^j = f_{opt}^{in}(a^j, t_i)$ is an optimal solution for the placement problem. As presented in Subsection 6.2.3, the time complexity of SSP on G^7 is $O(s^3m^2k^2)$. The Bipartite Graph algorithm presented next relies on SSP, and runs faster ($O(s^2mk(m+k))$).

6.3.3 The Bipartite-like Graph

To present the Bipartite Graph algorithm, we study the structure of the shortest paths between two nodes in the 7-level residual graph G_f^7.

Let f be a flow that the SSP algorithm calculates in some iteration. Let v_i represent a node in layer-i in G_f^7, for $1 \leq i \leq 7$. The directed path (v_2, v_3, v_4, v_5) in G_f^7 is called a *region-to-type path* since it is between a region node and a resource type node. Similarly, (v_1, v_2), (v_5, v_4, v_3, v_2), (v_5, v_6, v_7) are called a *source-to-region path*, a *type-to-region path*, and a *type-to-sink path*, respectively. Those paths, called *monotone paths*, are presented in Figure 6.9. Note that there are type-to-region paths in the residual graph G_f^7 which are composed of reverse edges of G^7.

The number of region-to-type paths in G_f^7 between a region node a and a resource type node t is $O(s)$. As we will see in the following analysis, G_f^7 possesses a special property. The structure of G^7 implies that on any iteration of SSP, only of these $O(s)$

paths needs to be considered. Focusing on this path will allow us to shrink the residual graph considerably to yield the much smaller bipartite graph. To do so, we say that a region-to-type path p between region node a and resource type node t is a *minimal path* if it is a path with the minimum weight among all the region-to-type paths between a and t. We denote the path by $a \overset{\min}{\to} t$ and its weight by $w_f(a \overset{\min}{\to} t)$. We denote similar definitions for a minimal path between resource type node t and region node a, between a resource type t and the sink y, and between the source s and a region node a by $t \overset{\min}{\to} a$, $t \overset{\min}{\to} y$ and $x \overset{\min}{\to} a$, respectively. For the analysis conducted in this section, we may concatenate between two minimal paths; for example paths $a \overset{\min}{\to} t \overset{\min}{\to} y$ represent the concatenation of paths $a \overset{\min}{\to} t$ and $t \overset{\min}{\to} y$.

Example 6.4. In the residual graph depicted in Figure 6.7, the weights of the source-to-region minimal paths are $w(x \overset{\min}{\to} a^1) = 0, w(x \overset{\min}{\to} a^2) = \infty$, the weights of the region-to-type minimal paths are $w(a^1 \overset{\min}{\to} t^1) = 0.8, w(a^2 \overset{\min}{\to} t^1) = \infty$, the weights of the region-to-type minimal paths are $w(t^1 \overset{\min}{\to} a^1) = -0.7, w(a^2 \overset{\min}{\to} t^1) = -0.6$, and the weights of the single type-to-sink minimal path is $w(t^1 \overset{\min}{\to} y) = 0.3$.

Remark 6.3. If there is more than one minimal path between two nodes v_1, v_2, then $v_1 \overset{\min}{\to} v_2$ is selected to be one of them.

In the next lemma, we see the importance of minimal paths to characterize the shortest path in G_f^7, allowing us to eliminate non-minimal paths when computing a shortest path:

Lemma 6.8. *Let f be a flow that the SSP algorithm calculates in some iteration over the residual graph G_f^7. Then, there is a shortest path between x and y, denoted by p_{opt}, which has a decomposition formula as follows:*

$$p_{opt} = x \overset{\min}{\to} a^{j_1} \overset{\min}{\to} t^{i_1} \overset{\min}{\to} a^{j_2} \ldots \overset{\min}{\to} a^{j_e} \overset{\min}{\to} t^{i_e} \overset{\min}{\to} y, \tag{6.3}$$

where x, y are respectively the sink and source of G_f^7, and a^{j_l}, t^{i_l} are, respectively, area and resource type nodes, for all $1 \leq l \leq e$.

For example, the shortest path in Figure 6.7 is $x \overset{\min}{\to} a^1 \overset{\min}{\to} t^1 \overset{\min}{\to} y$.

The reader may verify the correctness of the lemma resulting from the structure of G^7 and G_f^7. The proof is in Appendix 6.

The lemma helps understanding the *bipartite-like graph*, denoted as G_f^B. The bipartite-like graph is a four-layer graph that represents the minimal paths in G_f^7. The graph is composed of the following layers: the first layer includes the source node x, the second

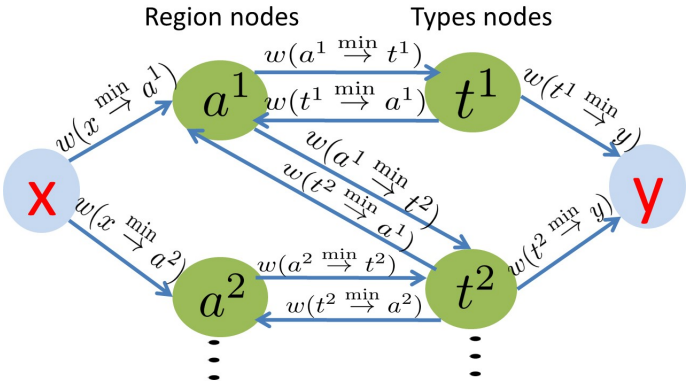


FIGURE 6.10: The bipartite-like graph G_f^B. For the sake of presentation, we omit the edges between region node a^2 and type node t^1.

layer comprises region nodes a^j, the third layer comprises resource type nodes t^i, and the last layer comprises the sink node y. The edge weights are determine by the minimal paths weight in G_f^7. The graph, depicted in Figure 6.10, resembles a bipartite graph, excluding the source and sink nodes.

To this end, using Lemma 6.8, we conclude that finding the shortest path in the bipartite-like graph G_f^B is equivalent to finding a shortest path in G_f^7, as stated in the next corollary:

Corollary 6.9. *Let f be a flow that SSP calculates in some iteration over the residual graph G_f^7. Then $p_{opt} = x \xrightarrow{\min} a^{j_1} \xrightarrow{\min} t^{i_1} \xrightarrow{\min} a^{j_2} \ldots \xrightarrow{\min} a^{j_e} \xrightarrow{\min} t^{i_e} \xrightarrow{\min} y$ is a shortest path in G_f^7 iff $\hat{p}_{opt} = (x, a^{j_1}, t^{i_1}, a^{j_2} \ldots, t^{i_e}, y)$ is a shortest path in G_f^B. Moreover, p_{opt} and $\hat{p}_{opt}$ have the same weight.*

In the next subsection, we will see how to calculate the bipartite-like graph edges G_f^B (i.e., the minimal path weight $w_f(v_1, v_2)$) in constant time ($O(1)$).

6.3.3.1 Efficiently calculating the bipartite-like graph edges

In addition for shrinking the residual seven-layer graph G_f^7 to a bipartite-like graph G_f^B, we will need to reduce the time complexity for constructing G_f^B. In the SSP algorithm we construct the residual graph G_f^7 in $O(smk)$ time. We will show how to construct the bipartite-like graph G_f^B only in $O(mk)$ time. The proofs of the corresponding Lemmas appear in Appendix 6.

Let $f = f_L$ be a flow that SSP calculates in some iteration. By Corollary 6.7, f is a flow corresponding to a placement L. We will prove that the weights of the minimal paths in G_f^7 (and therefore, the edge weights in $G_{f_L}^B$) depend only on the placement

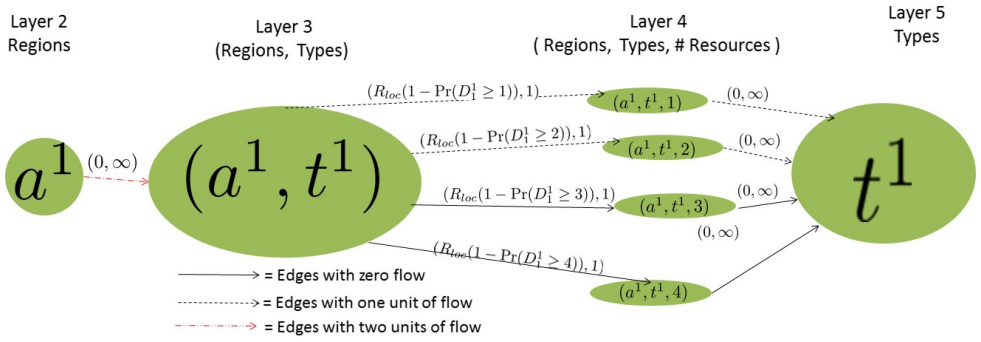


FIGURE 6.11: A subgraph of the flow of f_L where $L_1^1 = 2$.

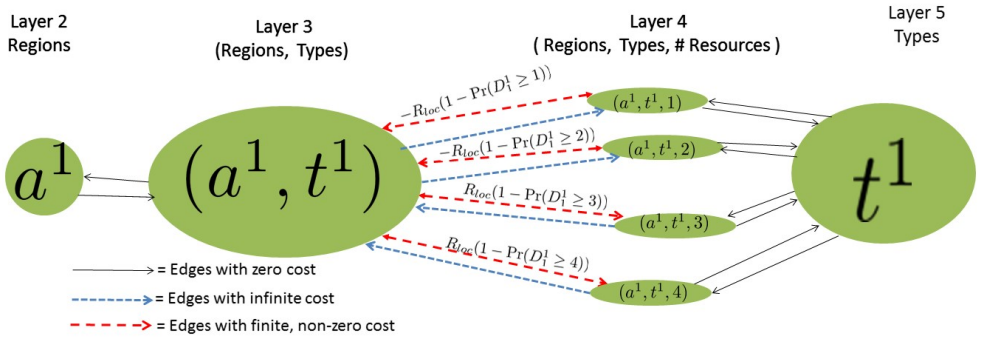


FIGURE 6.12: The corresponding residual graph of Figure 6.11.

quantities, i.e., the number of type i resources in region j (L_i^j). Moreover, the weight of every minimal path $w_f(v_1 \overset{\min}{\to} v_2)$ is computed in $O(1)$, as stated next:

Lemma 6.10. *Let f be a flow in G^7 corresponding to a placement L. The minimal path weights in G_f^7 are computed as follows:*
(a) $w_f(x \overset{\min}{\to} a^j) = 0$ *if* $L^j < s^j$ *and otherwise* ∞.
(b) $w_f(a^j \overset{\min}{\to} t^i) = R_{loc}(1 - \Pr(D_i^j \geq L_i^j + 1))$ *if* $L_i^j < s$ *and otherwise* ∞.
(c) $w_f(t^j \overset{\min}{\to} a^i) = -R_{loc}(1 - \Pr(D_i^j \geq L_i^j))$ *if* $L_i^j > 0$ *and otherwise* ∞.
(d) $w_f(t^i \overset{\min}{\to} y) = R_{loc}(1 - \Pr(D_i \geq L_i + 1))$ *if* $L_i < s$ *and otherwise* ∞.

Example 6.5. To see the correctness of the lemma, in Figure 6.11 we depict a subgraph of a flow $f = f_L$ corresponding to a placement L, where two resources of type 1 are placed in region 1 (i.e., $L_1^1 = 2$). We depict its corresponding residual graph in Figure 6.12. For the sake of presentation, edges with zero residual capacity (which have infinite weight) are omitted from the residual graph. There are two region-to-type paths from a^1 to t^1 with respective weights of $R_{loc}(1 - \Pr(D_1^1 \geq 3))$ and $R_{loc}(1 - \Pr(D_1^1 \geq 4))$. Of course, $R_{loc}(1 - \Pr(D_1^1 \geq 3)) \geq R_{loc}(1 - \Pr(D_1^1 \geq 4))$ and the minimal path is $R_{loc}(1-\Pr(D_1^1 \geq 3)) = R_{loc}(1-\Pr(D_1^1 \geq L_1^1+1))$. Similarly, we have $-R_{loc}(1-\Pr(D_1^1 \geq 2)) \geq -R_{loc}(1-\Pr(D_1^1 \geq 1))$, and thus the minimal type-to-region path between t^1 and a^1 is $-R_{loc}(1 - \Pr(D_1^1 \geq 2)) = -R_{loc}(1 - \Pr(D_1^1 \geq L_1^1))$.

6.3.3.2 SSP update of the placement quantities

In every iteration of SSP, the flow f_L corresponding to one placement L is updated to another flow $f_{L'}$, which is corresponding to a different placement L' (See Corollary 6.7). We can characterize the modification that SSP does to the placement quantities L_i^j, as follows:

Lemma 6.11. *Let f be a flow corresponding to L that SSP computes in some iteration. Suppose that SSP augments the flow by the shortest path (found in Step 2 of SSP) $p_{opt} = x \overset{\min}{\to} a^{j_1} \overset{\min}{\to} t^{i_1} \overset{\min}{\to} a^{j_2} \ldots \overset{\min}{\to} a^{j_e} \overset{\min}{\to} t^{i_e} \overset{\min}{\to} y$. Then, SSP updates the flow f to another flow f' that is corresponding to a placement L', where L' is set as follows: 1) $L'^{j_l}_{i_l} \leftarrow L^{j_l}_{i_l} + 1$ for $1 \leq l \leq e$. 2) $L'^{j_{l+1}}_{i_l} \leftarrow L^{j_{l+1}}_{i_l} - 1$ for $1 \leq l \leq e-1$. 3) $L'^j_i \leftarrow L^j_i$ for $(i,j) \neq (i_l, j_l), (i_l, j_{l+1})$.*

Remark 6.4. From Lemma 6.11, we derive that SPP changes the placement L corresponding to the flow f (i.e., $L_i^j = f_i^j$) as follows: 1) SSP adds a resource of type j_e to region t_e, and 2) SSP moves a resource of type i_l from regions j_{l+1} to region j_l for every $1 \leq e \leq l-1$.

From Lemmas 6.10, 6.11 and Corollary 6.9, we can construct the BG algorithm described in the next subsection.

6.3.4 The Bipartite Graph algorithm

We first initiate the zero placement $L_i^j = 0$ for all regions $1 \leq j \leq k$ and resource types $1 \leq i \leq m$. In each iteration, the algorithm performs the following steps: 1) Calculate the weight of the graph edges of $G^B_{f_L}$, as described in Lemma 6.10. 2) Find a shortest $x-y$ path p_{opt} in G^B_f using Bellman-Ford. 3) Update the placement quantities L_i^j as described in Lemma 6.11. Since G^B_f satisfies the equivalent path property (see Corollary 6.9), the weight of an edge in G^B_f is calculated as the weight of the shortest path in G^7_f (see Lemma 6.10). The algorithm stops after s iterations (s is the total storage value). The resulting placement, L_i^j, by Corollary 6.4 solves the placement problem.

The number of iterations the BG algorithm performs is $O(s)$. Steps 1) and 3) takes linear time (at most $O(|V^B| + |E^B|)$, where $G^B_f = (V^B, E^B)$ is the seven-layer graph), while Step 2) takes $O(|V^B||E^B|)$. Since the bipartite-like graph G^B_f contains $|V^B| = O(m+k)$ vertices (k is the number of regions, m number of resource types) and $|E^B| = O(mk)$ edges, then the complexity of BG is $O(smk(m+k))$, which is faster than applying SSP to G^7, whose complexity is $O(s^3m^2k^2)$.

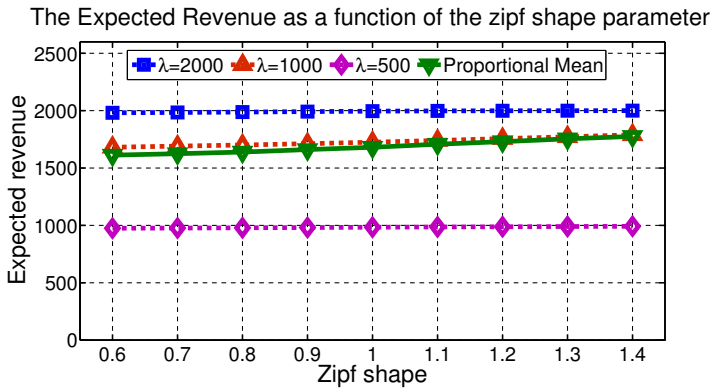


FIGURE 6.13: The optimal performance under deficit, balanced and surplus modes.

Remark 6.5. We assume that computing the c.d.f values $\Pr(D \leq x)$ for every random variable D and integer x takes $O(1)$ time. Otherwise, if it takes $O(e)$ time to compute the c.d.f values – then the complexity of the algorithm is $O(smk(m+k+e))$.

6.4 Numerical Examples

In this section, we evaluate the Bipartite Graph algorithm performance in order to study the following subjects: 1) The performance of the optimal placement under various scenarios. 2) A comparison of the optimal placement performance to that of a commonly used placement – the *Proportional Mean* placement.

We consider a demand that follows a Zipf distribution (which is consistent with prior analytical works [11, 68] and VoD empirical results [6, 69]). This means that there exists a real number $e > 0$ such that the probability for a single request to demand a type i resource is $p_i = \frac{1}{i^e \cdot H}$ for all resource types $1 \leq i \leq m$, where $H = \sum_{j=1}^{m} \frac{1}{j^e}$. We assume that the demand in region j is proportional to the region's storage, i.e., the probability for a request to originate from region j is $q^j = \frac{s^j}{s}$. To this end, we choose the demand D_i^j for type i resources originating from region j to be a Poisson distribution with a rate of $p_i \cdot q^j \cdot \lambda$, where λ is a constant number representing the expected number of requests.

In Figure 6.13, we choose the number of resource types to be $m = 100$, with revenue parameters of $R_{glo} = R_{loc} = 1$. Since it is unclear how future demands will behave (prior references mentioned above used a Zipf parameter of values between 0.56 and 1.5, depending on the study), we consider a wide range of Zipf distributions and vary the Zipf parameter from 0.6 to 1.4. The number of regions is $k = 3$, containing storage of $s^1 = 500$, $s^2 = 300$ and $s^3 = 200$, and thus the storage of the system is $s = 1000$. We plot the expected revenue (profit) of the optimal placement under surplus scenario

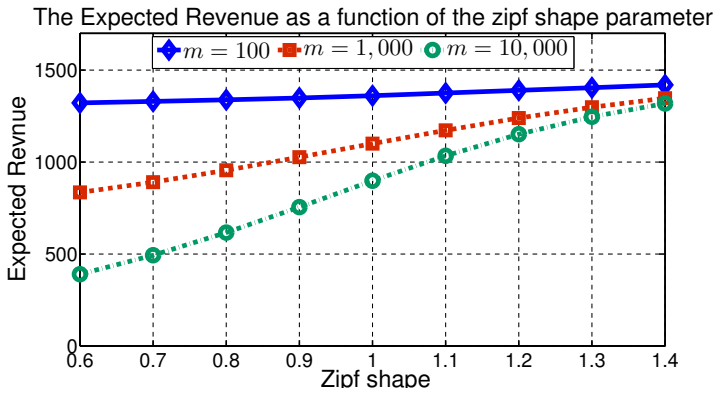


FIGURE 6.14: A performance comparison between optimal placement revenues (profits) when the number of resource types are set to $m = 100$, $m = 1,000$ and $m = 10,000$, respectively.

($\lambda = 500$), under balanced scenario ($\lambda = 1000$), and deficit scenario ($\lambda = 2000$). We compute the expected revenue (profit) using Eq. (6.1). We also plot the performance of Proportional Mean placement in the balanced scenario.

An important result from Figure 6.13 is the optimal placement in the different configurations. Its revenue in surplus mode is close to 1000, since almost every request is granted locally, contributing $R_{loc} + R_{glo} = 2$. In the deficit scenario, the revenue is almost 2000, since about 1000 requests are granted, and all locally. In the balanced scenario, some of the requests are granted locally, and others are granted remotely.

In Figure 6.14, we use the balanced settings, but increase the variety of resource types to $m = 1,000$ and $m = 10,000$. We can see three major results: 1) As the number of resource types increases, the optimal placement revenue decreases. This is implied from the increasing number of esoteric resources (resources with rank larger than the total storage value, which we denoted by s), which cannot be satisfied. 2) As the Zipf parameter increases, the optimal placement's revenue increases. This can be explained by the fact that the number of esoteric resources decreases as the Zipf parameter increases.

The Proportional Mean placement, used in several articles ([11, 68, 74]), places its replicas $\{L_i^j\}$ proportionally to the mean value of the demand distribution D_i^j. We compare the expected profit of the Proportional Mean placement with the profit of the optimal placement.

In Figure 6.13, we see that in the balanced mode, the Proportional Mean placement performance is close to the optimal placement. In contrast, in the scenario depicted in Figure 6.15, we see that the Proportional Mean performance degrades when the number of resource types approaches to infinity, while the optimal placement revenue stabilizes. In that scenario we assume a single-region system ($k = 1$) and the storage of region 1

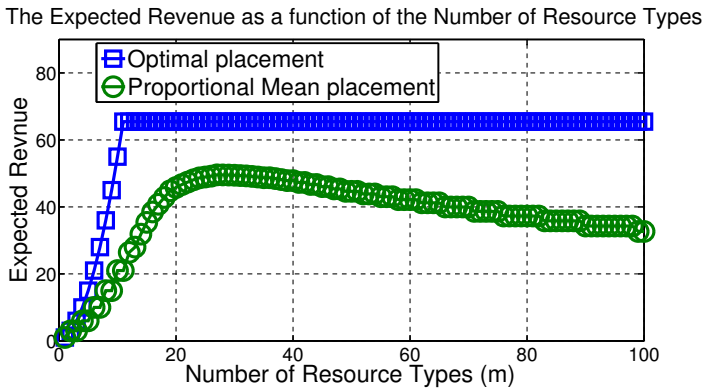


FIGURE 6.15: Performance comparison of the proportional mean placement and the optimal placement in back-up services applications.

is $s^1 = 500$. The revenue constants are $R_{glo} = 0$ and $R_{loc} = 1$. Therefore, the expected revenue represents the number of requests granted in region 1. The demand for type i resource is 0 with probability of $1 - 1/i$ ($\Pr(D_i^1 = 0) = 1 - 1/i$) and i with probability $1/i$ ($\Pr(D_i^1 = i) = 1/i$).

The demand distributions $(D_1^1, D_2^1, \ldots, D_n^1)$ reflect a scenario of an application that provides back-up services to m companies. If a failure occurs in one of the companies, the application will provide back up to the users in the company. We can assume that as the company becomes larger, it is better maintained and invulnerable to failures, which is reflected in its failure probability proportional to its size, namely $1/i$.

The figure (Figure 6.15) demonstrates that when the variance of some of the demands is high, the Proportional Mean placement does not perform well.

Chapter 7

The General Static Placement Problem: Unconstrained-Capacity System Optimizing the General Profit Function (SPP-4)

In this chapter, we present an algorithm for the placement problem, given a system that is not necessarily capacity-constrained; that means, the system can place an unbounded amount of resources in a region. In addition, the operator optimizes the general profit function, as opposed to the simple profit function, as done in the previous chapters. To this end, we present the Generalized-Bipartite Graph (G-BG) algorithm, which solves the problem. This work is presented in a yet-to-be-published article [14].

The need to deal with infinite capacities requires us to (somewhat) modify the analysis methodology used in Chapter 6. The differences are to be described in Sections 7.3.1, 7.4.

7.1 Problem Formulation

Let $D = \{D_i^j\}$ be the demand for type i resources in region j. In this chapter, we solve the placement problem where systems may have unconstrained storage, and the number of resources in every region is not restricted. Our placement problem is to find a finite resource placement L that maximizes the generalized profit $P(L, D)$ (as explained in

Section 3.1), which equals

$$P(L,D) = \sum_{i=1}^{m}\sum_{j=1}^{k}\zeta_i^j(L_i^j, D_i^j) + \sum_{i=1}^{m}\zeta_i(L_i, D_i) + \sum_{j=1}^{k}\zeta^j(L^j), \tag{7.1}$$

where,

$$\zeta_i^j(L_i^j, D_i^j) = R_i^j \cdot E_{D_i^j}[\min(L_i^j \cdot B_i, D_i^j)] - C_i^j(L_i^j), \tag{7.2}$$

$$\zeta_i(L_i, D_i) = R_i \cdot E_{D_i}[\min(L_i \cdot B_i, D_i)] - C_i(L_i), \tag{7.3}$$

$$\zeta^j(L^j) = -C^j(L^j), \tag{7.4}$$

are the marginal functions.

The placement problem in this chapter is formulated as follows:

The General Static Placement Problem (SPP-4)	
Assumptions:	None.
Input:	A demand $D = \{D_i^j\}$.
Problem:	Find the placement L_{opt} that maximizes the general profit in Eq. (6.1).

We assume that there is at least one finite solution for the problem. Note that there is no constraint over the number of resources a placement has. This makes the task of finding an optimal solution a hard one; the set of solutions contains an infinite number of placements. This leads to difficulties when using a reduction of this to a min-cost flow problem (which we present later in Section 7.3). We will explain these difficulties in Section 7.4.

7.2 Expressing the Profit by its Marginal-Differential Functions

As we recall from Section 3.2, given a demand D, we denote the **conditional-marginal functions** by $g_i(n) = \zeta_i(n, D_i^j), g_i^j(n) = \zeta^j(n, D_i^j)$ and $g^j(n) = \zeta^j(n)$. By Eq. (7.1) the profit equals the sum of the conditional-marginal functions, i.e.,

$$P(L,D) = \sum_{j=1}^{k} g^j(L^j) + \sum_{i=1}^{m} g_i(L_i) + \sum_{j=1}^{k}\sum_{i=1}^{m} g_i^j(L_i^j). \tag{7.5}$$

Given a function f, we denote its **differential** by $\Delta f(n) = f(n) - f(n-1)$. The differential of the conditional-marginal functions Δg are called the **marginal-differential** functions. For every function f, the sum of its differentials $\sum_{n=1}^{k} \Delta f(n)$ is a telescopic series that equals $\sum_{n=1}^{k} \Delta f(n) = f(k) - f(0)$. Thus, we can express the profit in Eq. (7.5) using the marginal-differential functions Δg as follows:

$$P(L, D) = c + \sum_{j=1}^{k} \sum_{n=1}^{L^j} \Delta g^j(n) + \sum_{j=1}^{k} \sum_{i=1}^{m} \sum_{n=1}^{L_i^j} \Delta g_i^j(n) + \sum_{i=1}^{m} \sum_{n=1}^{L_i} \Delta g_i(n). \tag{7.6}$$

where $c = \sum_{j=1}^{k} g^j(0) + \sum_{j=1}^{k} \sum_{i=1}^{m} g_i^j(0) + \sum_{i=1}^{m} g_i(0)$ is a constant that does not depend on the placement L. According to the concavity assumption (Section 3.2), the marginal-differential functions Δg_i^j are all monotonically non-increasing functions.

Remark 7.1. According to Eqs. (7.2)-(7.4), and Theorem 3.1, we can compute the marginal-differential functions directly using the model parameters, which are equal to

$$\Delta g_i^j(n) = R_i^j \cdot \sum_{k=(n-1)\cdot B_i+1}^{n \cdot B_i} \Pr(D_i^j \geq k) - C_i^j(n) + C_i^j(n-1), \tag{7.7}$$

$$\Delta g_i(n) = R_i \cdot \sum_{k=(n-1)\cdot B_i+1}^{n \cdot B_i} \Pr(D_i \geq k) - C_i(n) + C_i(n-1), \tag{7.8}$$

$$\Delta g^j(n) = -C^j(n) + C^j(n-1). \tag{7.9}$$

In the next sections, we will show how to find a placement L_{opt} that maximizes the profit in Eq. (7.6).

7.3 A Reduction to a Flow Problem over an Infinite Graph

In this section, we show how to reduce the placement problem of this chapter (SPP-4), into a min-cost flow problem over a graph that contains an infinite number of nodes and edges and an unknown required value. The reduction is similar to the one done in the previous chapter (SPP-3), i.e., the reduction of capacity-constrained placement problem optimizing the simple profit function (SPP-3) to the seven-layer graph G^7. In Section 7.3.1 describe next the differences between these reductions, and why the solution of the previous chapter SPP-3 do not apply to the problem (SPP-4). In Section 7.3.2, we recall the min-cost flow problem, and in Section 7.3.3 we describe the formal.

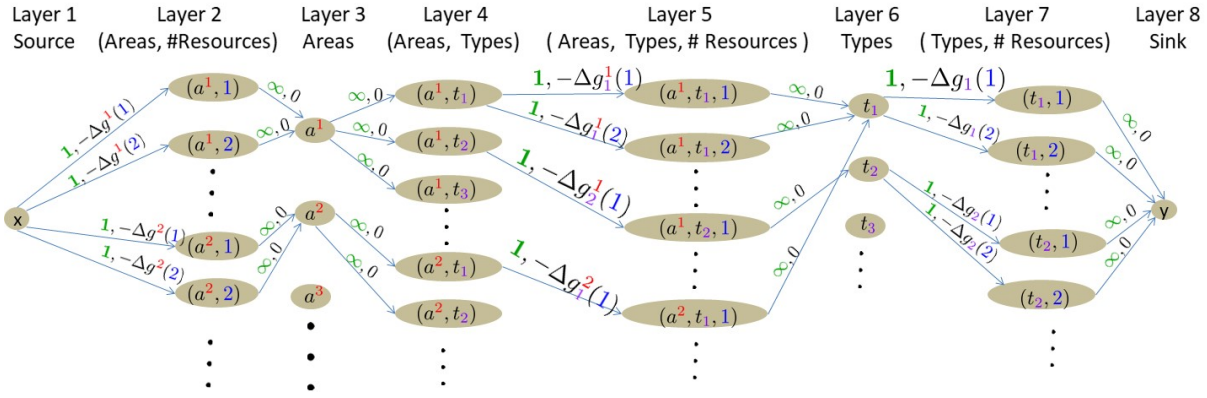


FIGURE 7.1: The eight-layer G^8 graph.

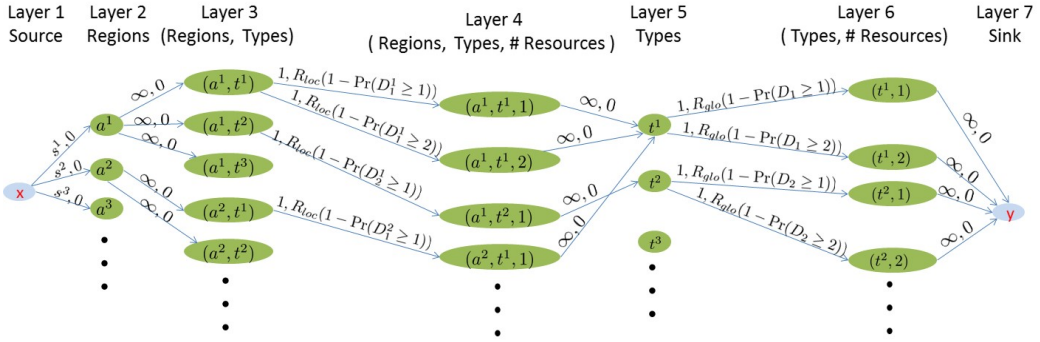


FIGURE 6.1: The seven-layer graph G^7 of SPP-3 (repeated from page 50).

7.3.1 The differences between the reduction of SPP-3 (Figure 6.1) and the reduction of SPP-4 (Figure 7.1)

We reduce the unconstrained-capacity placement problem optimizing the general profit function (SPP-4) to a min-cost problem using an eight-layer graph G^8 as given in Figure 7.1. The reduction is similar to the one presented in Section 6.2, where the capacity-constrained placement problem optimizing the simple profit function (SPP-3) is reduced to a min-cost flow problem. The difference between both reductions are described as followed:

1. The eight-layer graph G^8 contains an additional layer, which represents the areal costs (expressed by the marginal-differential function $\Delta g^j()$ for every region j) associated with the general profit function. The seven-layer graph G^7, in contrast, does not contain that layer. This is needed since in SPP-4, the goal is to optimize a general profit function, which incorporates areal costs, while in SPP-3 the goal is to optimize a simple profit function (as described in Chapter 3), which does not incorporate areal costs.

2. As the capacity of each region in SPP-4 is unconstrained, the eight-layer graph G^8 contains infinite number of edges and nodes. The capacity in every region in

SPP-3 is limited, and thus it is reduced to a finite graph G^7.

3. In SPP-4, the goal is to optimize a general profit function, while in SPP-3 the goal is to optimize a simple profit function (as described in Chapter 3). Thus, the eight-layer graph G^8 (corresponding to SPP-4) might have edges with a negative weight, while the weights of the edges in the seven-layer graph G^7 (corresponding to SPP-3) are all non-negative[1].

4. In addition, as the capacity in each region in SPP-4 is unconstrained, the problem is reduced to a min-cost flow problem with an **unknown flow value**. This is as opposed to the reduction presented in the previous chapter, where the required flow is equal to the to the total capacity of the system.

These differences make the current placement problem (SPP-4) more difficult than the one presented in the previous chapter (SPP-3). Thus, the same solutions we used in SPP-3 cannot apply directly to SPP-4. Moreover, we show later in Section 7.4 that standard min-cost flow algorithms cannot be used to solve the min-cost flow problem corresponding to SPP-4, and we need to use new theoretical techniques to solve the problem.

7.3.2 Preliminaries – the min-cost flow problem

Below we re-describe the **min-cost flow problem** [71], which is identical to the description given in Section 6.2.2. In this problem, one considers a directed graph $G = (V, E)$ where every edge $e \in E$ has a non-negative integer *capacity* $c(e)$ and a real-value *weight* $w(e)$ (also called *cost*). The graph must contain two different nodes: a source node x and a sink node y. An *x-y-flow* $f : E \to R^+$ is defined on the graph edges $(v, v') \in E$ in the same way as defined in the max-flow problem. That means, the flow must satisfy the following properties: 1) *Capacity constraint:* for each edge e, we have $0 \leq f(e) \leq c(e)$. 2) *Conservation of flows:* for every vertex $v \in V \setminus \{x, y\}$ we have $\sum_{(v',v)\in E} f(v', v) = \sum_{(v,v')\in E} f(v, v')$. In addition to the standard definitions, we define the *flow in node* $v \neq x, y$ as the income flow (and by conservation of flows, the outcome flow) to (from) node v. We denote it by $f^{in}(v)$, which equals $f^{in}(v) = \sum_{(v,v')\in E} f(v, v')(= \sum_{(v',v)\in E} f(v', v) = f^{out}(v))$. The *flow value* of f, as defined in the max-flow problem, is $|f| = \sum_{(x,v)\in E} f(x, v) = \sum_{(v,y)\in E} f(v, y)$. The *weight (or cost) of flow* f is $w(f) = \sum_{e\in E} f(e)w(e)$.

[1] In a finite graph, one can use the network transformations done in [70] to transformed the edge weights to be non-negative. However, these network transformations are not suitable for the infinite graph G^8.

Given a parameter n, the classic *minimum-cost flow* problem is to find a flow f_{opt} of value n that has minimum weight among all flows of value n. This means that for every flow f' where $|f'| = |f_{opt}| = n$ we have $w(f_{opt}) \leq w(f')$. It is well-known that if the capacities $c(e)$ of a min-cost flow network are integers, then there is a min-cost flow f where the flow in every edge is integer; a proof can be found in [70], Theorem 9.10.

7.3.3 The reduction to the eight-layer graph

This part is similar to Section 6.2.3.

We reduce the placement problem to a min-cost problem using the eight-layer graph given in Figure 7.1. On every we edge we depict a pair $c(e), w(e)$ so that $c(e)$ is the capacity function (presented in an olive green color) and $w(e)$ is the weight function. The graph is composed of the following eight layers: the **source** x, the **(area, #resources)** layer, the **area** layer, the **(area, type)** layer, the **(area, type, #resources)** layer, the **type** layer, the **(type, #resources)** layer, and finally the **sink** y.

In the graph, we denote area j by a^j, type i by t_i, and #resources by a number. The (area, #resources), (area, type, #resources) and (type, #resources) layers, for example, are respectively composed of an infinite number of nodes (a^j, n) (where $1 \leq j \leq k$ and $n \in \mathbb{N}$), (a^j, t_i, n) (where $1 \leq j \leq k$, $1 \leq i \leq m$,$n \in \mathbb{N}$) and (t_i, n) (where $1 \leq i \leq m$ and $n \in \mathbb{N}$). The weight of the entering edges to nodes (a^j, n), (a^j, t_i, n) and (t_i, n) are respectively equal to the marginal-differential function multiplied by -1 i.e., $-\Delta g^j(n)$, $-\Delta g_i^j(n)$ and $-\Delta g_i(n)$. The weight of these edges can be **negative**, and have a capacity of 1. All other edges have zero weight and infinite capacity.

In Appendix d, we show that for every placement L there is a flow f_L, called **the corresponding flow of** L such that the profit of L is equal to a constant c (defined in Eq. (7.6)) minus the weight of f_L in G^8, i.e., $P(L, D) = c - w(f_L)$. In addition, given an arbitrary flow f in G^8, there is a placement L such that the corresponding flow of L has a cost lower than f, i.e., $w(f) \geq w(f_L)$. This is similar to Lemmas 6.2, 6.3 in Section 6.2.3. To show this – we use the fact that the marginal-differential weights $-\Delta g$ are monotonically increasing. These lemmas lead to the following corollaries:

Corollary 7.1. *Let f_{opt} be an integer min-cost flow of G^8. Let L be the placement created from the values of the flow in (region, type) nodes $L_i^j = f_{opt}^{in}(a^j, t^i)$. Then f_{opt} is corresponding to L.*

Corollary 7.2. *Suppose that n_{opt} is the minimum number of resources an optimal solution can contain. Let f_{opt} be the min-cost flow in G^8 with a required flow of $|f| = n_{opt}$. Let L be a placement whose quantities are equal to the flow in nodes (a^j, t_i), i.e., $L_i^j = f_{opt}^{in}(a^i, t^j)$. Then L solves the placement problem (SPP-4).*

The correctness of these corollaries is also presented in Appendix d.

7.4 Difficulties in Solving the Flow problem and the Road-Map of the Solution

We cannot directly use standard min-cost flow algorithms (such as SSP) over G^8 to find the optimal placement, and need to develop new techniques. This stems from three major reasons:

1. Graph G^8 might have edges with negative weight, while min-cost flow algorithms work only over graphs with non-negative weights[2].
2. The required flow n_{opt} is not a given parameter, while many min-cost flow algorithms rely on the required flow whenever they need to stop.
3. Graph G^8 contain infinite number of edges and nodes, where the min-cost flow algorithms work over a finite graph.

We are not aware of prior works that dealt with these aspects of min cost flow.

The solutions for these issues, are respectively presented as follows:

1. To solve the negative edge problem, we present a modified variation of Successive Shortest Path (SSP) over a general acyclic graph G that does not require the edges to have non-negative weight. We call this algorithm **Successive Shortest Path with Negative Edges (SSP-NE)**. SSP-NE uses the well-know **node potentials** technique in order to work over non-negative weight edges.
2. To avoid relying on the required flow n_{opt}, we modify SSP-NE and present a **non-negative weight criterion** that provides an alternative **stopping rule** for the algorithm.
3. To reduce the size of the infinite graph G^8, we transform G^8 to a finite graph, forming a **finite** bipartite-like graph G_f^B. The transformation is similar to that in Section 6.3.

To this end, we describe the **Generalized-Bipartite Graph (G-BG)** algorithm that imitates SSP-NE with the non-negative weight criterion; this means that each iteration in G-BG over bipartite-like graph G_f^B corresponds to the equivalent iteration in SSP-NE over the eight-layer graph G^8, and G-BG stops when the non-negative weight criterion is violated.

[2] Unless one uses network transformations as done in [70], which are not suitable for the infinite graph G^8.

In the next section we describe the min-cost flow preliminaries needed to present the Generalized-Bipartite Graph (G-BG), i.e., the classic SSP algorithm and node potentials. Then, in Section 7.6, we describe SSP-NE, and Section 7.7 describes the non-negative weight criterion. In Section 7.8, we show how to shrink the graph of G^8 to the finite bipartite-like graph G_f^B, and finally Section 7.9 presents G-BG.

7.5 Preliminaries – The Classic SSP Algorithm and Node Potentials

To present the well-known classic SSP, in Section 7.5.1 we describe the well-known residual graph that the classic SSP uses (which is similar to one defined in Section 6.3.1). In Section 7.5.2, we describe the node potentials technique and shortest path algorithms. In Section 7.5.3, we describe the classic SSP algorithm, using node potentials.

7.5.1 The residual graph

This part is similar to Section 6.3.1. Given a flow f on a graph $G = (V, E)$, the Successive Shortest Path (SSP) uses the **residual graph** $G_f = (V, E_f)$. Every edge in the residual graph edges is associated with weight w_f and capacity c_f (also called the *residual capacity*). The residual graph G_f is constructed from G and f by the following steps: 1) Add to G_f edges from G, such that every edge $(v, v') \in E$ will have weight $w_f(v, v') = w(v, v')$ and capacity $c_f(v, v') = c(v, v') - f(v, v')$. 2) Add the reverse edges of G. That means, if $(v, v') \in E$, then add the reverse edge (v', v) to G_f with weight $w_f(v', v) = -w(v, v')$ and capacity $c_f(v', v) = f(v, v')$. Note that for every edge e in G_f we have $c(e) \geq 0$. 3) Set the weight of every edge $e \in E_f$ with zero residual capacity ($c_f(e) = 0$) to $w_f(e) = \infty$.

An example of a residual graph can be seen in Figure 6.6.

7.5.2 The shortest path algorithm building block and the node potentials

Shortest path algorithms are key building blocks for SSP. Two notable algorithms to calculate the shortest paths from one source v to all other vertices, are the **Bellman-Ford** and **Dijkstra's** algorithms. Dijkstra can only run on graphs with non-negative edges. Bellman-Ford can also run on graphs with negative edges (assuming it does not contain a negative cycle), but has a higher complexity than Dijkstra. The running time

of Bellman-Ford on graph $G = (V, E)$ is $O(|E| \cdot |V|)$ compared to $O(|E| + |V| \cdot \log |V|)$ for Dijkstra. More information on these two algorithms can be found in [72].

In addition, if G is a directed acyclic graph i.e., G does not contain cycles, then one can compute shortest paths in linear time ($O(|V| + |E|)$) using topological ordering.

Next, we describe the **node potentials**, which serves two major goals:

- We use node potentials to prove the correctness of our algorithm, i.e., it ensures that SSP-NE will work on acyclic graphs with non-negative edges.
- Reducing the time complexity of well-known min-cost flow algorithms, such as the classic SSP.

We associate with every vertex (node) $v \in V$ in the residual graph $G_f = (V, E_f)$ a potential $\pi(v)$. Given the potential function $\pi : V \to R$ the **reduced weight (cost)** of an edge $(v_1, v_2) = e \in E_f$ is defined as $w_f^\pi(v_1, v_2) = w_f(v_1, v_2) - \pi(v_1) + \pi(v_2)$. A node potential function π satisfies the **Reduced Weight (Cost) Optimality Conditions** if the reduced weight of every edge $e \in E_f$ is non-negative (i.e., $w_f^\pi(e) \geq 0$).

The reduced weight optimality conditions are presented in the following theorem. We will use it for proving the correctness of SSP-NE.

Theorem 7.3 (Reduced Weight Optimality Conditions). *Let f be a flow. Then f is a min-cost flow iff there is a node potential function $\pi : V \to R$ that satisfies the reduced weight optimality conditions.*

A proof can be found at [70].

7.5.3 A description of SSP, with node potentials

It is well-known that the node potentials can be incorporated with the classic SSP algorithm in order to reduce time complexity [70]. Bellow we present an implementation of the classic SSP algorithm (which was described in Section 6.3.2) with node potentials.

Given a graph G with non-negative edge weights, SSP works as follows: In the initial step, SSP assigns the zero flow $f := 0$ (i.e., $f(e) = 0$ for all $e \in E$) with flow value $|f| = 0$ and constructs the residual graph G_f. The algorithm sets the node potentials π to zero, i.e., $\pi(v) = 0$ for every vertex v in the residual graph G_f.

SSP works iteratively, and in the i^{th} iteration it calculates a min-cost flow, where the flow value is larger than or equal to i (in the G^8 graph, it equals i). SSP executes the

following steps in each iteration: 1) Check if the flow value $|f|$ is equal to the required flow n. If so, then f is the optimal flow and SSP terminates. 2) Use Dijkstra's algorithm to obtain the shortest path distances from the source x on G_f with respect to the non-negative reduced weights w^π. We retrieve the shortest path p between x and y, which is called the **augmenting path**. If the shortest path's weight is infinity (i.e., $w_f^\pi(p) = \infty$) then the maximum flow value of the graph G is strictly less than the required flow n, and SSP returns an error. 3) We augment $\delta = \min(n - |f|, \min\{c(e)|e \in E\}) > 0$ units of flow through p. That means that if $(v, v') = e \in p$ is in the original graph (i.e., $e \in E$) then we update $f(e) \leftarrow f(e) + \delta$, and if the reverse edge in G (i.e., $(v', v) \in E$) then we update $f(v', v) \leftarrow f(v', v) - \delta$. 4) The node potentials π are updated to the previous node potentials minus the shortest path distance vector, i.e., $\pi \doteq \pi - d$, where $d(v)$ is the shortest path between x and v. 5) A new residual graph G_f is created according to the updated flow f.

Remark 7.2. The SSP version presented in Section 6.3.2 does not include the node potentials, as we discovered this technique later in our research. Using node potentials reduces the time complexity of SSP, as it uses Dijkstra's algorithm instead of the Bellman-Ford algorithm. Given a graph $G = (V, E)$ and a required flow n, the complexity of SSP with the node potentials is $O(n \cdot (|E| + |V| \cdot \log |V|))$, while the complexity of SSP without the node potentials is $O(n \cdot |E| \cdot |V|)$.

7.6 Successive Shortest Path with Negative Edges (SSP-NE)

The classic SSP cannot be used on weighted graph G with negative edges, and particularity cannot be used on the eight-layer graph G^8. Thus, we introduce the SSP-NE algorithm for solving the problem over acyclic graph instances, which include the eight-layer graph G^8. Node potentials, which are used in SSP-NE, are critical to prove the correctness of the algorithm.

As opposed to the classic SSP, in our version, the node potentials of the vertices are not automatically set to zero in the initial step. Rather, we use topological ordering on G_f^8 to calculate the shortest paths from source x with respect to the original weight function w_f. If $d(v)$ is the shortest path weight between x and a node v, then we set the initial node potential π to be $\pi(v) = -d(v)$. By using this setting, π satisfies the reduced weight optimality conditions, as shown in the following theorem (proved in the appendix).

Theorem 7.4. *Let G be an acyclic graph with arbitrary edge weights w. In the i^{th} iteration of SSP-NE, the flow f_i obtained at the end of the i^{th} iteration satisfies the*

reduced weight optimality conditions with respect to node potential π_i. Thus, f_i is a min-cost flow.

Also, as opposed to the classic SSP algorithm, SSP-NE does not stop given that the flow value $|f|$ is equal to some required flow k. In our version, SSP-NE will continue to run until the non-negative weight criterion is violated.

To compute the node potentials in the initial step, we can remove the reverse edges, which all have zero residual capacity ($c_f(e) = 0$) and infinite weight ($w_f(e) = \infty$), from the residual graph G_f. Doing so does not affect the shortest distance vector $d()$. The residual graph G_f becomes an acyclic graph, and thus we can use topological ordering to compute $d()$ and node potentials $\pi = -d$ in linear time $O(|V| + |E|)$.

The time complexity of SSP-NE is similar to that of SSP with node potentials. Suppose that SSP-NE runs for n iterations over an acyclic graph $G = (V, E)$. Then the overall time complexity is $O(n \cdot (|E| + |V| \cdot \log |V|))$ (note that $O(|E| + |V| \cdot \log |V|)$ is the time complexity of using Dijkstra's algorithm).

Remark 7.3. In a similar way, we can use Bellman-Ford algorithm instead of topological ordering to compute the shortest path in the initial step. However, in such case SSP-NE might run slower and its time complexity will be $O(|V| \cdot |E| + n \cdot (|E| + |V| \cdot \log |V|))$

7.7 The Non-Negative Weight Stopping Rule and the SSP-NE Convexity Theorem

Next, we introduce the non-negative weight criterion, a stopping rule that must be used to restrict the number of iterations of SSP-NE. The non-negative weight criterion checks in each iteration of SSP-NE if the shortest path weight is positive (that means, $w_f^{\pi}(p_{opt}) > 0$). If it is non-positive then SSP-NE terminates, and the placement associated with the flow, i.e., $L_i^j = f_i^j$, solves the placement problem.

This is a non-obvious stopping rule: One must show that when SSP-NE stops when reaching an optimal solution. That is, when the shortest path weight is $w^{\pi}(p_{opt}) \geq 0$ then the placement $L_i^j = f_i^j$ is an optimal placement among **all** placements.

To prove the optimality of the non-negative rule, we prove the SSP-NE convexity theorem. That means that the weights of the shortest paths, which are the marginal costs of the min-cost flow weight $w(f)$, are monotonically non-decreasing.

Theorem 7.5 (The SSP-NE Convexity theorem). *Let G be an acyclic graph with respect to a weight function w. Suppose f_i is the flow SSP-NE computes at the beginning of the i^{th} iteration. Let p_i be the shortest path in the i^{th} iteration of SSP-NE.*

Then, the weights of the shortest paths are monotonically non-decreasing i.e., $w_{f_i}(p_i) \leq w_{f_{i+1}}(p_{i+1})$.

Remark 7.4. We can similarly show convexity of the classic SSP.

Theorem 7.5 holds even with regard to the eight-layer graph G^8, which has infinitely many nodes. The theorem follows from proving that the shortest path weight between the source x and a vertex v is monotonically increasing.

Proof of Theorem 7.5. Let $d_f(u,v)$ denote the distance of the shortest path between u and v in the residual graph G_f with respect to the weight w_f. Similarly, $d_f^{\pi}(u,v)$ denotes the distance with respect to reduced weights w_f^{π}. By the definition of the weight function, we have $d_{f_i}(x,y) = w_{f_i}(p_i)$ in every iteration i of the SSP-NE algorithm. Also, by Theorem 6.6, there is a node potentials function π_{i+1} such that $w^{\pi_{i+1}}(e) \geq 0$ for every edge e in $G_{f_{i+1}}$.

Suppose by way of contradiction that in the i-iteration (of SSP-NE) the weights of the shortest paths are decreasing, i.e., $d_{f_i}(x,y) = w_{f_i}(p_i) > w_{f_{i+1}}(p_{i+1}) = d_{f_{i+1}}(x,y)$. We denote by $A = \{v \in V | d_{f_i}(x,v) > d_{f_{i+1}}(x,v)\}$ the non-empty set of vertices where the distance decreases among iterations (note that $y \in A$). We will choose a vertex $v_{min} \in A$ such that the distance $d_{f_{i+1}}^{p_{i+1}}(x,v)$ is minimal among all vertices in A. That means $v_{min} = \arg\min_{v \in A} d_{f_{i+1}}(x,v)$. The vertex v_{min} cannot be the source x, as $d_{f_i}^{p_{i+1}}(x,x) = d_{f_{i+1}}^{p_{i+1}}(x,x) = 0$ and thus $x \notin A$. Thus, there is a vertex u such that the shortest path between x and v_{min} in $G_{f_{i+1}}$ ends with (u, v_{min}).

For every path p between a and b, the weight of p with respect to the reduced weights $w^{\pi_{i+1}}$ is $w^{\pi}(p) = w(p) - \pi_{i+1}(a) + \pi_{i+1}(b)$ (a proof can be seen in [70]). Thus, working with reduced weights does not affect whether a path p between two nodes is the shortest path or not. In particular, the shortest path between x and v_{min} with respect to the weight function $w_{f_{i+1}}$ is the shortest path with respect to the node potentials $w_{f_{i+1}}^{\pi_{i+1}}$.

By the reduced weight optimality conditions, the reduced weight of (u,v) at the end of the $i+1$ iteration is non-negative, i.e., $w_{f_{i+1}}^{\pi_{i+1}}(u, v_{min}) \geq 0$. Therefore, the distance of x to u is shorter than the distance of x to v_{min} with respect the reduced weights $w_{f_{i+1}}^{\pi_{i+1}}$. By the minimality of v_{min} we derive that $u \notin A$, and,

$$d_{f_i}(x,u) \leq d_{f_{i+1}}(x,u), \tag{7.10}$$

i.e., the distance cannot decrease over iterations.

The weight of (u, v_{min}) in iteration $i+1$ ($w_{f_{i+1}}(u, v_{min})$) should be finite. Otherwise, the weight of the shortest path between x and v_{min}, which ends with the edge (u, v_{min}),

is equal to infinity, i.e., $d_{f_{i+1}}(x, v_{min}) = \infty$. Thus, by the definition of A we derive that $v_{min} \notin A$. The weight of (u, v_{min}) in the i^{th} iteration ($w_{f_i}(u, v_{min})$) can be equal to either ∞ (if the residual capacity is $c_{f_i}(u, v_{min}) = 0$) or it is equal to weight in the previous iteration, i.e., $w_{f_{i+1}}(u, v_{min}) = w_{f_i}(u, v_{min})$). Either way, the weight of (u, v_{min}) cannot decrease among iterations, i.e.,

$$w_{f_i}(u, v_{min}) \leq w_{f_{i+1}}(u, v_{min}), \tag{7.11}$$

Now suppose p is a path between x and v_{min} that uses a shortest path between x and u in G_{f_i} and then passes through the edge (u, v_{min}). Then,

$$d_{f_i}(x, v_{min}) \underbrace{\leq}_{\text{Definition of shortest path in } G_{f_i}} w_{f_i}(p) \underbrace{=}_{\text{Definition of } p} d_{f_i}(x, u) + w_{f_i}(u, v_{min})$$
$$\underbrace{\leq}_{\text{Eqs. (7.11), (7.10)}} d_{f_{i+1}}(x, u) + w_{f_{i+1}}(u, v_{min}) \underbrace{=}_{\text{Definition of } u} d_{f_{i+1}}(x, v_{min}).$$

Therefore, distance cannot decrease among iterations and $v_{min} \notin A$. This contradicts that $v_{min} \in A$.

□

Theorem 7.5 leads to the optimality of the stopping rule:

Corollary 7.6. *Let f_i be the flow that SSP-NE finds at the end of the l^{th} iteration. Let p_l be the shortest path in the l^{th} iteration of SSP-NE. Suppose that the shortest path p_{l_0} is the first non-negative path found by SSP-NE, i.e., $w_{f_{l_0-1}}(p_{l_0-1}) < 0 \leq w_{f_{l_0}}(p_{l_0})$. Then, f_{l_0} has minimum weight over all possible min-cost flows (i.e., $f_{l_0} = \arg\min_{f \text{ is min-cost flow}} w(f)$). Moreover, among min-cost flows with minimum weight, the flow value of f_{i_0} is minimal. Thus, the placement L associated with the flow f_{l_0} (i.e., $L_i^j = f_{l_0}^{in}(a^j, t_i)$) solves the placement problem.*

The correctness of the corollary is based on the fact that the cost of f_l is equal to $w(f_l) = w(f_{l-1}) + w_{f_{l-1}}(p_{l-1})$ and by the optimality of SSP-NE (Theorem 7.4). A formal proof can be seen in the appendix.

7.7.1 SSP-NE in the context of the resource placement problem

Changing the potentials of SSP does not affect the shortest path the algorithm selects (a proof can be found at [70] Property 2.4). Thus, SSP-NE affects the resource placement

L similarly to the way SSP affects it. In Section 6.3.2.1, we show that the shortest path found by SSP (and SSP-NE) over the residual graph $G^8_{f_L}$ is equivalent to the best possibility of adding a resource to L and then moving resources among different regions. If the best possibility is not profitable (i.e., it does not improve the profit of the placement L) then by the stopping rule SSP-NE stops.

7.8 The Bipartite-like Graph

This section generalizes the bipartite-like graph transformation of Section 6.3.3, Figure 6.10. Owing to the similarity to those transformations, we present the full details of this chapter in Appendix e. In addition, we present next the differences between those transformations.

The main purpose of the bipartite graph is to "get rid" of the infinite number of edges. To do this, one should recognize that for the sake of finding an augmenting flow at iteration i, between every pair of nodes at most one edge/path is a candidate for an augmenting path (the edge with the best weight). The reduction to the bipartite graphs uses that edge/path.

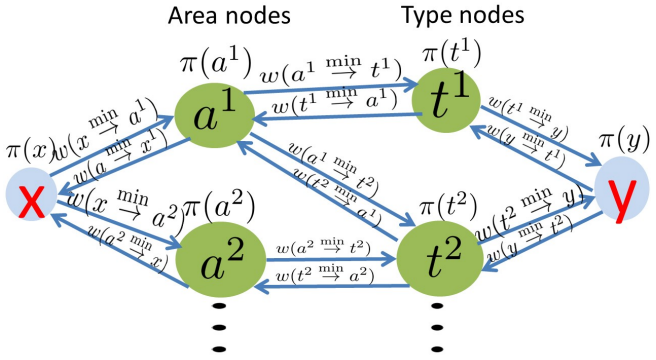


FIGURE 7.3: The bipartite-like graph G^B_f corresponding to the eight-layer graph G^8_f of SPP-4 (solved in this chapter). For the sake of presentation, we omit the edges between region node a^2 and type node t^1. The node potentials π are marked over the nodes.

The **bipartite-like graph**, denoted as G^B_f, is a directed graph, constructed from the eight-layer residual graph G^8_f. The graph is composed of the following four layers: the first layer includes the source node x, the second layer comprises area nodes a^j, the third layer comprises resource type nodes t^i, and the last layer comprises the sink node y. These are layers $1, 3, 6, 8$ in G^8_f. The graph, depicted in Figure 7.3, resembles a bipartite graph, excluding the source and sink nodes.

Between every two nodes of successive layers, we connect two edges of opposite direction. The edges, similar to Figure 6.10, represent the minimal paths in the residual graph G^8_f.

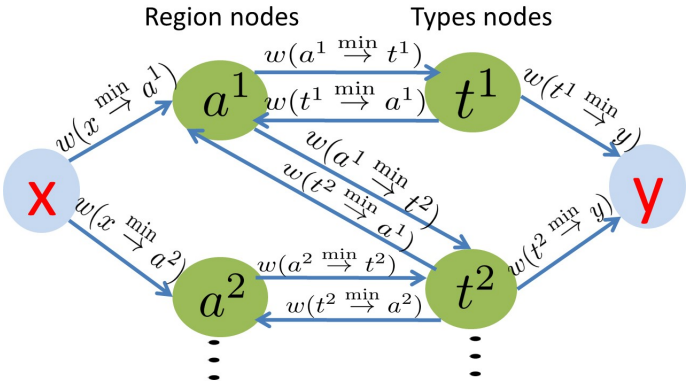


FIGURE 6.10: The bipartite-like graph G_f^B corresponding to the seven-layer graph G_f^7 of SPP-3 (solved in Section 6.3.3). Note that the node potentials are not defined over the nodes. (repeated from page 59).

The edge weight between u and v in the bipartite-like graph is equal to the weight of the minimal path between u and v in residual graph G_f^8, i.e., $w_f(u \overset{\min}{\to} v)$. For example, if we denote $p(n)$ as the forward path composed of nodes a^j, (a^j, t_i), (a^j, t_i, n) and t_i in the eight-layer residual graph G_f^8, then the edge weight between a^j and t_i in G_f^B is equal to the weight of the minimal path, i.e., $\min_n w_f(p(n))$.

7.8.1 The differences between Figure 6.10 and Figure 7.3

As opposed to Figure 6.10, in Figure 7.3 we set node potentials over the bipartite-like graph vertices, and define the reduced weight of an edge $u \overset{\min}{\to} v$ as $w_f^\pi(u \overset{\min}{\to} v) = w_f(u \overset{\min}{\to} v) - \pi(u) + \pi(v)$. The node potentials over the bipartite-like graph are identical to those we computed in SSP-NE. That means that if one denotes, respectively, π_i^{G-BG} and π_i^{SSP-NE} as the node potentials in the i-iteration of SSP-NE and G-BG (which we will define later), then $\pi_i^{G-BG}(v) = \pi_i^{SSP-NE}(v)$ for all $v \in V^B$. Doing so enables us to use potentials over the nodes (vertices) of G-BG.

In addition, we connect in Figure 7.3 the reverse edges between the region nodes a^j and the source node x, and between the sink node y and the type nodes t_i. Those edges are needed to compute the shortest path distance vector $d()$[3], which SSP-NE (and G-BG) uses for the computation of the node potentials. Those edges are omitted from Figure 6.10, as the classic SSP algorithm (and BG, the solution presented in the last chapter) uses only the edges of the shortest path between the sink x and the source y, which do not include those reverse edges (according to Lemma 6.8).

[3]We can show that removing the reverse edges between the region nodes a^j and the source node x does not affect the shortest path distance vector $d()$. However, removing the reverse edges between the sink node y and the type nodes t_i do affect the shortest path distance vector $d()$.

7.8.2 Correctness of the transformation, and efficient implementation

In Appendix e, we show the following properties: 1) The correctness of the transformation to the bipartite-like graph G_f^B. Formally, for every vertex v in the bipartite-like graph G_f^B, the weight of the shortest path between x and v in the residual graph G_f^8 is equal to the one in the bipartite-like graph G_f^B. 2) How to compute the edge weights of the bipartite-like graph $G_{f_L}^B$, given the placement quantities L_i^j. The weight of every edge can be computed in $O(1)$. 3) We show how augmenting the flow by the shortest path p_{opt} changes the flow f_L to another flow $f_{L'}$ corresponding to a new placement L'. The quantities of the new placement L' can be computed in linear time (by the size of the bipartite-like graph G_f^B). All these properties are similar to the one proven in previous chapter (Section 6.3.3), which are corresponding to bipartite-like graph of Figure 6.10.

7.9 The Generalized-Bipartite Graph (G-BG) Algorithm

G-BG imitates the behavior of SSP-NE, where every iteration in G-BG is corresponding to an equivalent iteration in SSP-NE. This is similar to BG (in Section 6.3.4) that imitates the classic SSP algorithm (without the node potentials). As opposed to BG, the improved solution (G-BG) uses the non-negative weight criterion and the node potentials (which are also used by SSP-NE). In addition, we will show that G-BG runs faster than BG.

7.9.1 The Generalized-Bipartite Graph (G-BG) description

We now present a sketch of G-BG: G-BG first initiates the zero placement $L_i^j = 0$ with its corresponding flow $f_L = 0$. Next, it constructs the initial bipartite-like graph $G_{f_L}^B$ (whose edge weights depend only the placement quantities). Similar to SSP-NE, to compute the initial node the algorithm removes all backward edges in the bipartite-like graph $G_{f_L}^B$ (those edges have infinite weight). The graph is now acyclic (as depicted in the Figure 7.5), and we can compute the shortest path distance vector $d()$ using topological ordering (See Section 7.5.2). We set the initial node potential to be $\pi = -d$.

In each iteration, G-BG will preform the following steps: (1) Find the shortest $x-y$ path in $G_{f_L}^B$ using Dijkstra's algorithm[4], with respect to the reduced weights w_f^{pi}. (2) Use the non-negative weight stopping rule, i.e., if the weight of the shortest path is non-negative then return the placement quantities L_i^j. Otherwise, the algorithm continues to the

[4]We cannot use topological ordering, as the bipartite-like graph might contain cycles.

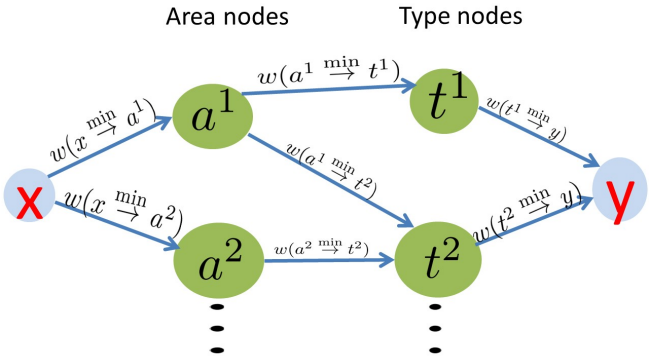


FIGURE 7.5: The bipartite-like graph $G^B_{f_L}$ corresponding to the zero placement $L := 0$, before computing the initial node potentials. We omit all backward edges, which have an infinite weight.

next step. (3) Update the placement quantities L^j_i (similar to the way BG updates the placement quantities; the exact details appear in Appendix e). (4) Update the weight of every edge in the bipartite-like graph G^B_f according to the new placement values L^j_i (similar to BG). (5) For every node v, update its potential to be $\pi(v) = \pi(v) - d(v)$, where $d(v)$ is the weight of the shortest path between x and v in $G^B_{f_L}$.

The optimality of G-BG is shown in the following corollary:

Corollary 7.7. *G-BG returns the optimal solution L that solves the placement problem (SPP-4).*

7.9.2 Complexity of G-BG

The bipartite-like graph contains $|V^B| = \theta(m + k)$ nodes and $|E^B| = \theta(m \cdot k)$ edges, where m is the number of resource types and k is the number of regions. The initial step of G-BG takes linear time; setting the zero flow values, as well as finding the initial node potentials (using topological ordering), takes $O(|V^B| + |E^B|) = O(mk)$. In each iteration of G-BG, we use Dijkstra's algorithm, which takes $O(|E^B| + |V^B| \cdot \log |V^B|) = O(mk + (m + k) \log(m + k))$. The time complexity of Steps (2)-(5) is linear by the size of the bipartite-like graph ($O(|V^B| + |E^B|) = O(mk)$), and thus each iteration of G-BG takes $O(mk + (m + k) \log(m + k))$

Suppose that the optimal solution with a minimum number of resources contains $|L_{opt}|$ resources. G-BG stops after $|L_{opt}|$ iterations. Thus, the time complexity of G-BG is $O(|L_{opt}|(mk + (m + k) \log(m + k)))$.

7.9.3 G-BG is faster than BG

The placement problem presented in this chapter is a generalization of the placement problem SPP-3, which was presented earlier in Chapter 6. Thus, G-BG can be used to solve SPP-3. Our generalized solution (G-BG) is faster than the one presented in Chapter 6, the Bipartite Graph (BG) algorithm, as shown by the following claim:

Claim 7.8. *The Generalized-Bipartite Graph (G-BG) algorithm solves SPP-3 faster than the Bipartite Graph (BG) algorithm.*

For example, suppose that the number of resource types is equal to the number of regions, i.e., $m = k$. Also, suppose that the optimal placement L_{opt} uses exactly s resource, where s is the capacity bound (i.e., $|L_{opt}| = s$). Then the time complexity of G-BG is $O(s \cdot m^2)$, while the time complexity of BG is $O(s \cdot m^3)$. This means that G-BG runs faster than BG by a factor of $O(m)$.

Proof of Claim 7.8. The correctness of the claim is based on the following facts: 1) The optimal solution for SPP-3 contains s resources at most, where s is the total storage value, i.e., $s \geq |L_{opt}|$. 2) We use the node potentials, which allows G-BG to compute the shortest paths, using Dijkstra's algorithm (which takes $O(|E| + |V| \cdot \log|V|) = O(mk + (m+k)\log(m+k))$ over a graph $G = (V, E)$) as opposed to the slower Bellman-Ford algorithm (which takes $O(|E| \cdot |V|) = O(mk(m+k))$). □

Remark 7.5. We assume that computing the marginal-differential functions values $\Delta g_i(n)$ for every integer n takes $O(1)$ time. Otherwise, if it takes $O(e)$ time to compute these values, then the complexity of the algorithm is $O(|L_{opt}|(mke + (m+k)(\log(m+k))))$.

7.10 Generalizing the Placement Problem (SPP-4) and Other Applications of G-BG

We can generalize our placement problem (SPP-4), and show that it is a special case of a mathematical problem called the **Discrete Semi-separable Concave (DiSCo)** problem, as presented below:

THE DISCRETE SEMI-SEPARABLE CONVEX (DISCO) PROBLEM

Input: Discrete and concave functions $h_i(x), h_i^j(x), h^j(x)$ for $1 \leq j \leq k$, $1 \leq i \leq m$.

Problem: Find the placement L_{opt} that maximizes the **semi-separable** profit function that is defined as follows:

$$P(L) = \sum_{j=1}^{k} \sum_{i=1}^{m} h_i^j(n) + \sum_{j=1}^{k} h^j(n) + \sum_{i=1}^{m} h_i(n). \tag{7.12}$$

The G-BG algorithm, which its correctness relies only on the monotonicity of the marginal-differential functions Δg, can be used to solve the DiSCo problem. That means that the correctness of G-BG algorithm holds if we replace the marginal-differential functions Δg by the differential of the corresponding concave functions Δh. Thus, we can run the G-BG algorithm over the bipartite-like graph (corresponding to the differential function Δh), and it will solve the DiSCo problem.

Remark 7.6. The DiSCo problem is related to the problem of finding a discrete vector $L = (L_1, L_2, \ldots, L_n)$ that maximizes the **separable** profit $P(L) = \sum_{i=1}^{m} g_i(L_i)$, where g_i are convex functions. A solution for this simple problem is to find the local maximum values of every g_i function, using $L_i^{opt} = \arg\max_{n \in \mathbb{N}} g_i(n)$. While there are many works on minimizing separable functions in the literature [75, 76], we have not yet found works on minimizing a semi-separable convex function.

7.11 Sublinear algorithms- Future work

The complexity of the G-BG shown in this chapter is linear by the number of resources $|L_{opt}|$ of the optimal placement. In most practical applications the operator must place the resources in the different regions (See Remark 1.2), as well as reading (and/or producing) the demand distribution thus leading to an algorithm that must be at least linear by the number of resources (i.e., $\Omega(|L_{opt}|)$). Nonetheless, we conjecture that one can implement an algorithm which is sublinear in the number of resources, i.e., polynomial in the thin input.

One possible way to implement a faster algorithm is by using the *Capacity-Scaling* [70] algorithm, which solves the min-cost flow problem in logarithmic time by the flow value, i.e., $\log |f|$. Using such an algorithm for solving our min-cost flow might yield a better algorithm with logarithmic run time in the number of resources, i.e., $\log |L_{opt}|$.

Chapter 8

Fundamental Sensitivity Properties of Resource Reposition

In this chapter, we start dealing with a dynamically changing (stochastic) demands. A naive strategy to adopt in case of dynamically changing conditions (demand) is to allocate an optimal placement in response to any change. However, this strategy might be expensive in the number of repositions, and it might be infeasible to implement due to limitations of operating new servers or turning them off. To this end, in this chapter, we analyze the sensitivity of optimal placements to changes in the (stochastic) demand. We establish two properties that are fundamental to the problem of resource placements and reposition. Our first theorem proves that optimal placements are quite sensitive, and even a small fluctuation in the demand can lead to a large change in the placement and therefore to a large reposition cost. The second theorem states that despite the fact that the placement is radically changed, the **profit** of optimal placements is not sensitive. The results reported in Sections 8.1-8.4 are published in [1], and the results present in Section 8.5 have not has not yet been published.

8.1 Preliminaries

We denote by $D = \{D_i^j\}$ a demand and by $L = \{L_i^j\}$ a placement, where D_i^j and L_i^j are, respectively, the demand and the placement of type i resources in region j. The profit of a placement L with respect to demand D in a multi-type system is equal to:

$$P(L,D) = \sum_{i=1}^{m}\sum_{j=1}^{k}\zeta_i^j(L_i^j, D_i^j) + \sum_{i=1}^{m}\zeta_i(L_i, D_i) + \sum_{j=1}^{k}\zeta^j(L^j), \tag{8.1}$$

where,

$$\zeta_i^j(L_i^j, D_i^j) = R_i^j \cdot E_{D_i^j}[\min(L_i^j \cdot B_i, D_i^j)] - C_i^j(L_i^j), \tag{8.2}$$

$$\zeta_i(L_i, D_i) = R_i \cdot E_{D_i}[\min(L_i \cdot B_i, D_i)] - C_i(L_i), \tag{8.3}$$

$$\zeta^j(L^j) = -C^j(L^j), \tag{8.4}$$

are the marginal functions.

We define the **reposition cost** between L and L' as the number of resources that have to be shifted from placement L to placement L', namely the number of repositions used between the placements, i.e., $\sum_{j=1}^{k}\sum_{i=1}^{m}|L_i^j - L'^j_i|$.

In this chapter, we assume that the demand changes over time, and we denote by $D(t)$ and $L(t)$ the demand and the placement at period t, respectively.

To simplify the technicalities of the analysis specifically in this chapter, we do not allow placements to contain infinite number of resources, and bound the size of a placement L by a large enough **total storage value** (abbreviated as **storage constant**) s. That means, every placement should include up to $\sum_{j=0}^{k} L^j \leq s$ resources.

8.2 L_1-c.d.f Distance

To establish the properties of sensitivity, we use the following definitions:

Definition 8.1. 1. Let N_1, N_2 be two discrete non-negative distributions that are defined over the same support set $\{0, 1, 2\ldots\}$.

(a) The L_1**-c.d.f distance** is defined as the L_1-distance between the c.d.f vectors of N_1 and N_2 i.e.,

$$d(N_1, N_2) = \sum_{n=0}^{\infty} |\Pr(N_1 \leq n) - \Pr(N_2 \leq n)|. \tag{8.5}$$

(b) The distributions N_1, N_2 are said to be ϵ**-near** to each other if the L_1-c.d.f distance between N_1 and N_2 is less than ϵ, i.e., $d(N_1, N_2) < \epsilon$.

2. The multi-dimensional demands $D = \{D_i^j\}$, $D' = \{D'^j_i\}$ are called **strongly** ϵ**-near** if the following conditions hold: 1) There is a region j_0 and a resource type

i_0 such that the demand distributions $D_{i_0}^{j_0}$ and $D'^{j_0}_{i_0}$ are ϵ-near to each other, i.e., $d(D_{i_0}^{j_0}, D'^{j_0}_{i_0}) < \epsilon$. 2) For all $(i,j) \neq (i_0, j_0)$ the demand distributions for type i resources in region j in both demand matrices are identical, i.e., $\Pr(D_i^j = n) = \Pr(D'^j_i = n)$ for every n.

3. The **revenue** of the system (as opposed to profit), is defined according to Eqs. (8.1)-(8.4) from Section 8.1 as $R(L,D) = \sum_i R_i E_{D_i}[\min(D_i, L_i)] + \sum_j \sum_i R_i^j E_{D_i^j}[\min(D_i^j, L_i^j)]$. Given a demand D, an **optimal unconstrained *revenue* placement** L is defined as the placement that maximizes $R(L,D)$. Note that the optimal revenue placement differs by definition from the optimal unconstrained placement (defined earlier), which maximizes the profit i.e., the revenue $R(L,D)$ minus the cost. The omission of the cost function is for simplicity of presentation and is done only in this section.

8.2.1 Properties of L_1-c.d.f distance

First we compare the ϵ-near distance to the well-known Klomogorov-Smirnov (KS) test [77]. We show that strongly-ϵ near is tighter than the KS test, in the sense that if two distributions are strongly ϵ-near then they also obey the KS test.

Claim 8.1. *Let N_1 and N_2 be two discrete distributions. If they are strongly-ϵ near then they obey the KS test.*

Proof of Claim 8.1. The KS test for distributions N_1 and N_2 is defined as $KS = \sup_n |\Pr(N_1 \leq n) - \Pr(N_2 \leq n)|$. Thus, by definition if N_1 and N_2 are ϵ–near than $KS < \epsilon$. □

Below we establish an interesting property of the L_1-c.d.f distance that relates the c.d.f-distance to the difference of expected values in some particular cases.

Claim 8.2. *Let N_1 and N_2 be two discrete non-negative distributions. If N_1 dominates N_2 then $d(N_1, N_2) = E(N_1) - E(N_2)$.*

Remark 8.1. Formally, distribution N_1 dominates N_2 if $\Pr(N_1 \geq n) \geq \Pr(N_2 \geq n)$ for every value $n \geq 1$.

Proof of Claim 8.2. In our proof, we use the fact that the expectation of a non-negative positive distribution is equal to:

$$E(X) = \sum_{k=1}^{\infty} \Pr(X \geq k) \tag{8.6}$$

If N_1 dominates N_2 then

$$d(N_1, N_2) = \sum_{n=0}^{\infty} |\Pr(N_1 \leq n) - \Pr(N_2 \leq n)|$$
$$\underbrace{=}_{1-\Pr(D_i \leq n)=\Pr(D_i \geq n+1)} \sum_{n=1}^{\infty} |\Pr(N_1 \geq n) - \Pr(N_2 \geq n)| =$$
$$\underbrace{=}_{\text{Definition of dominance}} \sum_{n=1}^{\infty} \Pr(N_1 \geq n) - \sum_{n=1}^{\infty} \Pr(N_2 \geq n)$$
$$\underbrace{=}_{\text{Eq. (8.6)}} E(N_1) - E(N_2). \quad (8.7)$$

□

The following claim states some particular cases where Claim 8.2 holds:

Claim 8.3. *The following holds: 1) If $N_1 \sim Poiss(\lambda_1), N_2 \sim Poiss(\lambda_2)$, where $\lambda_1 > \lambda_2$ then N_1 dominates N_2. 2) If $N_1 \sim Bin(n, p_1), N_2 \sim Bin(n, p_2)$, where $p_1 > p_2$ then N_1 dominates N_2.*

These results were proven in [78].

8.3 Sensitivity of Repositions to Demand Fluctuations

The first property, stated in Theorem 8.4, establishes that optimal unconstrained revenue placements can be quite sensitive (in fact, infinitely sensitive) to changes in the demand. Thus, an algorithmic framework that insists on finding optimal placement in response to any demand change may result in infinitely many resource repositionings and efforts.

Theorem 8.4 (Sensitivity in placement repositions). *For every system with revenue constants R_i, R_i^j optimal unconstrained **revenue** placements are sensitive to the demand D. This means that for every $\epsilon > 0$ there exists two multi-dimensional demands D, D' with optimal unconstrained revenue placements $L = \{L_i^j\}, L' = \{L'^j_i\}$ such that: 1) D', D are strongly ϵ-near. 2) The number of repositions used between L and L' is equal to the storage constant s, which can be infinitely large.*

Proof of Theorem 8.4. We set demand D to the instance where no request is made, i.e., $\Pr(D_i^j = 0) = 1$ for every resource type i and region j. An optimal revenue placement for D is the zero placement $L_i^j = 0$ for every resource type i and region j.

The demand D' is defined as follows: 1) The probability that the demand of type 1 in region 1 is more than n is $\frac{\epsilon}{2^{n+2}}$ (i.e., $\Pr(D'^1_1 \geq n) = \frac{\epsilon}{2^{n+2}}$). Therefore, the L_1-c.d.f distance between D^1_1 and D'^1_1 is

$$d(D'^1_1, D^1_1) \underbrace{=}_{Eq.\ (8.5)} \sum_{n=0}^{\infty} |\Pr(D'^1_1 \geq n) - \Pr(D^1_1 \geq n)| = \sum_{n=0}^{\infty} \frac{\epsilon}{2^{n+2}} \underbrace{=}_{\frac{1}{4}+\frac{1}{16}+\frac{1}{32}+\ldots=\frac{1}{2}} \frac{\epsilon}{2} < \epsilon.$$

2) The number of requests for type i resources in region j, where $(i,j) \neq (1,1)$, is 0 (i.e., $\Pr(D'^j_i = 0) = 1$). The L_1-c.d.f distance between D^j_i and D'^j_i is zero. Thus, the first part of the theorem holds, i.e., the demands D and D' are strongly ϵ-near.

To prove the second part of the theorem, we observe that an optimal revenue placement L' for D' should contain as many as possible type 1 resources in region 1. Since the size of each placement is bounded by the storage constant s, then L' should contain s resources of type 1 in region 1. Thus, the number of repositions used between the optimal revenue placement L and L' is equal to the storage constant s. □

The theorem implies that optimal unconstrained **profit** placements (solving SPP-4) are sensitive to the demand distribution if the cost function C is small enough and in particular if it equals zero:

Corollary 8.5. *For every system with revenue constants R_i, R^j_i there exists a cost function $C()$ where the optimal unconstrained **profit** placements (which maximize the profit) are sensitive to the demand D. This means that for every $\epsilon > 0$ there exists two multi-dimensional demands D, D' with optimal unconstrained placements $L = \{L^j_i\}, L' = \{L'^j_i\}$ such that: 1) D', D are strongly ϵ-near. 2) The number of repositions used between L and L' is equal to the storage constant s, which can be infinitely large.*

8.4 Sensitivity of Profit to Demand Distribution Fluctuations

The second result establishes that the profit of the optimal unconstrained placement is *not sensitive* to changes in the demand. So, while the placement can change radically (in response to small demand changes), its profit will not. The change depends only on the linear combination between the change of the demand and the revenue constants in Eqs. (8.2) and (8.3), regardless of the cost. We use the following definitions to formulate the theorem:

Definition 8.2. Let $D = \{D^j_i\}$, $D' = (D'^j_i)$ be two multi-dimensional demands.

1. The **demand-distance** between D and D', denoted as $d(D, D')$ is

$$d(D, D') = \sum_{i=1}^{m} \sum_{j=1}^{k} d(D_i^j, {D'}_i^j) R_i^j + \sum_{i=1}^{m} d(D_i, {D'}_i) R_i, \tag{8.8}$$

   where R_i^j and R_i are the revenue constants in Eqs. (8.2) and (8.3).

2. Demands sets D and D' are called **weakly ϵ-near** if the distance between them is less than ϵ, i.e., if $d(D, D') < \epsilon$.

3. Placement L is called an **ϵ-optimal placement** with respect to demand D if the profit of L is not less than the profit of an unconstrained (profit) optimal placement (the solution for SPP-4) of D minus ϵ. That is, if $P(L, D) \geq \max_{L'} P(L', D) - \epsilon$.

The second sensitivity theorem is presented next:

Theorem 8.6 (Sensitivity in placement profit). *Let ϵ be the threshold parameter, and let $D(t)$ and $D(t+1)$ be two multi-dimensional demands of periods $t, t+1$. Suppose $L(t)$ is an unconstrained optimal placement of $D(t)$ ($L(t) = \arg\max_L P(L, D(t))$). If $D(t)$ and $D(t+1)$ are weakly ϵ-near, then $L(t)$ is an ϵ-optimal placement with respect to demand $D(t+1)$.*

Remark 8.2. Using simple inequalities on the profits of the optimal placements on $D(t)$ and on $D(t+1)$ it is possible to provide a short proof that $L(t)$ is a 3ϵ-optimal placement with respect to demand $D(t+1)$. Below we provide a proof of the theorem (for ϵ-optimal) that is based on a reduction to a graph flow problem. The reduction and the results derived below will be needed later.

Proof of Theorem 8.6. The proof uses a mapping of the unconstrained reposition problem (SPP-4) to the min-cost flow problem similar to the one presented in Chapter 7. To clarify our examples and presentation, instead of considering the eight-layer graph G^8, we use a four-layer multigraph (a graph with parallel edges between vertices) G^4. The four-layer graph G^4 represents the problem equivalently (but more compactly) to the eight-layer graph G^8. The multigraph G^4 is composed of a source x, region nodes $j_1, j_2, \ldots j_k$, resource type nodes $i_1, i_2 \ldots i_m$ and a sink y. Between every two nodes of successive layers in G^4 are s edges, where s is the storage constant, which is an as large as needed constant (see Section 8.1). In addition, the graph contains s edges from x to y. In every edge we can transfer 1 unit of flow or not transfer flow at all, namely, the edge capacity is 1. The flow value $|f|$ is equal to s.

We construct the mapping by defining the weights $w(e)$, such that, roughly speaking, the weight of the n^{th} edge connecting a region node (say a) to a type node (say β)

represents the effect of adding the n^{th} resource of type β in region a to the corresponding marginal profit function ζ^a_β. This is equal to the marginal-differential functions $-\Delta g^a_\beta(n) = -(1) \cdot (\zeta^a_\beta(n, D) - \zeta^a_\beta(n-1, D))$. The marginal-differential functions can be expressed by the model parameters as seen in Eqs. (7.7)-(7.9) in Chapter 7. The edge weights are multiplied by minus one as we convert a maximum profit placement problem into a minimum cost flow problem. Similarly, edges from the source node x to a region node a are assigned weights corresponding to adding a resource in region a, and edges from a resource type node β to the sink node y are assigned weights corresponding to adding a resource of type β. Edges between the source x and the sink y have zero weight. They serve to preserve the flow value constant without providing any benefit to pass flow in them. Note that the graph edge weights depend on the demand D.

In this multigraph, every placement L is represented by a certain flow f_L, so that the quantities of a placement ($L = \{L^j_i\}$) are the flow values between region nodes j and resource type nodes i, i.e., the number of edges with 1 unit of flow from j to i. Similarly, the flow between x and a region node j represents the number of resources allocated in region j, and the flow between resource type node i and sink y represents the number of resources allocated of resource type i. The flow on edges between x and y represents resources (out of s) that are not utilized.

A formal representation of the min-cost flow reduction can be found at Appendix f.

Example 8.1. Consider a system with two regions denoted by a, b and two resource types denoted by α, β. Suppose that the number of resources in a of respective resource types α, β are $L^a_\alpha = L^a_\beta = 1$, while the number of resources in b are $L^b_\alpha = 0, L^b_\beta = 1$. The flow of the associated placement f_L is presented in Figure 8.1, where the red edges are edges with 1 unit of flow, and the blue edges are with 0 units of flow. If there are n red edges between every two nodes, then there are $s - n$ blue edges. In the diagram, the total flow is $s = 3$ (for example, there are three blue edges between x and y).

Note that Figure 8.1 presents the flow f_L in the G^4 multigraph. Alternatively, we can depict the flow over the eight-layer graph G^8, as we present in Figure 8.2. However, we think that depicting the flow over G^4 is clearer.

By the following claims we establish how the profit of a placement relates to the cost of its corresponding flow:

Claim 8.7. *Let L be a placement. Then its profit is equal to a constant c minus the weight of its corresponding flow f_L, i.e.,*

$$P(L, D) = c - W(f_L, w(D)), \tag{8.9}$$

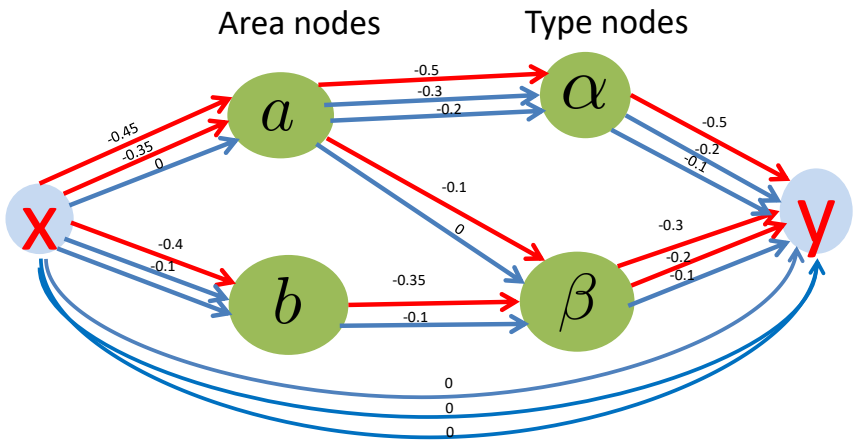


FIGURE 8.1: The flow f_L in the multigraph G^4 where $L^a_\alpha = L^a_\beta = 1$ and $L^b_\alpha = 0, L^b_\beta = 1$ (Example 8.1). Red edges are edges with 1 unit of flow, and blue edges with 0 units of flow. Between every two nodes there are a total of $s(= 3)$ edges. We omit some blue edges from the graph for the sake of presentation.

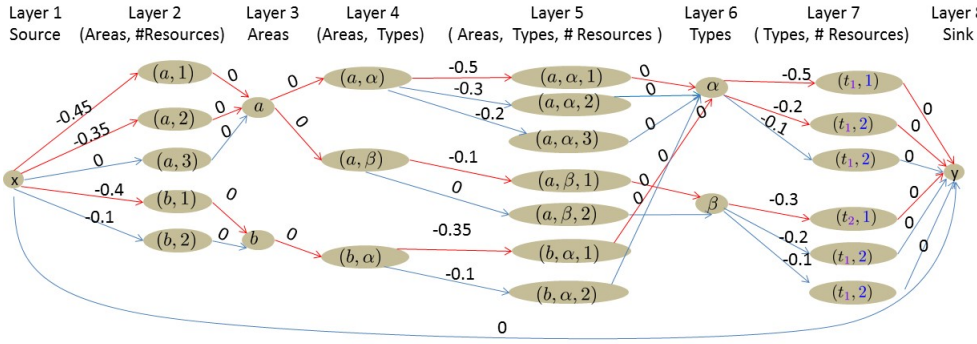


FIGURE 8.2: The flow f_L in the graph G^8. Depicting the flow in eight-layer graphs are more complex than depicting the flow in four-layer graphs (see Figure 8.1).

where c depends neither on the placement L nor the demand D, and $W(f, w(D))$ is the flow weight of the flow f using the edge weights w resulting from the demand D.

This means, for example, that the profit of placement L of Figure 8.1 is $c + 3.15$.

Thus, the solution for the unconstrained optimal placement L_{opt} corresponds to a min-cost flow f_{min}. This result is formulated in the following claim:

Claim 8.8. *There exists a placement L whose corresponding flow f_L is the min-cost flow of G^4 with edge weights $w(D)$ corresponding to demand D and with required flow $|f| = s$. Moreover, L is the optimal placement (i.e., solves SPP-4) for demand D.*

The correctness of Claims 8.7, 8.8 are proven in Appendix f. We will use these claims later in Chapters 10.4, and 10.3.

Now we can start to prove Theorem 8.6. We next observe the min-cost flows $f_{opt}(t)$ and $f_{opt}(t+1)$, which respectively are corresponding to the unconstrained optimal placements in periods $t, t+1$ i.e., $L(t)$ and $L(t+1)$. Let $w(t)$ and $w(t+1)$ denote, respectively, the graph weights of demands $D(t), D(t+1)$ in G^4 (and G^8). According to Eq. (8.9), the change in the flow weight between $f_{opt}(t)$ and $f_{opt}(t+1)$ with respect to weight $w(t+1)$ is equal to the profit deviation between placements $L(t+1)$ and $L(t)$ with respect to demand $D(t+1)$, i.e.,

$$P(L(t+1), D(t+1)) - P(L(t), D(t+1)) = W(f_{opt}(t), w(t+1)) - W(f_{opt}(t+1), w(t+1)).$$

Thus, if we prove that the flow weight difference $W(f_{opt}(t), w(t+1)) - W(f_{opt}(t+1), w(t+1))$ is smaller than the demand distance $d(D(t), D(t+1))$, which is assumed to be smaller than ϵ, then we can conclude that $P(L(t), D(t+1)) > P(L(t+1), D(t+1)) - \epsilon$, i.e., $L(t)$ is an ϵ-optimal placement with respect to demand $D(t+1)$, as required. Therefore, to prove Theorem 8.6 we only need to show the following equation:

$$W(f_{opt}(t), w(t+1)) - W(f_{opt}(t+1), w(t+1)) \leq d(D(t), D(t+1)). \tag{8.10}$$

To restrict the difference $W(f_{opt}(t), w(t+1)) - W(f_{opt}(t+1), w(t+1))$ we present and prove a new graph-theoretic result called the **Flow Sensitivity Theorem**. The theorem, which we proved in [22], is as follows:

Theorem 8.9 (Flow Sensitivity Theorem). *Let $G = (V, E)$ be a multigraph with unit capacities defined over G edges ($c(e) = 1$). Let w and w' be the sets of edge weights defined over G, and let f_{min} and f'_{min} denote, respectively, their integer min-cost flows of the same flow value ($|f_{min}| = |f'_{min}|$)[1]. Then the flow weight of f_{min} with respect to the new weight w' is bounded by the flow weight of f'_{min} with respect to w', plus the sum over all edges in G of the edge-weight deviations $|w'(e) - w(e)|$. That means*

$$W(f_{min}, w') - W(f'_{min}, w') \leq \sum_{e \in E} |w'(e) - w(e)|. \tag{8.11}$$

For the sake of clarity, we will postpone presenting the correctness of the theorem to Section 10.3.8.

[1] As stated above, the flow values of the min-cost flows $f_{opt}(t)$ and $f_{opt}(t+1)$ are equal to s, i.e., $|f_{min}| = |f'_{min}| = s$. The flow value s is the storage constant, which is an as large as needed constant.

Now, given that $w = w(t)$ and $w' = w(t+1)$ are, respectively, the edge weights of G^4 corresponding to demands $D(t), D(t+1)$, we can bound the distance $|w'(e) - w(e)|$ for every edge e.

Claim 8.10. *Let $w = w(t)$ and $w' = w(t+1)$ be the edge weights of G^4 with respect to demands $D(t)$ and $D(t+1)$ respectively. Then the difference of weights $|w(e) - w'(e)|$ is bounded by the demand distance $d(D(t), D(t+1))$, i.e.,*

$$\sum_{e \in E} |w'(e) - w'(e)| \leq d(D(t), D(t+1)). \tag{8.12}$$

Proof of Claim 8.10. Below we provide the main arguments of the proof. The full detailed proof is given in Appendix f.

We first express the edge weights $w(e)$ by the model parameters, using the expressions as seen in Eqs. (7.7)-(7.9) in Chapter 7.

Second, given a region node (say a) and a type node (say β), we show that we can bound the sum of weight differences over all edges by

$$\sum_{e \text{ connects } a \text{ to } \beta} |w(e) - w'(e)| \leq d(D^a_\beta(t), D^a_\beta(t+1))R^a_\beta, \tag{8.13}$$

$$\sum_{e \text{ connects } \beta \text{ to } y} |w(e) - w'(e)| \leq d(D_\beta(t), D_\beta(t+1))R_\beta, \tag{8.14}$$

$$\sum_{e \text{ connects } x \text{ to } a} |w(e) - w'(e)| = \sum_{e \text{ connects } x \text{ to } y} |w(e) - w'(e)| = 0. \tag{8.15}$$

The first two inequalities hold as the L_1-c.d.f distance between two distributions $d(N_1, N_2)$ is, by definition, the sum of the absolute value differences between N_1 and N_2 c.d.f values. Note that the weights between the sink x and region nodes a are not affected by the demand and therefore $w(e) = w'(e)$ for every edge connecting x to a.

Therefore we conclude that

$$\sum_{e \in E} |w(e) - w'(e)| = \leq \underbrace{\sum_{i=1}^{m} \sum_{j=1}^{k} d(D^j_i(t), D^j_i(t+1))R^j_i}_{\text{Eq. (8.13)}} + \underbrace{\sum_{i=1}^{m} d(D_i(t), D_i(t+1))R_i}_{\text{Eq. (8.14)}} = d(D(t), D(t+1)). \tag{8.16}$$

as required. □

Finally, from Claim 8.10 and Theorem 8.9 we derive Eq. (8.10) and consequently Theorem 8.6 is proven.

□

From Theorem 8.6 the following is implied:

Corollary 8.11. *For any parameter setting, for any $\epsilon > 0$ and for every demand distribution D with an optimal placement L there exists demand distribution $D' \neq D$ such that L is an ϵ-optimal placement with respect to demand D'.*

Proof of Corollary 8.11. One can create D' from D by adding an $\epsilon/2$ mass at some position k to the p.d.f of D and subtracting it at position $k+1$. Following Theorem 8.6 the corollary holds. □

8.5 Proof of the Flow Sensitivity Theorem (Theorem 8.9)

Theorem 8.9. *Let $G = (V, E)$ be a multigraph with unit capacities defined over G edges ($c(e) = 1$). Let w and w' be the sets of edge weights defined over G, and let f_{min} and f'_{min} denote, respectively, their integer min-cost flows of the same flow value ($|f_{min}| = |f'_{min}|$). Then the flow weight of f_{min} with respect to the new weight w' is bounded by the flow weight of f'_{min} with respect to w', plus the sum over all edges in G of the edge-weight deviations $|w'(e) - w(e)|$. That means*

$$W(f_{min}, w') - W(f'_{min}, w') \leq \sum_{e \in E} |w'(e) - w(e)|. \tag{8.17}$$

Proof of Theorem 8.9. Below we generalize Theorem 8.9 to any bounded minimization problem. Formally, we use the following settings: Given two vectors $x = (x_1, x_2 \ldots, x_n) \in R^n, y = (y_1, y_2, \ldots, y_n) \in R^n$, we denote $x \geq y$ iff $x_i \geq y_i$ for every coordinate i. Given a set $X \subset R^n$ of vectors, an **upper bound** $u \in R^n$ is a vector such that for every $x \in X$ we have $x \leq u$. Similarly, a **lower bound** $l \in R^n$ is a vector such that for every $x \in X$ we have $x \geq l$. The goal of the **bounded minimization problem** is to find the optimal solution in the set $x \in X$ that minimizes the dot product $x \cdot w = \sum_i x_i w_i$, where $w \in R^n$ is a given vector called the **weight (cost) vector**.

The min-cost flow problem is, in particular, a bounded minimization problem. We can represent every flow by a vector $x \in R^{|E|}$ such that x_i is the flow in edge i. The set X represents the set of all feasible min-cost flows. A lower bound for X is the zero vector $(0, 0, \ldots, 0)$ and an upper bound is the capacity function c (in case of unit capacities, it is equal to $(1, 1, \ldots, 1)$). The weight of a flow f with respect to a weight function w, i.e., $W(f, w)$, is equal to the dot production $w \cdot f$.

Thus, the following lemma generalizes Theorem 8.9 and it is sufficient to show its correctness to prove Theorem 8.9.

Lemma 8.12. *Let X be a set of feasible solutions with an upper bound $u = u(X)$ and a lower bound $l = l(X)$. Let x and x' be the optimal solutions of the bounded minimization problem with respect to weight vectors w, w'. Then the following equation holds:*

$$x \cdot w' - x' \cdot w' \leq (u - l) \cdot |w - w'|. \tag{8.18}$$

Proof. First, we show that $(x - x') \cdot w' \leq (x - x') \cdot (w - w')$. By definition of x we derive that

$$x \cdot w \leq x' \cdot w,$$

and therefore,

$$(x - x') \cdot w \leq 0.$$

Thus,

$$(x - x') \cdot w' \leq (x - x') \cdot w' - (x - x') \cdot w = (x - x') \cdot (w - w').$$

Taking the absolute value in both sides implies

$$|(x - x') \cdot w'| \leq |(x - x') \cdot (w - w')|.$$

The left side is equal to $x \cdot w' - x' \cdot w'$ (x' is the optimal solution with respect to w'). The right side is bounded by $|x - x'| \cdot |w - w'|$, according to the triangle inequality (i.e., for every two vectors $a, b \in R^n$ we have $|a \cdot b| = |\sum_{i=1}^{n} a_i \cdot b_i| \underbrace{\leq}_{\text{Triangle inequality}} \sum_{i=1}^{n} |a_i \cdot b_i| = |a| \cdot |b|$).

This implies that

$$x \cdot w' - x' \cdot w' \leq |x - x'| \cdot |w - w'|.$$

The vector $|x - x'|$ is bounded by $(u - l)$. Thus, $|x - x'| \cdot |w - w'| \leq (u - l) \cdot |w - w'|$, and the theorem follows.

□

□

Chapter 9

The Constrained-Reposition Placement Problem – Single-Type (ConRePP-1)

In this chapter, we solve the Constrained-Reposition Placement Problem (ConRePP-1) for a single-type system ($m = 1$). We provide a greedy optimal solution for the problem, called the **Unary and Move (U&Me)** algorithm that solves the problem. The work is based on a submitted paper [18] (with variation on the analysis approach).

9.1 Problem Formulation

Remark 9.1. For the sake of presentation, the notations used in this chapter are slightly different from the notations used in the rest of the thesis. For example, L_i represents the number of resources in **region** i (not of resource type i).

We denote by $D = (D_1, D_2, \ldots, D_k)$ a demand and by $L = (L_1, L_2, \ldots, L_k)$ a placement, where D_j and L_j denote the local demand and locally allocated resources in region j, respectively. We denote the total number of resources in L and the number of requests in D by respectively $|L| = \sum_{i=1}^{k} L_i$ and $|D| = \sum_{i=1}^{k} D_i$. The profit of a placement L with respect to demand D in a single-type system is equal to:

$$P(L, D) = \sum_{i=1}^{m} \zeta_i^{local}(L_i, D_i) + \zeta^{global}(|L|, |D|), \tag{9.1}$$

where,

$$\zeta_i^{local}(L_i, D_i) = R_i^{local} \cdot E_{D_i}[\min(L_i \cdot B, D_i)] - C_i^{local}(L_i), \tag{9.2}$$

$$\zeta^{global}(|L|, |D|) = R^{global} \cdot E_{|D|}[\min(|L| \cdot B, |D|)] - C^{global}(|L|), \tag{9.3}$$

are the marginal profit functions.

The problem addressed in this chapter poses the repositioning challenge, where the demand is a random variable that changes over time. A natural motivating example for this challenge is the familiar difference between stochastic demands during day or night. A naive approach for optimal response to the dynamic demand is to achieve an optimal placement in response to any change.However, this strategy might be expensive in the number of repositions, and it might be infeasible to implement due to limitations of operating new servers or turning them off. Thus, the optimal placement problem becomes a *Repositioning Problem*: given initial placement $L(t)$ at period t what is the optimal placement that can be achieved under a constraint of allowing up-to r operations (each operation is a single addition or subtraction of a resource). The problem is formulated as follows:

Constrained-Reposition Placement Problem – Single-Type (ConRePP-1)	
Assumptions:	Single-type system.
Input:	A demand $D(t+1) = \{D_j(t+1) \mid j = 1, 2, \ldots, k\}$ of period $t+1$, placement $L(t)$ of period t, a reposition bound r.
Problem:	Find the optimal placement of period $t+1$, $L(t+1) = \{L_i(t+1)\}$, that maximizes the profit $P(L(t+1), D(t+1))$ in Eq. (9.1), under the reposition constraint: $\sum_{j=1}^{k} \lvert L_j(t+1) - L_j(t) \rvert \leq r$.

9.2 Expressing the Profit Function by its Marginal-Differential Functions

Using a similar transformation done in Section 7.2 we can show that the profit can be expressed by the monotonically non-increasing marginal-differential Δg functions as follows:

$$P(L, D) = P(L) = c + \sum_{i=1}^{k} \sum_{n=1}^{L_i} \Delta g_i^{local}(n) + \sum_{n=1}^{|L|} \Delta g^{global}(n). \tag{9.4}$$

where $c = \sum_{i=1}^{k} g_i^{local}(0) + g^{global}(0)$ is a constant that does not depend on the placement L.

Remark 9.2. According to Eqs. (9.2)-(9.3), and Theorem 3.1 we can compute the marginal-differential functions by the model parameters, which are equal to

$$\Delta g_i^{local}(n) = R_i^{local} \cdot \sum_{k=(n-1)\cdot B+1}^{n\cdot B} \Pr(D_i(t+1) \geq k) - C_i^{global}(n) + C_i^{global}(n-1),$$

$$\Delta g^{global}(n) = R^{global} \cdot \sum_{k=(n-1)\cdot B+1}^{n\cdot B} \Pr(|D(t+1)| \geq k) - C^{global}(n) + C^{global}(n-1).$$

9.3 Single Repositioning Approach May Mislead

One may be tempted to consider a simplistic approach for trying to solve the repositioning problem – that of iteratively trying to add or remove single resources (single repositions). As we demonstrate next, this approach may not work. We demonstrate this by illustrating a non-intuitive situation where no single addition or subtraction is attractive in any region, yet a combination of such operations **is** attractive.

Figure 9.1 depicts a two-region example of the model setting. Specifically, the example illustrates a placement $L(t)$ where every bucket represents a region. There are two resources at the first region $L_1 = 2$, and a single resource at the second region $L_2 = 1$. The white blocks at each upper bucket represent the resources of L in that region. The 3 white blocks at the bottom bucket represent the total number $|L|$ of included resources, which is $1 + 2 = 3$. Black blocks represent the fact that resources may be added.

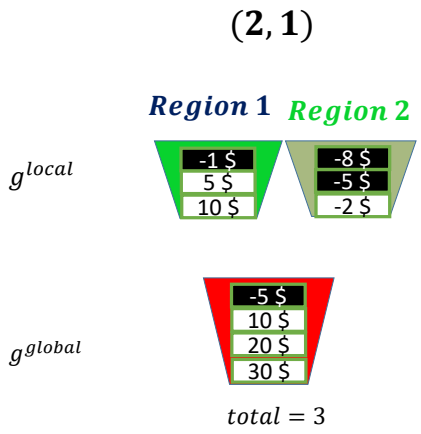


FIGURE 9.1: Resource Repositioning.

Each block at the upper buckets in the diagram is marked by local marginal-differential function $\Delta g_i^{local}(n)$, attributed to the corresponding resource. In Figure 9.1, for example, the local marginal-differential of Region 1 are $\Delta g_1^{local}(1) = 10\$, \Delta g_1^{local}(2) = 5\$, \Delta g_1^{local}(3) = -1\$$ and the local marginal-differential of Region 2 are $\Delta g_2^{local}(1) = -2\$, \Delta g_2^{local}(2) = -5\$, \Delta g_2^{local}(3) = -8\$$. The value of each block at the bottom bucket corresponds to the global marginal-differential function, which are equal to $\Delta g^{global}(1) = 30\$, \Delta g^{global}(2) = 20\$, \Delta g^{global}(3) = 10\$, \Delta g^{global}(4) = -5\$$. The marginal profits are decreasing in all the buckets, as the marginal-differential functions are concave.

The profit of every placement is computed as the sum of the values on the white boxes with the constant c. In particular, the profit of L equals $P(L) = 73 + c\$$. Suppose one adds a single resource to Region 1. The local bucket of the Region 1 will contain three resources, i.e., the local marginal function is increased by the value of the third white box in the bucket, i.e., $\Delta g_1^{local}(3) = -1\$$. The global bucket will contain four resources, i.e., the system contains number four resources from all regions, and the global marginal function is increased by $g^{global}(4) = -5\$$. In total, this action increases the profit by $\Delta g_1^{local}(3) + \Delta g^{global}(4) = -6\$$, and this addition will not increase the profit.

In a similar way, we can see that adding a resource in Region 2 decreases the profit by $-5-5 = -10\$$. Subtracting a resource from Region 2 removes the first white block from Region 2 bucket, and the third white block from the global bucket. Thus, this action increases the profit by $-\Delta g_2^{local}(1) - \Delta g^{global}(3) = 2 - 10 = -8\$$ and it is not profitable. Subtracting a resource from Region 1, increases the profit by $5 - 10 = -15\$$.

Thus, we have shown that there is neither a profitable addition nor a profitable subtraction operation. However, the combination of addition to Region 1 and subtraction from Region 2 is profitable: the local profit in Region 1 is -1 (adding the first black block from the bottom in the upper left bucket), the local profit in Region 2 is 2 (subtracting the upper white block in the upper right bucket), and the global profit does not change (no change in the bucket at the bottom). In total, these operations increase the profit function by $-1 + 2 = 1$. Note that this is the optimal placement.

9.4 The Algorithm

A solution for the placement problem (ConRePP-1) as described in Section 9.1 is a set of up to r operations (each is a single addition or subtraction of a resource), leading to the most attractive placement that can be achieved. It is challenging, since the number of sets to consider is exponential in r. It is further challenging since the global value, which depends on the number of resources, changes as a function of the resources added/removed to and from the placement. As illustrated in the previous section, an algorithm which is greedy over the next operation is not appropriate as it may lead to non-optimal solutions.

Fortunately, the problem lends itself to an effective algorithm. The algorithm uses a special greedy approach whereby it starts by greedily applying an *unary operation* (i.e., one addition or one subtraction of a resource from any region) and ends by greedily applying a *move (binary) operation* (i.e., move a resource from one region to another). To express the essentiality of this feature, we call the algorithm *Unary and Move*, or **U&Me** in short.

To present the algorithm, we use the following formal definitions:

Definition 9.1. An **operation** o is a function that applies repositioning of resources. Given a placement L, applying the operation o over L returns a placement denoted by $o(L)$. The **profitability** of operation o over placement L is $\Delta(o, L) = P(o(L)) - P(L)$. The most profitable unary operation, denoted by $U^{opt}(L)$, is the unary operation with the highest profitability among all unary operations. Similarly, we denote by $M^{opt}(L)$, the move operation with the highest profitability among all move operations. Operation o is called **profitable** if $\Delta(o, L) > 0$.

Remark 9.3. The notation $U^{opt}(L)$ denotes a unary operation with the highest profitability. This notation has different meaning than $u(L)$, which applies an unary operation u on L and returns a placement.

A pseudo-code of the algorithm is presented by Algorithm 4.

Algorithm 4 Unary and Move (U&Me)

Input: An initial placement L, and a constraint r on the number of repositioning operations.

1. $i \leftarrow 0$.
2. $u \leftarrow U^{opt}(L)$.
3. (Begin unary phase) **while** $i < r$ and u is profitable:
   (a) $L \leftarrow u(L)$, $i \leftarrow i + 1$.
   (b) $u \leftarrow U^{opt}(L)$
4. $m \leftarrow M^{opt}(L)$.
5. (Begin move phase) **while** $i < r - 1$ and m is profitable:
   (a) $L \leftarrow m(L)$, $i \leftarrow i + 2$.
   (b) $m \leftarrow M^{opt}(L)$
6. **return** L.

Remark 9.4. If there are two or more unary/move operation with highest profitability – we can choose any of them using some tie-breaking rule.

9.4.1 Efficient implementation of U&Me

To present an efficient implementation of the U&Me algorithm, we use the following definition:

Definition 9.2 (Addition, subtraction and move operations). We denote by a_j the operation of adding a single resource to region j. Similarly, s_j denotes the operation of subtracting a resource from region j. The operation $b_{j_1 \to j_2} = a_{j_2} \circ s_{j_1}$ denotes the move (binary) operation of moving a single resource from region j_1 to region j_2.

We can compute the profitability of every unary or move operation using the following observation:

Observation 9.1. *Given a placement L, the profitability of every unary or move operation can be computed as follows:*

$$\Delta(a_j, L) = \Delta g_j^{local}(L_j + 1) + \Delta g^{global}(|L| + 1) \tag{9.5}$$

$$\Delta(s_j, L) = -\Delta g_j^{local}(L_j) - \Delta g^{global}(|L|) \tag{9.6}$$

$$\Delta(b_{j_1 \to j_2}, L) = -\Delta g_{j_1}^{local}(L_{j_1}) + \Delta g_{j_2}^{local}(L_{j_2} + 1), \quad \textit{for} \quad j_1 \neq j_2 \tag{9.7}$$

In particular, the move operation with the highest profitability is composed of the highest profitability addition and the highest profitability subtraction.

The observation follows from the definition of profitability and Eq. (9.4). A formal proof can be seen in Appendix g.

Observation 9.1 leads to an efficient implementation of our algorithm:

Claim 9.2 (Efficient Implementation of U&Me). *U&Me can be implemented efficiently and have time complexity of $O((r+k)\cdot\log k)$, where r is the reposition bound and k is the number of regions.*

Remark 9.5. We assume that calculating the marginal-differential values $\Delta g(n)$ for every integer n takes $O(1)$. Otherwise, if it takes $O(e)$ time to compute these values, then the complexity of the algorithm is $O((r+k)\cdot(e+\log k))$.

Proof. To implement U&Me, we hold the total number of resources $|L|$ and in addition we hold two maximum priority queues using binary heaps. One heap contains the local marginal-differential values for addition, $\Delta g_j^{local}(L_j+1)$, and the other contains the local marginal-differential values for subtraction, $-\Delta g_j^{local}(L_j)$. Both heaps contain k elements, where k is the number of regions. Initially, the values of these heaps are corresponding to the initial placement $L = L_{init}$. To construct both heaps, the algorithm inserts k marginal-differential values to each heap, and each insertion takes $O(\log k)$ time. Thus, the time complexity of inserting these marginal-differential values takes $O(k\log k)$ time.

Using these binary heaps, we can find in each iteration the operation with the highest profitability $U^{opt}(L)$ or $M^{opt}(L)$ in $O(1)$ time. After applying a unary operation in region j_0, only two corresponding values in both heaps $\Delta g_{j_0}^{local}(L_{j_0}+1)$ and $-\Delta g_{j_0}^{local}(L_{j_0})$ need to be updated. This can be done in $O(\log k)$ time using a maximum priority queue heap. Other variables of the algorithm, such as the total number of resources $|L|$ and the number of j_0 resources L_{j_0} are updated in $O(1)$. Similarly, after a move operation, we update the binary heaps and set the other variables of the algorithm in $O(\log k)$ time. Thus, every iteration in the unary phase takes $O(\log k)$ time.

The number of iterations is bounded by the reposition bound r, and thus the time complexity of U&Me is $O((r+k)\log k)$.

□

9.4.2 Algorithm Optimality

We observe that a valid repositioning of resources can be represented by a multiset[1] of r unary operations at most, where r is the reposition bound. For example, the multiset

[1] A multiset is a generalization of the concept of a set that, unlike a set, allows for multiple instances of the multiset's elements and the order of the elements does not matter. For example, $\{a,a,b\}$ and $\{a,b\}$ are different multisets, while $\{a,b,b\}$ and $\{b,a,b\}$ represent the same multiset. We can define naturally set operations such as union, intersection, set difference and others. For a formal definition on multiset operations, we refer to [79].

$G = \{a_1, a_1, s_2\}$ represents the repositioning of adding two resources to region 1 and subtracting a single resource from region 2.

The optimality of the U&Me algorithm is presented in the following theorem:

Theorem 9.3 (Optimality theorem). *Let G be a multiset of r unary operations at most. Let L^G be the placement derived by conducting G operations over the initial placement L_{init}. Then the profit of the U&Me placement is higher than or equal to the placement of L^G, i.e., $P(L_{U\&Me}) \geq P(L^G)$.*

The proof of the optimality theorem is complex and involved. It requires three different threads, where the first and second threads are used to prove the third thread. These threads are based on the U&Me sequence $C_{U\&Me} = (o^1, o^2, \ldots, o^n)$, where o^i is the operation used in the i^{th} iteration of U&Me[2]. At first we establish in Appendix h that the U&Me sequence $C_{U\&Me}$ satisfies the **non-canceling** property. That means, if the U&Me algorithm adds a resource to region j_1, then it will not subtract from that region. Formally, this lemma is as follows:

Lemma 9.4. *Let L_{init} be the initial placement, and let $C_{U\&Me}$ be the U&Me sequence over L_{init}. Then $C_{U\&Me}$ is non-canceling.*

The formal proof of Lemma 9.4 is in Appendix h. Roughly speaking, it is proven by way of contradiction. Assume the contrary, i.e., that the U&Me algorithm adds a resource to region j_{org} and afterwards subtracts from that region. We show that there must exists another region j_{alt} such that the profitability of adding a resource to j_{alt} is strictly greater than the profitability of adding a resource to j_{org}. This contradicts the mechanism of the U&Me algorithm, where in each iteration it adds to the region with the highest addition profitability; in particular, the profitability of adding a resource to j_{alt} cannot be strictly greater than the profitability of adding a resource to j_{org}.

Let $G_{U\&Me}$ be the multiset of unary operations that the U&Me algorithm uses. By Lemma 9.4 we conclude that $G_{U\&Me}$ contains non-canceling operations, i.e., $G_{U\&Me}$ is non-canceling. This is a useful property for the second and the third threads, as we will explain later.

The second thread, presented in Section 9.6, is regrading non-canceling multisets, and in particular $G_{U\&Me}$. We present an algorithm called the **Generalized U&Me algorithm** that runs as follows: Given a repositioning multiset G of non-canceling operations, it returns a sequence C_G that satisfies the following properties: 1) The sequence C_G repositions the resources exactly as G does. 2) The sequence C_G resembles the

[2]Every move operation $b_{j_1 \to j_2} = s_{j_1} \circ a_{j_2}$ in the U&Me sequence is represented by a single operation in the sequence, and **not** by their corresponding addition and subtraction operations.

properties of the U&Me sequence, and thus it is called a **U&Me-like sequence**. The properties of a U&Me-like sequence are presented in Section 9.6.3, and are based on Lemma 9.4.

In the third thread, we define a **potential function** that for each repositioning multiset G it associates a triple $\pi(G) = (\pi_1(G), \pi_2(G), \pi_3(G))$ of non-negative integers. The lower the potential, with respect to the lexicography order, then the more G "resembles" the U&Me multiset $G_{U\&Me}$. We show that the potential of a multiset G is zero, i.e., $\pi(G) = (0, 0, 0)$, if and only if G is the U&Me multiset, i.e., $G = G_{U\&Me}$.

Our major lemma shows that for every multiset G there exists another multiset G' that has better performance and smaller potential, as formulated next:

Theorem 9.5. *Let G be a multiset of r unary operations at most with non-zero potential (i.e., $\pi(G) \neq (0, 0, 0)$). There exists another multiset G' of r unary operations at most that satisfies the following properties: 1) The profit of the placement corresponding to G' is not smaller than the profit of placement corresponding to G, i.e., $P(L^G) \leq P(L^{G'})$. 2) The potential of G' is strictly smaller than the potential of G, i.e., $\pi(G) >_L \pi(G')$, with respect to the lexicography order $>_L$.*

The proof of Theorem 9.5 is divided into two stages: First, we take care of several special cases where it is easy to construct a multiset G' that satisfies conditions 1) and 2). For example, one special case is where G is a multiset that contains canceling operations. Let G' be the multiset G where canceling operations are removed. The placement corresponding to G' is the placement corresponding to G; thus, the profits of these placements are equal to each other, i.e., $P(L^G) = P(L^{G'})$ and the first condition is satisfied. In addition, to derive that the potential of G' is strictly smaller than the potential of G, we use Lemma 9.4; Lemma 9.4 implies that the U&Me multiset $G_{U\&Me}$ is non-canceling, and thus it "resembles" more the non-canceling multiset G' than the canceling multiset G.

After presenting the special cases, we continue to our major case, where G is non-canceling, and thus we can derive its U&Me-like sequence C_G. We show that in this case we can replace one unary operation in G by another unary U&Me operation in $G_{U\&Me}$. Doing so will increase of the profit of the corresponding placement and decrease the potential. To define these operations, we compare the U&Me-like sequence of G, i.e., C_G, with the U&Me sequence $C_{U\&Me}$. Based on the properties of the U&Me-like sequences (Section 9.6.3), we show which operation in G to replace by and which operation in $G_{U\&Me}$ to keep.

From Theorem 9.5, we can conclude that for every multiset G, there is a finite chain of multisets $G = G_0, G_1, G_2 \ldots, G_n$ that satisfies the following properties: 1) The profit

of the placement corresponding to G_i is not larger than the profit of the placement corresponding to G_{i+1}, i.e., $P(L^{G_i}) \geq P(L^{G_{i+1}})$. 2) The potential of G_i is strictly larger than the potential of G_{i+1}, i.e., $\pi(G_i) >_L \pi(G_{i+1})$. 3) The last multiset in the chain G_n has zero potential, and thus it is the U&Me multiset, i.e., $G_n = G_{U\&Me}$. This shows that the profit of the placement corresponding to G is, at most, the profit of the U&Me placement, i.e., $P(L_{U\&Me}) = P(L^{G_n}) \geq P(L^{G_0}) = P(L^G)$.

Thus, the optimality of the U&Me algorithm (Theorem 9.3) follows.

9.5 Preliminaries

To prove the optimality of the U&Me algorithm, we use the definitions and properties of repositioned placement, relative profitability, and the composite operation.

9.5.1 Definitions

Definition 9.3 (Composite operation). Let o_1, o_2 be two operations. Then, the **composite operation** of o_1, o_2, denoted by $o_1 \circ o_2$, is a an operation such that for every placement L we have $(o_1 \circ o_2)(L) = o_1(o_2(L))$.

For example, suppose $L = (1, 2, 3)$ contains i resources in region i. Then the operation $a_1 \circ a_2$ adds a resource to regions 1 and 2, and therefore $(a_1 \circ a_2)(L) = (2, 3, 3)$. Note that the operation composition is similar to function composition.

Definition 9.4 (Repositioned placement and relative profitability). Given a sequence C, the repositioned placement, denoted by $R(k, C)$, is the placement derived from conducting the first k operations of $C = (o^1, o^2, \ldots)$ on the initial placement L_{init}, i.e., $R(k, C) = (o^1 \circ o^2 \circ \ldots o^k)(L_{init})$. The **relative profitability** of operation o^k is the profitability of o^k over the repositioned placement $R(k-1, C)$, and is denoted by $Rel(k, C) = \Delta(o^k, R(k-1, C))$. Finally, for the sake of presentation we respectively denote the repositioned placement and the relative profitability corresponding to the U&Me sequence $C_{U\&Me}$ by $R(k) = R(k, C_{U\&Me})$ and $Rel(k) = Rel(k, C_{U\&Me})$.

For example, suppose $L_{init} = (1, 2, 3)$ contains i resources in region i. Let $C_{U\&Me} = (a_1, s_2, b_{3\to1})$ be the U&Me sequence. Applying the first operation of $C_{U\&Me}$ (adding a single resource in region 1) on L_{init} results in a placement $R(1) = (2, 2, 3)$. Conducting the first two operations of $C_{U\&Me}$ on L_{init} is equal to $R(2) = (2, 1, 3)$. Finally, conducting all operations of $C_{U\&Me}$ on L_{init} is $R(3) = (3, 1, 2)$, which is the U&Me placement (i.e., $R(3) = L_{U\&Me} = (3, 1, 2)$).

9.5.2 Properties of repositioned placements, relative placements and composite operations

We next establish simple properties of repositioned placements, relative placements, and composite operations.

Observation 9.6. *Suppose o^k is the k^{th} operation that the U&Me algorithm selects. Then o^k is profitable over the repositioned placement $R(k-1)$, i.e., o^k has positive relative profitability ($Rel(k) > 0$).*

This observation follows immediately since the U&Me algorithm selects a profitable operation in each iteration.

Observation 9.7. *Let L be a placement, o_1, o_2 two operations. Then the profitability of composite operation $o_1 \circ o_2$ over L is equal to:*

$$\Delta(o_1 \circ o_2, L) = \Delta(o_2, L) + \Delta(o_1, o_2(L)). \tag{9.8}$$

In particular, we derive that if o_1 is profitable over L and o_2 is profitable over $o_1(L)$, then $o_1 \circ o_2$ is profitable over L.

This observation follows immediately from the definition of profitability, i.e., $\Delta(o, L) = P(o(L)) - P(L)$.

Observation 9.8. *Let $C = (o^1, o^2, \ldots, o^n)$ be a sequence. Then the profit of the repositioned placement $R(k, C)$ is equal to*

$$P(R(k, C)) = P(C) + \sum_{i=1}^{k} Rel(i). \tag{9.9}$$

Moreover, if L^C is the placement corresponding to C (i.e., derived from conducting all operations of C on L), then

$$P(L^C) = P(L) + \sum_{i=1}^{n} Rel(i). \tag{9.10}$$

This observation follows immediately from the definition of the relative profitability and Observation 9.7.

9.6 The Generalized U&Me Algorithm and U&Me-like Sequences

In this section, we present the **Generalized U&Me algorithm**, devised solely for the sake of the optimality proof. Given a multiset G of non-canceling operations the Generalized U&Me algorithm returns a **U&Me-like sequence** of G, denoted by C_G. We show that the U&Me-like sequence satisfies properties which are vital to the optimality of the U&Me algorithm.

9.6.1 Definitions and preliminaries

To show the properties of the Generalized U&Me algorithm, we use the following definitions:

Definition 9.5 (The corresponding unary multiset). Given a sequence $C = (o^1, o^2, \ldots, o^n)$ its **corresponding unary multiset** $mult(C)$ is the multiset of unary operations in C.

For example, if $C = (a_1, s_2, b_{2\to 3})$ then its corresponding unary multiset is $mult(C) = \{a_1, s_2, s_2, a_3\}$. By definition, the U&Me multiset $G_{U\&Me}$ is the corresponding unary multiset of the U&Me sequence $C_{U\&Me}$. Note that there can be multiple sequences with the same corresponding unary multiset. For example, the corresponding unary multiset of $C = (a_1, s_2)$ and $C' = (s_2, a_1)$ are equal to each other, i.e., $mult(C) = mult(C') = \{a_1, s_2\}$.

Definition 9.6 (The most profitable unary and move operations in a multiset). Given a placement L and a multiset G of unary operations, we define the **most profitable unary operation in** G $U(L, G)$ as the unary operation with the highest profitability in G, i.e., $U(L, G) = \arg\max_{u \in G} \Delta(u, L)$. In a similar way, the **most profitable move operation in** G $M(L, G)$ is the move operation with the highest profitability composed of a subtraction and an addition operation in G. We define $U(L, G) = \emptyset$ if $G = \emptyset$ and $M^{opt}(L, G) = \emptyset$ if $G = \emptyset$ or G contains either only subtraction operations or only addition operations (i.e., we cannot compose a move operation from G).

This definition is an extension of the most profitable unary and move operations (Definition 9.1). Suppose we denote G_{all} to be the set of all available unary operations, i.e., for every region j the operations a_j and s_j appears infinitely in G_{all}. Then, the most profitable unary and move operations, as defined in Definition 9.1, are corresponding to G_{all}, i.e., $U(L) = U(L, G_{all}), M(L) = M(L, G_{all})$.

Using Observation 9.1, we can derive that the most profitable move operation in G is composed of the highest profitability addition and the highest profitability subtraction in G:

Observation 9.9. *Let a_{j_1} and s_{j_2} respectively denote the most profitable subtraction and addition operation in G. Then the most profitable move operation in G is $b_{j_2 \to j_1} = a_{j_1} \circ s_{j_2}$.*

Definition 9.7. Given two multisets G_1, G_2 of unary operations, the **set difference** or the **difference** of the multisets, denoted by $G_1 \setminus G_2$, is the multiset G_1 where the elements of G_2 are removed.

For example, if $G_1 = \{a_3, a_1, a_1, s_3\}$ $G_2 = \{a_1, a_1, s_2, s_3\}$ then difference of G and $G_{U\&Me}$ is $G \setminus G_{U\&Me} = \{a_3\}$.

9.6.2 The Generalized U&Me algorithm

Given a multiset G of non-canceling unary operations, the Generalized U&Me algorithm finds a sequence C_G that satisfies the following properties: 1) The corresponding multiset of C_G is G. 2) C_G shares similar properties with the U&Me sequence $C_{U\&Me}$, and thus it is called a **U&Me-like sequence**. The Generalized U&Me-like (Algorithm 5) resembles the U&Me algorithm, by selecting the highest profitability operation in G in each iteration. The algorithm, as opposed to U&Me, comprises three phases: In the first phase, similarly to the U&Me algorithm, the Generalized U&Me algorithm selects the highest profitability unary operations in G, one by one, and removes them from G. The first phase ends when there remains no profitable unary operation. In the second phase, the Generalized U&Me algorithm greedily selects the most profitable move operations from G (by constructing each move from one addition and one subtraction). As opposed to the U&Me algorithm, the Generalized U&Me algorithm continues selecting move operations even if they are not profitable. The second phase ends in the case where G contains either only subtraction operation or only addition operations (i.e., the algorithm cannot compose a move operation from G). Finally, in the third phase, the algorithm greedily selects unary operations from G.

Example 9.1. We present a running example of the Generalized U&Me algorithm over the multiset $G = \{a_1, s_2, s_2, a_3, a_3, a_3, a_3\}$, i.e., the repositioning that adds one resource to Region 1, subtracts two resources from Region 2 and adds four resources to Region 4. That means, after conducting the operations of G over the initial placement L_{init}, the derived placement will be $L^G = (3, 0, 4)$. The marginal-differential functions Δg is depicted in Figure 9.2.

Algorithm 5 The Generalized Unary and Move (Generalized U&Me) algorithm

Input: The initial placement L_{init}, a multiset of non-canceling operations G.
Output: A U&Me-like sequence $C_G = (o_G^1, o_G^2, \ldots, o_G^k)$ corresponding to G.

1. $L \leftarrow L^{init}$, $i = 1$.
2. $u \leftarrow U(L, G)$.
3. (Begin unary phase) **while** $u \neq \emptyset$ and u is profitable:
   (a) $o_G^i \leftarrow u$.
   (b) $L \leftarrow u(L)$.
   (c) $G \leftarrow G \setminus \{u\}$.
   (d) $u \leftarrow U(L, G)$.
   (e) $i \leftarrow i + 1$.
4. $m \leftarrow M(L, G)$
5. (Begin move phase) **while** $m \neq \emptyset$:
   (a) $o_G^i \leftarrow m$.
   (b) $L \leftarrow m(L)$.
   (c) $G \leftarrow G \setminus \{s_{j_1}, a_{j_2}\}$, given that $m = b_{j_1 \to j_2}$.
   (d) $m \leftarrow M(L, G)$.
   (e) $i \leftarrow i + 1$.
6. $u \leftarrow U(L, G)$
7. (Begin unary phase) **while** $u \neq \emptyset$:
   (a) $o_G^i \leftarrow u$.
   (b) $L \leftarrow u(L)$.
   (c) $G \leftarrow G \setminus \{u\}$.
   (d) $u \leftarrow U(L, G)$.
   (e) $i \leftarrow i + 1$
8. **return** C_G.

Let G_i and L_i respectively denote the multiset G and the placement L in the i^{th} iteration, and let $C_G = (o_G^1, o_G^2, \ldots)$ be the U&Me-like sequence corresponding to G. In the first iteration, the profitability of the unary operations of $G_0 = G$ are $\Delta(a_1, L_0) = 8 - 1 = 7, \Delta(s_2, L_0) = -2 - 6 = -8$ and $\Delta(a_3, L_0) = -1 - 1 = -2$. Thus, the unary operation with the highest profitability, i.e., the first operation in C_G is $o_G^1 = a_1$, and thus we continue with the algorithm.

In the second iteration, the multiset and the placement are $G_1 = \{s_2, s_2, a_3, a_3, a_3, a_3\}$ and $L_1 = (3, 2, 0)$. There is no profitable unary operation in G_1 over L_1 ($\Delta(s_2, L_1) = -2 + 1 = -1$ and $\Delta(a_3, L_1) = -1 - 20 = -21$). Thus, the Generalized U&Me algorithm terminates the first phase, and it runs the second phase, i.e., the move phase. The move operation $b_{2\to3} = s_2 \circ a_3$ is the single move operation that can be conducted by G_1. Thus, the algorithm selects the second operation to be $o_G^2 = b_{2\to3}$. The operations $\{s_2, a_3\}$ are removed from the unary multiset G, and therefore $G_2 = G_1 \setminus \{s_2, a_3\} = \{s_2, a_3, a_3, a_3\}$. In a similar way, the third operation in the sequence should be $o_G^3 = b_{2\to3}$, and the unary multiset includes only addition operations, i.e., $G_3 = \{a_3, a_3\}$. Thus, the algorithm terminates the second phase, and it runs the third phase, i.e., the unary phase. The algorithm sets the forth and fifth operations to $o_G^4 = o_G^5 = a_3$.

We show that the U&Me-like sequence of G is $C_G = (a_1, b_{2\to3}, b_{2\to3}, a_3, a_3)$. Note that the last operations in the U&Me-like sequence, i.e., $o^3_G = b_{2\to3}, o^4_G = a_3$ and $o^5_G = a_3$ have non-positive relative profitability, i.e., are not profitable. This property, i.e., the last operations in the sequence are not profitable, is shown next.

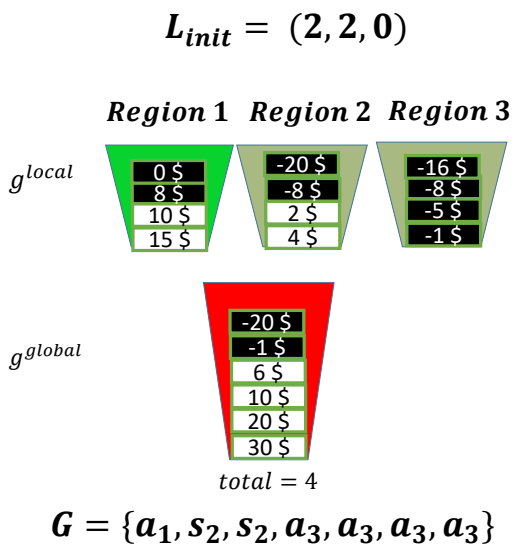


FIGURE 9.2: A running example of the Generalized U&Me algorithm over the unary multiset $G = \{a_1, s_2, s_2, a_3, a_3, a_3, a_3\}$ and the initial placement $L_{init} = (2, 2, 0)$. The white blocks in the bottom bucket represent the resource the initial placement contains. Black blocks represent the fact that resources may be added.

9.6.3 Properties of the U&Me-like sequence

The U&Me-like sequence is a useful mechanism by which we prove Theorem 9.5. The first observation, which follows immediately from the definition of the Generalized U&Me algorithm, shows that the every U&Me-like sequences is a **greedily ordered sequence** presented as followed:

Observation 9.10. *[U&Me-like sequences are greedily ordered sequence] Let $C_G = (o^1_G, o^2_G, \ldots, o^n_G)$ be a U&Me-like sequence of multiset G. Then C_G is a **greedily ordered sequence**, i.e., every operation is either the most profitable unary or the most profitable move in the sequence. Formally that means that either $o^k = U(L_{k-1}, G_{k-1})$ or $o^k = M(L_{k-1}, G_{k-1})$, where $L_{k-1} = R(k-1, C)$ is the repositioned placement of conducting the first $k-1$ operations of C over the initial placement L_{init}, and $G_{k-1} = G \setminus mult(o^1_G, o^2_G, \ldots, o^{k-1}_G) = mult(o^k, o^{k+1}, \ldots, o^n)$ are a unary operation multiset that was not used in the first $k-1$ operations.*

Next, we present two vital properties to derive which operations in the U&Me-like sequences are profitable and have positive relative profitability.

In the first claim, we show that the relative profitability in a U&Me-like sequence is decreasing, given that it contains operations that are all either addition, subtraction or move operations. This is formulated as follows (proven in in Appendix g):

Claim 9.11 (Monotonicity Property). *Suppose $C_G = (o^1, \ldots, o^n)$ is a U&Me-like sequence. Suppose that both operations o^i and o^{i+1} are either addition, subtraction or move operations. Then the i^{th} relative profitability is not smaller than that of $i+1^{th}$, i.e., $Rel(i, C) \geq Rel(i+1, C)$.*

In the second claim, we show that if a multiset of non-canceling unary operations G has no profitable unary operation over a placement L, then G does not contain any profitable unary operations after conducting a move operation over L. This is formulated as follows:

Claim 9.12. *Let L be a placement, and G a multiset of non-canceling unary operations. Suppose that G does not contain any profitable unary operations on L. Let m be a move operation, composed of a subtraction and an addition operation contained in G. Let $m(L)$ be the placement derived by conducting m over L. Then G does not contain any profitable unary operations on $m(L)$.*

The proofs of both claims are presented in Appendix g, and are straightforward and technical (similar to the claims of Section 9.5.2).

Based on these claims, we can easily show which operations in the U&Me-like sequences are profitable, as follows:

Claim 9.13 (Last operations in U&Me-like sequences are not profitable). *Suppose $C_G = (o^1, o^2, \ldots, o^n)$ is a sequence returned from the Generalized Unary and Move (U&Me) algorithm over a multiset G. Then, there exist indices $1 \geq i_1 \geq i_2 \geq i_3 \geq n$ such that the following holds:*

1. *$o^1, o^2, \ldots, o^{i_1}$ are all profitable (with positive relative profitability) unary operations, selected in the first (unary) phase.*
2. *$o^{i_1+1}, o^{i_1+2}, \ldots, o^{i_2}$ are all profitable move operations, selected in the second (move) phase.*
3. *$o^{i_2+1}, o^{i_2+2}, \ldots, o^{i_3}$ are all non-profitable (with non-positive relative profitability) move operations, selected in the second (move) phase.*
4. *$o^{i_3+1}, o^{i_3+2}, \ldots, o^n$ are all non-profitable unary operations, selected in the third (unary) phase.*

Moreover, during the second (move) and third (unary) phases, there is no profitable unary in G over the repositioned placement L.

Proof. We set the indices i_1, i_2, i_3 as follows: 1) The operations $o^1, o^2, \ldots, o^{i_1}$ are the unary operations selected in the first phase. 2) The operation o^{i_2+1} is the first move operation (selected in the second phase) that is non-profitable. 3) The operations $o^{i_3+1}, o^{i_3+2}, \ldots, o^n$ are the unary operations selected in the third phase. Of course, the operations $o^1, o^2, \ldots, o^{i_1}$ are profitable unary operations, and the operations $o^{i_1+1}, o^{i_1+2}, \ldots, o^{i_2}$ are profitable move operations. Since o^{i_2+1} is non-profitable, then according to Claim 9.11, the move operations $o^{i_2+1}, o^{i_2+2}, \ldots, o^{i_3}$ are non-profitable. According to the Generalized U&Me algorithm, after selecting the i_1^{th} operation, there is no profitable unary operation in G. Since $o^{i_1+1}, o^{i_1+2}, \ldots, o^{i_3}$ are all move operations, then by Claim 9.12, the unary operation o^{i_3} is not profitable. Thus, according to Claim 9.11, the unary operations $o^{i_3+1}, o^{i_2+2}, \ldots, o^n$ are non-profitable. □

Note that using Claims, 9.11, 9.13 over the U&Me multiset $G_{U\&Me}$, which is non-canceling according to Lemma 9.4, implies the following corollary:

Corollary 9.14 (Properties of the U&Me algorithm). *The following results regarding the U&Me algorithm hold:*

1. *The U&Me sequence $C_{U\&Me}$ satisfies the monotonicity property (Claim 9.11).*
2. *During the move phase of the U&Me algorithm, there is no profitable unary operation over the repositioned placement.*
3. *Suppose that the U&Me algorithm finds that there is no profitable move operation in its move phase, and it terminates. Then, the U&Me placement, $L_{U\&Me}$, solves the Static Placement Problem (namely, without reposition constraint), i.e., for every other placement L', the profit of $L_{U\&Me}$ is larger than the profit of L', i.e., $P(L') \leq P(L_{U\&Me})$.*

Proof. The first and the second parts follow from Claims, 9.11, 9.13. The third, can be proven as follows: Let placement $L' \neq L_{U\&Me}$ be a placement. There exists a multiset G of non-canceling unary operations such that conducting all operations of G over $L_{U\&Me}$ yields L'. By Claim 9.13, we derive that there is no profitable unary or move operation over $L_{U\&Me}$, and therefore all operations in the U&Me-like sequence, $C_G = (o^1_G, o^2_G, \ldots)$, are not profitable and have non-positive relative profitability ($Rel(i, C_G) \leq 0$). Since the profit of L' is equal to the profit of $L_{U\&Me}$ plus the sum of negative relative profitabilities, $P(L') = \sum_i Rel(i, C_G) + P(L_{U\&Me})$ (Observation 9.8) where $Rel(i, C_G) \leq 0$ for every i, we derive the desired result. □

9.7 The Potential Function and the Proof of Theorem 9.5

As we presented in Section 9.4.2, if we show a potential function π that satisfies the following conditions: 1) A multiset G has zero potential if and only if $G = G_{U\&Me}$. 2) Theorem 9.5, which is presented below, then the optimality of the U&Me algorithm is proven.

Theorem 9.5. *Let G be a multiset of at most r unary operations with non-zero potential (i.e., $\pi(G) \neq (0,0,0)$). There exists another multiset G' of r unary operations at most that satisfies the following properties: 1) The profit of the placement corresponding to G' is not smaller than the profit of placement corresponding to G, i.e., $P(L^G) \leq P(L^{G'})$. 2) The potential of G' is strictly smaller than the potential of G, i.e., $\pi(G) >_L \pi(G')$, with respect to the lexicography order $>_L$.*

9.7.1 The potential function

For the sake of our discussion, we define a **valid multiset** G as a multiset of r unary operations at most. We denote $\hat{G}$ to be the set of all valid multisets, and $\mathbb{Z}^+$ the set of all non-negative integers. The potential function $\pi : \hat{G} \rightarrow \mathbb{Z}^{+3}$, is composed of three potentials functions, i.e., $\pi(G) = (\pi_1(G), \pi_2(G), \pi_3(G))$.

We define the first potential function $\pi_1(G)$ as follows:

Definition 9.8 (First potential function $\pi_1(G)$). The U&Me multiset, denoted by $G_{U\&Me}$, is the multiset of unary operations the U&Me algorithm uses to reposition its resources (i.e., $G_{U\&Me} = mult(C_{U\&Me})$). For every valid multiset G, the first potential function is defined as the size of the difference between G and $G_{U\&Me}$, i.e., $\pi_1(G) = |G \setminus G_{U\&Me}|$.

For example, suppose that the U&Me algorithm adds to region 1 in the first iteration, subtracts from region 2 in the second iteration, and moves a resource from region 3 to region 1 in the third iteration. Then, the U&Me multiset is $G_{U\&Me} = \{a_1, s_2, s_3, a_1\}$. If $G = \{a_1, a_3, s_3, a_1\}$ then the difference of G and $G_{U\&Me}$ is $G \setminus G_{U\&Me} = \{a_3\}$, and thus the potential is $\pi_1(G) = 1$. Note that the U&Me multiset contains r operations at most, since the U&Me sequence (and thus its multiset) does not contain canceling operations. This means, the U&Me multiset is also a valid multiset.

While the first potential function π_1 is defined of the multisets G and $G_{U\&Me}$, and how these multisets differ from each other, the second potential function π_2 is defined over the U&Me-like sequence C_G and the U&Me sequence.

Definition 9.9 (Second potential function $\pi_2(G)$). Let $C = (o_C^1, o_C^2, \ldots, o_C^{k_C})$ and $C' = (o_{C'}^1, o_{C'}^2, \ldots, o_{C'}^{k'_C})$ be two sequences. We say that C is a **prefix** of C' if $o_C^i = o_{C'}^i$ for every $1 \leq i \leq k_C$. For example $C_1 = (a_1), C_2 = (a_1, s_2)$ are prefixes of $C_3 = (a_1, s_2, b_{3\to1})$.

Let $C_G = (o_G^1, o_G^2, \ldots)$ and $C_{U\&Me} = (o_{U\&Me}^1, o_{U\&Me}^2, \ldots)$ respectively denote the U&Me-like sequence corresponding to G and the U&Me sequence. The **common prefix sequence**, denoted by C_{com}, is the longest common prefix sequence of C_G and $C_{U\&Me}$. Formally, this means that if we denote the length of C_{com} by n, then $C_{com} = (o_G^1, o_G^2, \ldots, o_G^n) = (o_{U\&Me}^1, o_{U\&Me}^2, \ldots, o_{U\&Me}^n)$ and $o_{U\&Me}^{n+1} \neq o_{U\&Me}^{n+1}$. An operation is called a **common operation** if it is in C_{com}. The **multiset of uncommon operations of** G, denote by $uncommon(G)$ is the unary multiset of C_G operations that are uncommon with the U&Me sequence $C_{U\&Me}$. Finally, the second potential function is the number of uncommon operations in G, i.e., $\pi_2(G) = |uncommon(G)|$.

For example, suppose that the sequences of $G = \{a_1, s_3, a_1, a_1\}$ and $G_{U\&Me} = \{a_1, s_4, a_3\}$ are $C_G = (a_1, b_{3\to1}, a_1)$ and $C_{U\&Me} = (a_1, b_{4\to3})$. The common prefix sequence is $C_{com} = (a_1)$, and thus $o_{U\&Me}^1 = o_G^1 = a_1$ is a single common operation. The operations $o_G^2 = b_{3\to1} = a_1 \circ s_3$ and $o_G^3 = a_1$ of the sequence of G are not common operations. Thus, the multiset of uncommon operations in G is $uncommon(G) = \{s_3, a_1, a_1\}$, and the second potential function of G is equal to $\pi_2(G) = |uncommon(G)| = 3$.

Remark 9.1. There is a special case where we remove a single unary from the multiset $uncommon(G)$, which, for the sake of presentation, we discuss later in Remark 9.2.

The third potential function is an indicator whether $G = G_{U\&Me}$, as follows:

Definition 9.10 (Third potential function $\pi_3(G)$). The third potential function, i.e., $\pi_3(G)$, is equal to one if $G \neq G_{U\&Me}$ and otherwise is equal to 0.

Of course, by the definition of the third potential function, we derive that G has zero potential if and only if $\pi(G) = (0, 0, 0)$. Thus, it is left to prove the correctness of Theorem 9.5, in order to establish the optimality of the U&Me algorithm. In the rest of this section, we will prove Theorem 9.5.

9.7.2 Proof of Theorem 9.5 – the special cases

We say that a multiset G' is said to be **better than** a multiset G if it satisfies the following conditions: 1) The profit of its corresponding placement increases, i.e., $P(L^G) \leq P(L^{G'})$. 2) The potential strictly decreases $\pi(G) >_L \pi(G')$.

We first considered several special cases, using the properties of the U&Me-like sequences. In these cases, it is easy to show the correctness of Theorem 9.5, i.e., show a multiset G' that is better than G. These special cases are presented by the following claim:

Claim 9.15. *Let G be a valid multiset with a U&Me-like sequence C_G. In the following cases, we show a multiset G' that is better than G:*

(A) If C_G is a prefix of the U&Me sequence $C_{U\&Me}$, then $G_{U\&Me}$ is better than G.

(B) If the U&Me sequence $C_{U\&Me}$ is a prefix of C_G, then $G_{U\&Me}$ is better than G.

(C) Suppose that C_G contains non-profitable operations, and let G' be G where these non-profitable operations are removed. Then G' is better than G.

(D) Suppose G contains canceling operations, and let G' be the valid multiset G where canceling operations are removed. Then G' is better than G.

To illustrate the special cases, we present an example for each respective one, as follows: (A) If $C_G = (a_1)$ and $C_{U\&Me} = (a_1, s_2)$, i.e., C_G is a prefix of $C_{U\&Me}$, then $G_{U\&Me}$ is better than G. (B) If $C_G = (a_1, s_2, a_1)$ and $C_{U\&Me} = (a_1, s_2)$ then $G_{U\&Me}$ is better than G. (C) If the multiset $G = \{a_1, a_2, s_3, s_4\}$ and its U&Me-like sequence is $C_G = (a_1, b_{3\to 2}, s_4)$ where $b_{3\to 2}, s_4$ are both non-profitable, then $G' = \{a_1\}$ is better than G. (D) Suppose that $C_G = (a_1, s_1, s_2)$. Then a_1, s_1 cancel each other out and $G' = \{s_2\}$ is better than G.

The correctness of the first part (A) is based on the fact that the U&Me algorithm selects only profitable operations, and thus conducting further U&Me operations can only reduce the profit. The second and third parts are based on the fact that the last operations in C_G are not profitable (i.e., Claim 9.13). The forth part follows as the placements corresponding to G and G' are identical ($L^G = L^{G'}$), and the U&Me multiset $G_{U\&Me}$ is non-canceling (Lemma 9.4). Thus, for every pair of canceling operations $\{a_j, s_j\}$ in G, one of these operations is not in $G_{U\&Me}$, and thus the first potential function $\pi_1(G) = |G \setminus G_{U\&Me}|$ strictly decreases.

9.7.3 Proof of Theorem 9.5 – the major cases

Suppose now that G in not one of the special cases presented in Claim 9.15. This means that the U&Me-like sequence C_G is not a prefix of the U&Me sequence $C_{U\&Me}$, or vice versa, and C_G contains only non-canceling and profitable operations.

In the major cases, we select one uncommon unary operation $u_G \in \mathit{uncommon}(G)$, and one unary operation in the U&Me multiset $u_{U\&Me} \in G_{U\&Me}$. We define G' to be G

where u_G is replaced by $u_{U\&Me}$, i.e., $G' = (G \setminus \{a_{j_{rep}}\}) \uplus \{a_{j_{U\&Me}}\}$[3]. To identify u_G and $u_{U\&Me}$, we use the following definition of the **first uncommon operation**:

Definition 9.11 (First uncommon operation). Let $C_G = (o_G^1, o_G^2, \ldots)$ and $C_{U\&Me} = (o_{U\&Me}^1, o_{U\&Me}^2, \ldots)$ respectively denote the U&Me-like sequence corresponding to G and to the U&Me sequence. Suppose that the length of the common prefix sequence C_{com} is n, i.e., $C_{com} = (o_G^1, o_G^2, \ldots, o_G^n) = (o_{U\&Me}^1, o_{U\&Me}^2, \ldots, o_{U\&Me}^n)$ and $o_{U\&Me}^{n+1} \neq o_{U\&Me}^{n+1}$. The **first uncommon operation of** $C_{U\&Me}$ is $o_{U\&Me}^{n+1}$, i.e., the operation with the minimal index in the U&Me sequence that is not in the common prefix sequence. The **first uncommon operation of** C_G is o_G^{n+1}, i.e., the operation with the minimal index in the sequence of G that is not in the common prefix sequence.

For example, suppose that the sequences of $G = \{a_1, s_3, a_1, a_1\}$ and $G_{U\&Me} = \{a_1, s_4, a_3\}$ are $C_G = (a_1, b_{3\to1}, a_1)$ and $C_{U\&Me} = (a_1, b_{4\to3})$. The common prefix sequence is $C_{com} = (a_1)$, and thus the length of common sequence is $n = 1$. The move operation $o_G^2 = b_{3\to1}$ is the first uncommon operation of C_G, and $o_{U\&Me}^2 = b_{4\to3}$ is the first uncommon operation of $C_{U\&Me}$. Note that the first uncommon operation in both C_G and $C_{U\&Me}$ exists since C_G is not a prefix of $C_{U\&Me}$, and vice versa.

Remark 9.2. As mentioned in Remark 9.1, there is one special case where we remove a single unary operation from the multiset of uncommon operations, $uncommon(G)$, before identifying the first uncommon operation. If the first uncommon operation of $C_{U\&Me}$ and C_G is either an addition or a move operation, which both add to the same region j, then a single addition to region j is removed from $uncommon(G)$. For example, suppose that $C_G = (s_1, b_{2\to3})$ and $C_{U\&Me} = (s_1, a_3)$. The first uncommon operation of $C_{U\&Me}$ and C_G is respectively $b_{2\to3} = s_2 \circ a_3$ and a_3. These operations are both adding to region 3. Thus, an addition operation to region 3 is removed, and the multiset of unary common operations in G is equal to $uncommon(G) = \{s_2\}$. In a similar way, if the first uncommon operation of $C_{U\&Me}$ and C_G is either a subtraction or a move operation, which both remove from the same region j, then a single subtraction from region j is removed from $uncommon(G)$.

Based on the characteristics of o_G^{n+1} and $o_{U\&Me}^{n+1}$ we next define the first uncommon unary operation $u_{U\&Me}$ according to the following cases:

(A) The first uncommon operation in $C_{U\&Me}$ adds to region $j_{U\&Me}$, and the first uncommon operation in C_G does not add to that region. For example, if $C_G = (a_1, b_{3\to1}, a_1)$ and $C_{U\&Me} = (a_1, b_{4\to3})$ the first uncommon operation in $C_{U\&Me}$,

[3]The **multiset sum** $\uplus$ of two multisets A and B is a multiset $C = A \uplus B$ such that every element that appears x_A and x_B times in, respectively, A and B will appear $x_C = x_A + x_B$ in C. For example, if $A = \{1, 1, 2\}$ and $B = \{3, 4, 1\}$ then $C = \{1, 1, 2, 3, 4, 1\}$.

$o^2_{U\&Me} = b_{4\to3}$, is a move operation that adds to region $j_{U\&Me} = 3$ while the first uncommon operation of C_G does not add to that region. In this case, we define $u_{U\&Me} = a_{j_{U\&Me}}$, the addition operation to $j_{U\&Me}$.

(B) Suppose that case (A) does not hold. If the first uncommon operation in $C_{U\&Me}$ subtracts from region $j_{U\&Me}$ and the first uncommon operation in C_G does not subtract from that region, we define $u_{U\&Me} = s_{j_{U\&Me}}$.

In both cases, the unary U&Me operation $u_{U\&Me}$ is replacing an uncommon operation in G. Note that either case (A) or case (B) holds, according to the following observation:

Observation 9.16. *Either one of the following conditions holds: 1) The first uncommon operation in $C_{U\&Me}$ adds to region $j_{U\&Me}$, and the first uncommon operation in C_G does not add to that region. 2) The first uncommon operation in $C_{U\&Me}$ subtracts from region $j_{U\&Me}$ and the first uncommon operation in C_G does not subtract from that region.*

This follows as the first uncommon operation in $C_{U\&Me}$, $o^{n+1}_{U\&Me}$, differs than the first uncommon operation in C_G, i.e., $o^{n+1}_{U\&Me} \neq o^{n+1}_G$. A formal proof can be seen in Appendix g.

For the sake of presentation, we will assume without loss of generality that case (A) holds, i.e., the unary U&Me operation is an addition operation to region $j_{U\&Me}$, i.e., $u_{U\&Me} = a_{j_{U\&Me}}$. The proof for case (B) is symmetric to case (A).

We show in Appendix g that the multiset of uncommon operations of G does not contain an addition to region $j_{U\&Me}$.

Claim 9.17. *Suppose that $u_{j_{U\&Me}} = a_{j_{U\&Me}}$. Then, there is no uncommon operation in C_G that adds to region $j_{U\&Me}$, i.e., $a_{j_{U\&Me}} \notin uncommon(G)$.*

For example, if $C_G = (a_1, b_{3\to1}, a_1)$ and $C_{U\&Me} = (a_1, b_{4\to3})$ then the first uncommon operation in $C_{U\&Me}$, $o^2_{U\&Me} = b_{4\to3}$, is a move operation that adds to region $j_{U\&Me} = 3$, while the unary multiset of uncommon operation G is $\{a_1, s_3, a_1\}$, which does not contain $a_{j_{U\&Me}} = a_3$.

The correctness of Claim 9.17 is shown by way of contradiction. If the claim does not hold, then based on the greediness of the U&Me sequence, the first uncommon operation in C_G would add to region $j_{U\&Me}$, which contradicts case (A).

Based on the structure of the multiset of uncommon operations $uncommon(G)$, the following claims show how to select operation u_G, the uncommon operation for which u_G is replaced by $u_{U\&Me} = a_{j_{U\&Me}}$ to construct the better multiset G':

Claim 9.18. *Suppose that uncommon(G) contains an addition operation that adds a resource to region j_{rep}. Suppose that the first uncommon operation in the U&Me sequence adds to region $j_{U\&Me}$, while the first uncommon operation in G sequence does not. Let G' be the unary multiset G, where the addition to region j_{rep} is replaced by an addition to region $j_{U\&Me}$, i.e., $G' = (G \setminus \{a_{j_{rep}} = u_G\}) \uplus \{a_{j_{U\&Me}}\}$. Then, G' is a valid multiset that satisfies $P(L^G) \leq P(L^{G'})$.*

Claim 9.19. *Suppose that uncommon(G) contains only subtraction operations. Suppose that the first uncommon operation in the U&Me sequence adds to region $j_{U\&Me}$. Suppose that the first uncommon operation in C_G subtracts from region j_{rep}. Let G' be the unary multiset G, where the subtraction of j_{rep} is replaced by an addition of $j_{U\&Me}$, i.e., $G' = (G \setminus \{s_{j_{rep}} = u_G\}) \uplus \{a_{j_{U\&Me}}\}$. Then, G' is a valid multiset that satisfies $P(L^G) \leq P(L^{G'})$.*

Since G is a valid multiset (i.e., a multiset that contains up to r unary operations) then G' must be also a valid multiset. We can use Claim 9.18 if the uncommon multiset, *uncommon*(G), contains at least one addition operation, while Claim 9.19 is used if *uncommon*(G) contains only subtraction operations.

Below we present in examples for the use of Claim 9.18,9.18 in, respectively, Examples 9.2, 9.3, as presented below:

Example 9.2. Suppose that the sequences of multisets $G = \{a_1, s_2, a_3\}$ and $G_{U\&Me} = \{a_1, a_3, a_4\}$ are $C_G = (a_1, s_2, a_3)$ and $C_{U\&Me} = (a_1, a_4, a_3)$. The first uncommon operation in the U&Me sequence is $o^2_{U\&Me} = a_4$, i.e., adds to region $j_{U\&Me} = 4$ (i.e., $u_{j_{U\&Me}} = a_{j_{U\&Me}} = a_4$). The uncommon operations in G are $o^2_G = s_2$ and $o^2_G = a_3$ and therefore *uncommon*$(G) = \{s_2, a_3\}$. Since *uncommon*(G) contains an addition operation $u_G = a_3$, we use Claim 9.18, where the replaced addition operation is $a_{j_{rep}} = a_3$. Claim 9.18 implies that the multiset

$$G' = (G \setminus \{a_{j_{rep}}\}) \uplus \{a_{j_{U\&Me}}\} = (\{a_1, s_2, a_3\} \setminus \{a_3\}) \uplus \{a_4\} = \{a_1, s_2, a_4\} \tag{9.11}$$

satisfies $P(L^G) \leq P(L^{G'})$.

Example 9.3. Suppose that the sequence of $G = \{a_1, s_2, s_3\}$ is $C_G = (a_1, s_2, s_3)$ and the U&Me sequence is $C_{U\&Me} = (a_1, a_2)$. The multiset of uncommon unary operations in G i.e., *uncommon*$(G) = \{s_2, s_3\}$, contains only subtraction operations. The first U&Me-uncommon operation adds to region $j_{U\&Me} = 2$ ($u_{U\&Me} = a_2$) and the first uncommon operation in C_G subtracts from region $j_{rep} = 2$ ($u_g = s_2$). Thus, according to Claim 9.19

the multiset

$$G' = (G \setminus \{a_{j_{rep}}\}) \uplus \{a_{j_{U\&Me}}\} = \{a_1, a_2, s_3\}, \tag{9.12}$$

satisfies $P(L^G) \leq P(L^{G'})$.

The proof of Claim 9.18 is done by showing that the profitability of moving a resource from region j_{rep} to region $j_{U\&Me}$ is profitable over the placement L^G corresponding to G. The proof of Claim 9.19 uses a different technique: we show that replacing $s_{j_{rep}}$ by $a_{j_{U\&Me}}$ in the U&Me-like sequence C_G can only increase the relative profitability of all operations in the sequence. The proof of both claims is technical.

Next, we establish that in both cases, the potential of G is lexicographically greater than that of G':

Claim 9.20. *Suppose that uncommon(G) contains a unary operation u_G. Let G' be the unary multiset G where u is replaced by an addition of $j_{U\&Me}$, i.e., $G' = (G \setminus \{u_G\}) \uplus \{a_{j_{U\&Me}} = u_{U\&Me}\}$. Then, the potential of G is lexicographically greater than the potential of G'. More formally, we show that:*

(1) The number of operations not in the U&Me multiset cannot increase, i.e., $\pi_1(G) = |G \setminus G_{U\&Me}| \leq |G' \setminus G_{U\&Me}| = \pi_1(G')$.

(2) The number of uncommon operation is strictly reduced, i.e., $\pi_2(G) = |uncommon(G)| > |uncommon(G')| = \pi_2(G')$.

The correctness of the first part follows, as operation $a_{j_{U\&Me}}$ is a U&Me operation, and thus the replacement, $G' = (G \setminus \{u_G\}) \uplus \{a_{j_{U\&Me}}\}$, cannot decease the first potential. Note that equality happens in case the replaced operation u_G is also a U&Me operation in $G_{U\&Me}$. The second part follows, as u_G is an uncommon operation of G, and we show that according to the U&Me greediness, $a_{j_{U\&Me}}$ must be a common operation of G'.

Finally, the combination of Claims 9.20, 9.18, and 9.19 constructs the proof of Theorem 9.5.

Example 9.4. Let us take the setting in Example 9.2, i.e., the sequences of multisets $G = \{a_1, s_2, a_3\}$ and $G_{U\&Me} = \{a_1, a_3, a_4\}$ are $C_G = (a_1, s_2, a_3)$ and $C_{U\&Me} = (a_1, a_4, a_3)$. We replace the operation $u_G = a_3$ in the uncommon set, $uncommon(G) = \{s_2, a_3\}$, by the first uncommon operation in $C_{U\&Me}$, i.e., operation $a_{j_{U\&Me}} = a_4$, and derive that $G' = \{a_1, s_2, a_4\}$ (according to Eq. (9.11)). The first potential function of these multisets is equal both to 1 (i.e., $\pi_1(G) = \pi_1(G') = |\{s_2\}| = 1$). However, we will show in the next paragraph that the second potential function of G is decreasing, i.e., $\pi_2(G) = |uncommon(G)| = 2 > |uncommon(G')| = \pi_2(G') = 1$.

The multiset G contains one common operation (a_1), and two uncommon operations (s_2, a_3). Replacing a uncommon operation $u_G = a_3$ by another operation can only reduce the multiset of uncommon operations (i.e.,$|uncommon(G)| \geq |uncommon(G')|$). We claim that the new operation, i.e., $a_{j_{U\&Me}} = a_4$, must be a common operation with the U&Me sequence $C_{U\&Me} = (a_1, a_4, a_3)$ in G', i.e., $a_{j_{U\&Me}} \notin uncommon(G')$. This follows from the greediness of the U&Me algorithm: the operations a_1 and a_4 are respectively the most profitable unary operations over L_{init} and $a_1(L_{init})$. Thus, the sequence of the multiset $G' = \{a_1, a_4, s_2\}$ should be $C_{G'} = (a_1, a_4, s_2)$, a_4 is a common operation of both $C_{G'}$ and $C_{U\&Me}$, and $|uncommon(G)|$ is reduced by one, i.e., $|uncommon(G')| = \pi_2(G') = 1$.

9.8 Performance Evaluation

We use synthetic simulations along with real demand traffic, taken from [5], to numerically evaluate the algorithm and the solution. We are interested in examining how the reposition parameter r affects system performance and its efficiency, in cases of demand changes and disaster recovery.

9.8.1 Demand shift & disaster recovery

We start by evaluating the efficiency of resource reposition in response to a demand change under synthetic demand. We consider a system with $k = 2$ regions of opposite time zones (e.g., Chicago and Beijing), so while one region is subject to day-time demand $D_1 \sim N(400, 100^2)$ (i.e., Normal demand with (mean, stdvar)= (400,100)), and the other is subject to night-time demand $D_2 \sim N(25, 5^2)$.

We assume a profit function as in Eq (9.1) where the cost and revenue functions are linear and the revenue from serving a local request is 1.5 larger than that of a remote request.

We assume that an initial placement $L(0)$ has been optimized to meet the demand at time 0. Now, when time changes (12 hour shift) and the demands flip ($D_1 \Leftrightarrow D_2$) between the regions, the operator is interested in repositioning the resources, while it is constrained by the number of resources (r) it can reposition.

In Figure 9.3, we depict the relative profit of U&Me as a function of the allowed repositions (as a fraction of the number of repositions that achieve optimal placement). The profit is normalized compared to the optimal unconstrained placement. Also depicted is the plot for a three-region system, with an additional region of demand

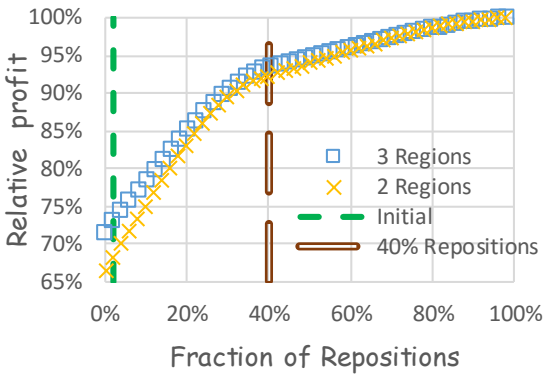


FIGURE 9.3: Effect of repositioning on profit - response to a demand shift.

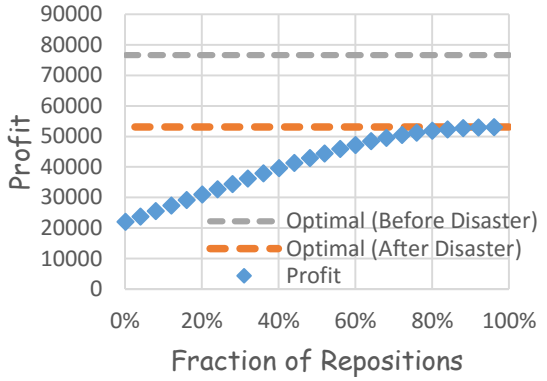


FIGURE 9.4: Effect of repositioning on profit - response to disaster.

$D_3 \sim N(100, 20^2)$. The demand in this case shifts cyclically across regions every 8 hours.

The figure demonstrates that resource repositions can increase revenue by a factor of about $\frac{1}{0.7} \approx 1.43$ (70% to 100%). Further, it is demonstrated that repositioning under constraints can be quite effective, whereby more than 90% of the profit can be achieved by repositioning only 40% of the resources.

Next (Fig 9.4) we demonstrate how repositions behave in response to a failure. We consider the previous synthetic three-region example, and assume that the resource placement was optimized for that settings. Now, the region with a data center located at the high demand region experiences a failure such that no resource in the region can be used. This causes the profit to instantly drop by almost $(80000-20000)/80000 = 75\%$ (!), and forces the optimal (reposition-unconstrained) placement not to use resources in that region. U&Me increases the profit significantly using a small number of repositions; e.g., 60% of the repositions yield 90% of the after-disaster optimal profit, increasing its initial profit $(50000/20000) = 2.5$ times.

In Figure 9.5, we examine the sensitivity of performance to the ratio between satisfying a request remotely and locally. The figure demonstrates, as one could expect, that the

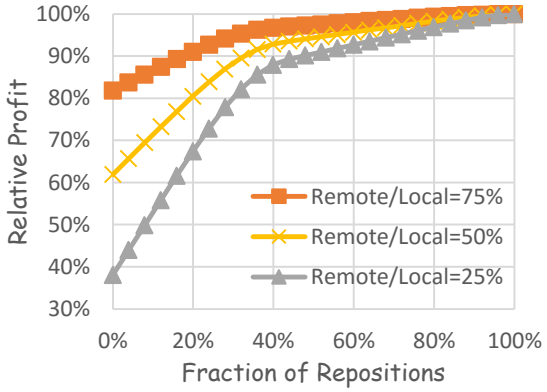


FIGURE 9.5: Sensitivity to revenue structure and effect of remote / local cost ratio.

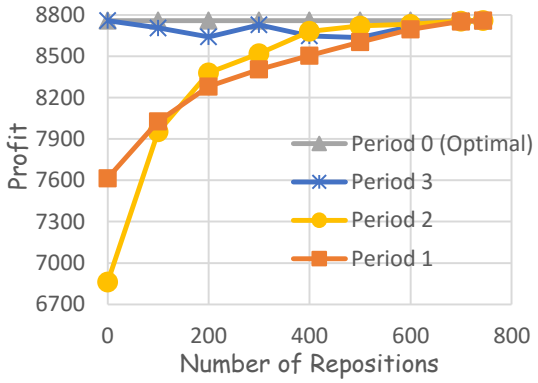


FIGURE 9.6: Follow the sun ("real" data).

profit resulting from repositions decreases with this ratio (a ratio of 1 should entail no benefit from repositioning).

9.8.2 "Follow the sun" on "real" demand

We next evaluate how U&Me operates when demand "follows the sun" and the algorithm is used over successive time periods. In every 8-hour period the demands shift cyclically across the regions and the operator repositions up to r resources between periods.

With the lack of real demand data, we use data from [5] that reflects the total demand in a *whole data center*; we expect that this data significantly underestimates the effect of the algorithm since its over-time variation should be significantly smoother than that of a real application (smoothing due to merging demands of many sources). Using that data, we consider $k = 3$ regions where in every 8 hours the demands shift cyclically over $D_1 \sim N(334, 115^2), D_2 \sim N(504, 100^2), D_3 \sim N(186, 55^2)$.

In Figure 9.6, we depict the profit as a function of the reposition parameter r for four successive time periods (spreading over one day where period 0 is the initial one and the demand at period 3 is cyclically identical to it). The figure demonstrates that using

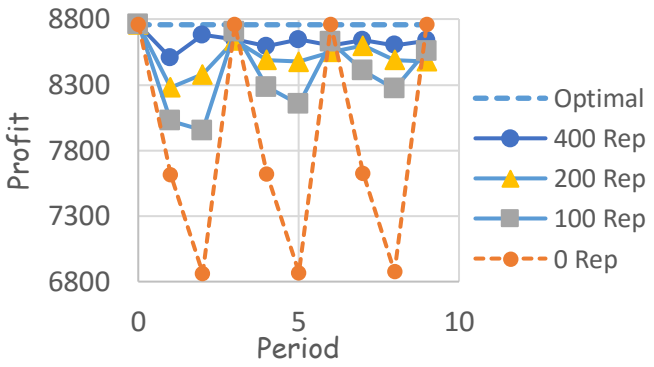


FIGURE 9.7: Sequential operation of repositions under periodic demand.

a small number of repositions yields poor performance at the first and second periods (since resource placement is not optimized); increasing r to the value of 200 drastically improves the performance at these periods, while slightly degrading that of period 3 – and thus the overall benefit is very significant. Further, increasing r to 800 yields optimal performance in all periods.

9.8.3 Sequential operation of repositions

An important question is how one should reposition resources in environments where the demand distribution continuously changes; for example, consider the case of Figure 9.6, where changes continue to occur for many periods. A very simplistic approach is to optimize the resources once, and then at every eight-hour period to conduct a reposition in response to the new demand. Figure 9.7 demonstrates the performance of such an approach on a setting that is similar to that used in Figure 9.6. The figure depicts the performance as a function of the period index (where periods 0, 1, and 2 represent the first day. Periods 3, 4, and 5 represent the second day, and so on) and as a function of the constraint on the number of repositions allowed per period. The reader may observe that when the number of repositions allowed is very small (0-100), periods 0, 3, 6, 9 benefit from optimal performance, leading periods 1, 4, 7 to suffer from sub-optimal performance (as expected). When the constraint increases, all periods converge to close to optimal performance.

The approach described above (optimize at period 0 and reposition using U&Me at all other periods) is very logical and quite effective/efficient when the future demands *are not known to the operator in advance.*

An interesting open question is how to position and reposition the resources in the event that all future demands (possibly periodic) are known to the operator. This is the subject of an ongoing research. The low complexity of U&Me allows us to utilize it as

an efficient building block in a wider framework algorithm that seeks optimality over a collection of periods.

Chapter 10

The Constrained-Reposition Placement Problem – Multi-Type (ConRePP-2)

In this chapter, we solve the general Constrained-Reposition Placement Problem (ConRePP-2) in a multi-type system that is capacity-constrained, i.e., there is a bound over the global number of resources a placement can contain. We show in this chapter that the problem is hard, and cannot be solved unless there is a solution for an old open problem in graph theory, called the **Exact Perfect Matching Problem**. Then, we provide a polynomial greedy heuristic solution called **The Shortest Cycle Operation (SCO) algorithm**, which is a generalization of the optimal solution the problem for the single-type environment(U&Me algorithm). In addition, we present another heuristic algorithm called **The Shortest Cycle Canceling (SCC) algorithm** in Section 10.3. Finally, based on fundamental sensitivity properties of resource repositioning (Chapter 8), we create the **Hybrid algorithm**. We provide simulations and analysis of these heuristic algorithms. The work in this chapter was published in [1], except the SCC algorithm (Section 10.4), which has not yet been published.

10.1 Problem formulation

We denote by $D = \{D_i^j\}$ a demand and by $L = \{L_i^j\}$ a placement, where D_i^j and L_i^j are, respectively, the demand and the placement of type i resources in region j. The profit of a placement L with respect to demand D in a single-type system is equal to:

$$P(L,D)=\sum_{i=1}^{m}\sum_{j=1}^{k}\zeta_i^j(L_i^j,D_i^j)+\sum_{i=1}^{m}\zeta_i(L_i,D_i)+\sum_{j=1}^{k}\zeta^j(L^j), \tag{10.1}$$

where,

$$\zeta_i^j(L_i^j,D_i^j)=R_i^j\cdot E_{D_i^j}[\min(L_i^j\cdot B_i,D_i^j)]-C_i^j(L_i^j), \tag{10.2}$$

$$\zeta_i(L_i,D_i)=R_i\cdot E_{D_i}[\min(L_i\cdot B_i,D_i)]-C_i(L_i), \tag{10.3}$$

$$\zeta^j(L^j)=-C^j(L^j), \tag{10.4}$$

are the marginal profit functions (the formulation is similar to Chapter 3).

The problem addressed in this chapter poses the repositioning challenge, where the demand is a random variable that changes over time. A natural motivating example for this challenge is the familiar difference between stochastic demands during day or night. A naive approach for optimal response to dynamic demand is to achieve an optimal placement in response to any change. However, this strategy might be expensive in the number of repositions, and it might be infeasible to implement due to limitations of operating new servers or turning them off. Thus, the optimal placement problem becomes a *Repositioning Problem*: given initial placement $L(t)$ at period t what is the optimal placement that can be achieved under a constraint of allowing up-to r operations (each operation is a single addition or subtraction of a resource)? The problem is formulated as follows:

Constrained-Reposition Placement Problem Multi-Type (ConRePP-2)	
Assumptions:	None.
Input:	A demand $D(t+1)=\{D_i^j(t+1)\}$ of period $t+1$, placement $L(t)=\{L_i^j(t)\}$ of period t, a reposition bound r.
Problem:	Find the optimal placement of period $t+1$, $L(t+1)=\{L_i^j(t+1)\}$ that maximizes the profit $P(L(t+1),D(t+1))$ in Eq. (10.1), under the reposition constrain: $\sum_{j=1}^{k}\sum_{i=1}^{m}\lvert L_i^j(t)-L_i^j(t+1)\rvert\leq r$.

Note that if $r=\infty$, then the problem is to find an optimal placement with respect to a demand $D(t+1)$ (and regardless of repositions)[1]. This is, eventually, the placement problem presented in Chapter 7 (SPP-4). For this chapter, we call that problem the **Unconstrained-Reposition Placement Problem**.

Remark 10.1. To simplify the technicalities of the analysis specifically in this section – we do not allow placements to contain an infinite number of resources, and bound the

[1] There might be multiple optimal placements.

size of a placement L by a large enough **total storage value** (abbreviated as **storage constant**) s. This means that every placement should include up to $\sum_{j=0}^{k} L^j \leq s$ resources. This is as opposed to the analysis presented in Chapter 7, where we assume that there is no such storage constant. Our provided algorithms do not depend on the storage constant s.

The focus of this chapter is on the Constrained-Reposition Placement Problem (ConRePP-2), which we show to be hard to solve, and for which we provide an heuristic solution. The Unconstrained-Reposition Placement Problem (SPP-4) is solved as seen in Chapter 7.

10.2 Hardness of the Constrained-Reposition Problem

In this section, we prove the hardness of the Constrained-Reposition Placement Problem (ConRePP-2); specifically, we show that if there is an optimal polynomial[2] solution for the Constrained-Reposition Problem – then there is a polynomial solution to the Exact Perfect Matching Problem[3]. The question whether the Exact Perfect Matching problem has a polynomial time solution has been considered to be a well-known open problem in graph theory for almost 30 years ([19–21]). One may, therefore, conjecture that ConRePP-2 cannot be solved in polynomial time. We show that the hardness of the ConRePP-2 remains true even if the marginal profit functions ζ_i^j (defined in Eqs. (3.6)-(3.8)) are linear functions with limitations (See Remark 10.2 in the sequel).

For the sake of proving the hardness we define the *Fair Christmas Game Problem*, and show a polynomial reduction from the Fair Christmas Game Problem to ConRePP-2. Then, in Appendix i we show that there is a polynomial reduction from the *Exact Cycle Sum Problem* to the Fair Christmas Game Problem. The Exact Cycle Sum Problem was proven to be polynomially equivalent to the Exact Perfect Matching Problem (i.e., there is a polynomial solution to one problem iff there is a polynomial solution to the other problem).

[2] Polynomial in the reposition constant r, the number of regions k, and the number resource types m. See Remark 1.3 in Chapter 1.

[3] Given a graph where edges are colored in red or blue and a parameter k, the Exact Perfect Matching Problem asks for a perfect matching that has exactly k red edges; a perfect matching is a set of edges covering all vertices such that no two edges share a vertex.

10.2.1 The Fair Christmas Game Problem and its reduction to the Constrained-Reposition Placement Problem(ConRePP-2)

The Fair Christmas Game Problem is formulated for the sake of proving that the ConRePP-2 is hard. In a Christmas Game, there are n players where each of them has n Christmas gifts which they can send to their friends. In a *fair* game every player that receives x gifts must send x gifts. We assume a player can give only a single gift to a friend, and she can give gifts only to players that she defines as her friends. We denote S_i to be the set of player-i's friends, and define $\hat{S} = \{S_1, S_2, \ldots, S_n\}$ to be the **friend list** of all players.

The Fair Christmas Game Problem is formulated as follows:

Fair Christmas Game Problem	
Input:	A parameter k, the number of players n, the friend list $\hat{S} = \{S_1, S_2, \ldots, S_n\}$ of the players.
Problem:	Is there a fair game in which a total number of k gifts are delivered between players?

We first show that the Fair Christmas Game Problem can be reduced to ConRePP-2 by showing that every instance of the former can be formulated as an instance of the latter, and thus conclude that ConRePP-2 is hard, at least as the Fair Christmas Game Problem:

Theorem 10.1. *There is a polynomial reduction from the Fair Christmas Game Problem to ConRePP-2.*

Proof of Theorem 10.1. We construct the reduction by creating an instance of ConRePP-2 consisting of n regions (region j represents player j), and n resource types (resource type i represents the gifts originated by player i, called **type** i **gifts**). Under this construction, a placement $L = \{L_i^j\}$ represents the number of type i gifts possessed by player j. We consider the placement of period t, $L(t) = \{L_i^j(t)\}$, to represent the state of the gifts prior to giving them; that is $L_i^i(t) = n$ (player i possesses n copies of gift i) and $L_i^j(t) = 0$ for $i \neq j$ (no one has yet been given a gift). The placement $L(t+1)$, resulting from the reposition, corresponds to the states of the gifts after they are given. This means that if $L_i^j(t+1) = 1$ for $i \neq j$ then player i sends to player j a gift.

Given the friend-list $\hat{S} = \{S_1, S_2, \ldots, S_n\}$ of a Fair Christmas Game instance, a placement of gifts L is called $\hat{S}$**-valid** if for every $i \neq j$, the number of type i gifts player j possesses is either zero or one (i.e., $L_i^j \in \{0, 1\}$) and in case player j possesses a single gift from player i, then j is a friend of i according to $\hat{S}$ (i.e., if $L_i^j = 1$ then $j \in S(i)$). We

say that a placement is **balanced** if every player possesses exactly n gifts ($L^j = n$ for every player j) and if every gift type appears exactly n times in the placement ($L_i = n$ for every gift type i). It is obvious that the initial placement $L(t)$ is balanced and $\hat{S}$-valid for every $\hat{S}$. If a placement is not $\hat{S}$-valid or it is not balanced, it is called, respectively, an **$\hat{S}$-invalid** placement or an **imbalanced** placement.

We construct the marginal profit functions ζ_i, ζ^j and ζ_i^j such that balanced and $\hat{S}$-valid placements will be more profitable than imbalanced placements and $\hat{S}$-invalid placements; such a construction will force the solution for ConRePP-2 to be balanced and $\hat{S}$-valid. The functions are selected as follows:

$$\zeta_i(x, D_i) = \zeta^j(x) = \begin{cases} x \cdot (n^2+1) & \text{if } x \leq n. \\ -\infty & \text{otherwise.} \end{cases} \tag{10.5}$$

$$\zeta_i^j(L_i^j, D_i^j) = \begin{cases} 0 & \text{if } i = j \text{ or } L_i^j = 0. \\ 1 & \text{if } i \neq j,\ j \in S(i) \text{ and } L_i^j = 1. \\ -\infty & \text{otherwise.} \end{cases} \tag{10.6}$$

It is easy to see that under these functions the following holds:

1. The functions are concave and are legitimate for using in ConRePP-2 (where we set the revenue constants in Eqs. (3.6), (3.7) to be zero, i.e., $R_i = R_i^j = 0$).
2. The profit of $L(t)$ according to Eq. (3.5) is $P(L, D) = 2n^2(n^2+1)$.
3. For every i, j we have $\zeta_i^j(L_i^j, D_i^j) \leq 1$. Thus, the sum of ζ_i^j is bounded from above by n^2.
4. The profit of an $\hat{S}$-invalid placement is $-\infty$ according to Eq. (10.6).
5. The profit of an imbalanced placement must be $< 2n^2(n^2+1)$ since the sum of ζ_i and ζ^j is $\leq 2n^2(n^2+1) - (n^2+1)$ and the sum of ζ_i^j is $\leq n^2$.
6. The profit of any placement which is $\hat{S}$-valid and balanced is $\geq 2n^2(n^2+1)$ (the term $2n^2(n^2+1)$ results from Eq. (10.5), and Eq. (10.6) cannot decrease the profit).
7. Thus, placements which are $\hat{S}$-valid and balanced are more profitable than imbalanced placements or $\hat{S}$-invalid placements. This implies that the optimal solution must be balanced and $\hat{S}$-valid.

The structure of $L(t)$ and the fact that the optimal constrained placement for period $t+1$ ($L(t+1)$) must be balanced and $\hat{S}$-valid implies that $L(t+1)$ must obey:

1. In the transformation from $L(t)$ to $L(t+1)$, no gifts are created or lost (thus they are only moved between players).

2. The total number of gifts possessed by each player is kept constant (is equal to n). So the number of gifts player i gives is identical to the number of gifts player i received.

3. For $i \neq j$, the number of type i gifts player j possesses is either 0 or 1. In case this number is equal to 1, then $j \in S(i)$ i.e., player i can send a gift to player j in a Fair Christmas Game with the friend-list $\hat{S}$. Thus, $\sum_i \sum_j \zeta_i^j(L(t+1)_i^j, D_i^j)$ is equal to the number of gifts delivered between players in a Fair Christmas Game with friend-list $\hat{S}$.

4. The profit of a balanced and $\hat{S}$-valid placement is $2n^2(n^2+1) + \sum_i \sum_j \zeta_i^j(L(t+1)_i^j, D_i^j)$.

These properties derive the following claims:

Claim 10.2. *If $L(t+1)$ is the optimal placement of period $t+1$ – then the transformation from $L(t)$ to $L(t+1)$ must represent a Fair Christmas game. Moreover, if the profit of $L(t+1)$ is $2n^2(n^2+1)+k$ – then the number of gifts delivered between players in the corresponding Fair Christmas game is k.*

Claim 10.3. *Given a friend list $\hat{S}$, any fair game corresponding to $\hat{S}$ where k gifts are delivered between players can be modeled by a placement $L(t+1)$ with profit of $2n^2(n^2+1)+k$.*

These claims show that the solution to the Fair Christmas Game can be found by finding the optimal reposition, as required. □

Remark 10.2. The marginal profit functions $\zeta_i, \zeta^j, \zeta_i^j$ (Eqs. 10.5, 10.6) are all linear functions with limitations over the number of type i resources, the number of region j resources, and type i region j resources.

10.2.2 Hardness of the ConRePP-2

The following theorem, proved in Appendix i, establishes the hardness of the Fair Christmas Problem. This is done by first showing that the Fair Christmas Game Problem is as hard as the Exact Cycle Sum problem, a problem that was introduced in [19].

Theorem 10.4. *There is a polynomial reduction from the Cycle Sum Problem to the Fair Christmas Game Problem.*

Then we quote a result of Papadimitriou et al. [19] showed that the Exact Cycle Sum Problem is at least as hard as the Exact Matching Problem:

Theorem 10.5 ([19]). *There is a polynomial reduction from the Exact Matching Problem to the Cycle Sum Problem.*

Theorems 10.4, 10.5 derive that a polynomial solution for the Fair Christmas Game Problem implies a polynomial solution for the Exact Matching Problem[4]. Combining it with Theorem 10.1 implies the hardness of Constrained-Reposition Placement Problem (ConRePP-2) as follows:

Corollary 10.6. *The Constrained-Reposition Placement Problem (ConRePP-2) is hard as the Exact Matching Problem. I.e., if there is a polynomial solution to the ConRePP-2 – then there is a polynomial solution to the Exact Matching Problem.*

Finally, to imply the hardness of the Constrained-Reposition Placement Problem we observe that polynomially solving the Exact Matching Problem is a studied and well-known problem which has been open for almost 30 years ([19–21]), and graph theory experts have not yet solved it. Providing a polynomial solution to the Constrained-Reposition Problem might be infeasible, and would imply a polynomial algorithm to solve the Exact Matching Problem.

10.3 The Shortest Cycle Operation (SCO) Algorithm

As there is no polynomial time solution for the Constrained-Reposition Placement Problem (as shown in Section 10.2) we present a pseudo-polynomial heuristic algorithm called **SCO** that solves the problem. We prove that SCO finds an optimal solution in the following cases: 1) If the reposition bound r is very large; in particular, SCO can find the unconstrained optimal placement ($r = \infty$). 2) If the reposition bound r is very small. 3) In a single-region system, or a single-type system.

The SCO algorithm was published in [1].

10.3.1 Description of SCO

To describe SCO we use the following definitions:

[4]It can be shown that the problems are equivalent; however, for the sake of this work, this is not required.

Definition 10.1. An **operation** o is a composition of unary operations (i.e., addition or subtraction operations). The **length** l of an operation o is the number of repositions the operation does.

For example, suppose o is the operation that adds two resources of type 1 to region 1, and substrates one resource of type 2 in region 1. If L is a placement such that $L_1^1 = 2$ and $L_2^1 = 2$ then the placement $L' = o(L)$ has four resources of type 1 in region 1, and one resource of type 2 in region 1 ($L'^1_1 = 4$ and $L'^1_1 = 1$). The operation o repositions three resources and thus its length is $l = 3$. It can be seen that an operation that is a composition of l unary operations can be of length smaller than l, if the operations cancel each other out.

Definition 10.2. Given the demand D, the **profitability** of an operation o over placement L, denoted by $\Delta(o, L)$, is the marginal profit of using o over L, i.e., $\Delta(o, L) = P(o(L), D) - P(L, D)$. The operation o is called **profitable** over L if $\Delta(o, L) > 0$.

For example, suppose the demand D_1^1 is deterministic, is equal to 5, and the profit function is $P(L, D) = 5 \cdot \min(L_1^1, 5) - 4 \cdot L_1^1$. Then, the profitability of removing a single resource of type 1 from region 1 is $\Delta(o, L) = [5 \cdot \min(L_1^1 - 1, 5) - 4 \cdot (L_1^1 - 1)] - 5 \cdot \min(L_1^1, 5) + 4 \cdot L_1^1$ which equals 4 if $L_1^1 \geq 6$ and -1 if $1 \leq L_1^1 \leq 6$. Of course, the operation is profitable if $L_1^1 \geq 6$.

Definition 10.3. Given a placement L a **shortest profitable operation** of L is a profitable operation o of length l such that: 1) The length of every profitable operation o' is not less than l. 2) If o' is an operation of length l, then o is more profitable than o', i.e., $\Delta(o, L) \geq \Delta(o', L)$.

It should be noted that every two shortest profitable operations of the same placement L have the same profitability and the same length. Also, unless L is an optimal unconstrained solution, there exists at least one profitable operation, and thus a shortest profitable operation exists.

To this end the SCO algorithm finds, given an initial placement $L(t)$ at period t, the optimal placement with respect to a new demand $D(t+1)$ that can be achieved under a reposition constraint, as described as follows:

SCO algorithm: We set initially $A(0) \leftarrow L(t)$. In each iteration i, we find the shortest profitable operation o_i for placement $A(i)$ and set $A(i+1) \leftarrow o_i(A(i))$. SCO terminates in one of the following conditions: 1) The placement $A(i)$ is the optimal unconstrained solution for $D(t+1)$. 2) The reposition constraint will be violated. 3) The algorithm runs for r iterations at most. We next present an implementation of the SCO algorithm

in a polynomial time. Below, we describe a running example of SCO regardless of its implementation.

Consider a system with two regions and a single resource type. The demand distributions D_1^1 and D_1^2 are deterministic, and are equal to 5 and 3, respectively. The profit function is $P(L, D) = \min(L_1^1, 5) + \min(L_1^2, 3) + p_1 \cdot \min(L_1, 8) - p_2 \cdot L_1^1$, where $p_1 >> p_2 >> 1$ are constants. The initial placement is set to $A(0) = L(t)$, where $L_1^1(t) = 6, L_1^2(t) = 1$ and the reposition bound is $r = \infty$, i.e., the reposition constraint cannot be violated. One can check that the optimal unconstrained placement is $(L_1^1 = 5, L_1^2 = 3)$. The profitability of adding a single resource to $A(0)$ in regions 1 and 2 is equal to $p_1 - p_2 > 0$ and $1 + p_1 - p_2 > 0$, respectively. The profitability of removing a single resource from $A(0)$ in regions 1 and 2, is respectively, $p_2 - p_1 < 0$ and $p_2 - p_1 - 1 < 0$. Thus, the shortest profitable operation adds a single resource to region 2, and SCO sets $A(1) = (6, 2)$. SCO will continue to the next iteration as the reposition constraint is not violated and $A(1) = (6, 2)$ is is not the optimal unconstrained solution and the algorithm cannot run for infinite iterations.

An addition or a subtraction operation over $A(1)$ is not profitable: adding a resource has a profitability $c - p_2 < 0$ where c is a constant, while removing a resource has a profitability of $p_2 - p_1 + c < 0$. Thus, there is no profitable unary operation (i.e., a profitable operation of length 1).

The SCO thus finds a shortest profitable operation of length $l \geq 2$. One might check that SCO in the second iteration adds to region 2 and subtracts (removes) from region 1. SCO will terminate after two iterations, as $A(2) = (5, 3)$ is the optimal unconstrained solution.

Remark 10.3. If the reposition bound in the above running example was $r = 1$ or $r = 2$, then SCO terminates after the first iteration due to the reposition constraint violation and returns the placement $A(1) = (6, 2)$. For a reposition bound larger than 3, SCO returns $A(2) = (5, 3)$.

10.3.2 Key properties of the SCO algorithm

Next, we establish the key properties of SCO presented in the following theorems:

Theorem 10.7 (Optimality of SCO in a single-region system or a single-type system). *In systems with a single region or with a single resource type, SCO will return an optimal solution for the Constrained-Reposition Placement Problem (ConRePP-2).*

Proof. For a single-type system, we show that the U&Me algorithm in Chapter 9 solves the problem. The U&Me algorithm first conducts a sequence of unary operations of

length $l = 1$ (as long as they are profitable), followed by a sequence of operations of move operations of length $l = 2$ (as long as they are profitable). This yields the optimal solution for the Constrained-Reposition Placement Problem. It is easy to see that SCO follows that algorithm and therefore yields the optimal result. The proof holds similarly for a single-region system. □

Theorem 10.8 (Optimality of SCO for small reposition bounds). *For every system there is a constant r_1 such that running SCO over a reposition bound $r \leq r_1$ returns an optimal solution for the Constrained-Reposition Placement Problem. In particular, it can find the optimal unconstrained solution for a large enough r.*

Proof. Suppose the shortest profitable operation of $L(t)$ is of length l. Then, for every $r \leq l-1$ there is no profitable operation of length r that can improve the profit of $L(t)$. To this end, we set $r_1 = l - 1$, and for every $r < r_1$ SCO finds an optimal solution for the Constrained-Reposition Placement Problem[5]. □

Theorem 10.9 (Optimality of SCO for large reposition bounds). *For every system there is a constant r_2 such that running SCO over a reposition bound $r \geq r_2$ SCO returns an optimal unconstrained solution. In particular, for $r \geq r_2$, SCO solves the Constrained-Reposition Placement Problem.*

Remark 10.4. Theorem 10.9 holds due to the fact that every placement should contain up to s resources, where s is the total storage value. If there is no bound over the number of possible placements – then the theorem, theoretically, might not hold.

Proof. Let $D = D(t+1)$ be the new demand in period $t+1$ and $L(t)$ the placement in period t. To prove that there is such r_2 we use the fact that every placement should contain up to s resources, and thus there exists a finite number of placements. We order all possible placements by their profit (according to the demand D). That means, $P(L_1, D) \geq P(L_2, D) \geq P(L_3, D) \dots P(L_n, D)$ where L_1 is the optimal unconstrained solution, and L_n is the placement with the lowest profitability. For every placement L we define its rank $rank(L)$ as the index of L in the order (if $rank(L) = 1$ then L is the optimal unconstrained placement).

We observed that the following properties hold:

- Placements with higher profits have smaller ranks. That means, given two placements L and L' such that L' has a higher profit than L (i.e., $P(L', D) > P(L, D)$) then the rank of L' is smaller than that of L' (i.e., $rank(L') \leq rank(L) - 1$).

[5] It can be shown that for $r_1 = l$ SCO finds an optimal solution for the Constrained-Reposition Placement Problem.

- SCO improves the profit of the previous placement i.e., the profit of the placement computed in the i^{th} iteration is larger than the profit of the placement computed in the $i-1^{th}$ iteration. That means, $P(A(i-1), D) < P(A(i), D)$.

We derive from both properties that the rank of the placement $A(i)$ is $rank(A(i)) \leq rank(A(i-1)) - 1 \leq \ldots \leq rank(A(0)) - i = rank(L(t)) - i$. Thus, regardless of the reposition bound r, SCO must terminate after, at most, $q = rank(L(t))$ iterations. Otherwise $1 \leq rank(A(q)) \leq rank(L(t)) - q = 0$ – a contradiction.

Suppose we run SCO over $r = \infty$, and after $q_{min} < q$ iterations SCO finds the optimal unconstrained solution and terminates. We define r_2 to be a big enough reposition bound that enables SCO to run q_{min} iterations without violating the reposition constraint. We set

$$r_2 = \max_{1 \leq i \leq q_{min}} \sum_{i=1}^{m} \sum_{j=1}^{k} |A_i^j(i) - L_i^j(t)| + 1.$$

Then, the placement SCO finds that in the i^{th} iteration $A_i^j(i)$ cannot violate the reposition constraint for every $1 \leq i \leq q_{min}$. Thus, SCO will not terminate until it finds the optimal unconstrained solution after q_{min} iterations. □

10.3.3 Implementation of the SCO algorithm – outline

Finding the shortest profitable operation is not trivial. The number of available operations is equal to the number of placements that are bounded by a large storage number s, and therefore the number of possibilities that must be examined can be very large (exponential by s); checking if there is a profitable operation may not be simple to implement, let alone in polynomial time. We next propose a polynomial time algorithm that addresses this problem, namely that given a placement L over demand D it finds its shortest profitable operation.

Our strategy is as follows: First (Section 10.3.4), we narrow down the space of shortest profitable operations to operations of very specific characteristics, called extended chains that, roughly speaking, can be described as a sequence of move operations between regions (rather than an arbitrary sequence of add and subtract operations). We show that only these operations are candidates to be the shortest profitable operation and this is the key for our algorithm. Second (Section 10.3.5), we analyze the profitability of these extended chains; roughly speaking, we show that the profitability is equal to the sum of the profitability of each of the move operations. Lastly (Section 10.3.6) we use this information to construct a graph (see Figure 10.1), whose nodes represent the regions and an edge connecting i to j represents the move operation from region i to j where the edge cost is the profit of the corresponding move. This implies that an

extended chain is a path on this graph and the profit of this extended chain is the length of the corresponding path. This allows us to find the shortest profitable operations by finding shortest paths in a graph; this approach allows us to capitalize on the knowledge in the graph theory field and the efficient algorithms designed in that domain.

10.3.4 Shortest profitable operation is an extended chain operation

We characterize that every shortest profitable operation must be an *extended chain operation*. We then show that a profitable shortest cyclic operation can be found in polynomial time.

Definition 10.4 (Extended chain operation). A **move operation** of format $O(j_1 \rightarrow j_2)$ is an operation that moves a resource from region j_1 to j_2[6]. A **chain operation** of format $O(j_1 \rightarrow j_2 \rightarrow \ldots \rightarrow j_n)$ is a concatenation of zero or more move operations, i.e., it moves a resource from j_k to j_{k+1} $(k = 1, 2 \ldots n-1)$. A chain operation $O(j_1 \rightarrow j_2 \rightarrow \ldots \rightarrow j_n)$ where resource of type i_k is moved between j_k and j_{k+1} is called **simple** if $j_{k_1} \neq j_{k_2}$ and $i_{k_1} \neq i_{k_2}$ for every $k_1 \neq k_2$. An **extended chain** $E(j_1 \rightarrow j_2 \rightarrow \ldots \rightarrow j_n)$ is a simple chain operation $O(j_1 \rightarrow j_2 \rightarrow \ldots \rightarrow j_n)$ that in addition, can possibly conduct either (non exclusive) one of the following two unary operations: 1) add a resource to region j_1, or 2) remove a resource from region j_n.

For example, suppose there are two resource types $1, 2$ and two region a, b. Let $L = \{L_1^a = 1, L_2^a = 2, L_1^b = 3, L_2^b = 4\}$. A move operation $O(a \rightarrow b)$ can either move a resource of type 1 or a resource of type 2. Applying the former move operation over L will result in placement $L = (0, 2, 4, 4)$ and the later move operation will result in placement $L = (1, 1, 3, 5)$. An extended chain operation $E(a \rightarrow b)$ can, in addition, add a resource to region a (of types $1, 2$), remove a resource from region b, do both of these, or neither of them. Applying an extended chain operation over L can yield results such as $L = (0, 2, 4, 4), (1, 2, 4, 4), (0, 3, 4, 4)$ and others.

Remark 10.5. 1) In the particular case where the chain length is zero, the extended chain operation $E(j_1)$ is simply a subset of the operation set consisting of a resource subtract at j_1 and resource add at j_1. That means, a resource of type i_1 is replaced by a resource of type i_2 at j_1. 2) The number of repositions an extended chain operation $E(j_1 \rightarrow j_2 \rightarrow \ldots \rightarrow j_n)$ can do is either $2(n-1)$ (if the operation does not add a resource to j_1 nor removes one from j_n), $2(n-1)+1$ (the operation adds a resource to j_1 or removes a resource from j_n, but not both), or $2(n-1)+2$ (the operation adds a resource to j_1 and removes a resource from j_n).

[6] This move is identical to an addition of some type i resource in region j_2 and a subtraction of a type i resource from region j_1.

Our fundamental theorem shows that the search for a shortest profitable operation can be narrow only in extended chains operations.

Theorem 10.10 (Shortest profitable operation are extended chain operations). *For every placement L its shortest profitable operation must be an extended chain.*

Note that the optimal unconstrained placement L_{uncon} is trivially excluded from Theorem 10.10, as there is no profitable operation for L_{uncon}.

The proof of this theorem (based on mapping to a flow graph problem as in Figure 8.1) is postponed to Section 10.3.8, and we now focus on the algorithmic side of SCO. The proof is involved and detailed, and uses a transformation to a flow problem explained in the end of Chapter 8, based on the concavity assumption of the marginal profit functions (Section 3.2). The proof is a long theoretical result heavily based on min-cost arguments in graph theory. As many of our readers are interested more on the algorithmic side of SCO and less in graph theory results, we will present the correctness of the theorem later in Section 10.3.8.

Based on Theorem 10.10, we devise an algorithm that, given a placement L, it finds a shortest profitable operation in $O(k^2(k^2+m))$ steps (k is the number of regions, m is the number of resource types) by exploring the set of extended chains. The algorithm uses the structure of extended chains, which allows the computation of the shortest profitable operation to be equivalent to finding the shortest path in the graph. To compute the edge weights in the graph, in the next subsection we study the structure of the profitability of the extended chain operations as computed using the algorithm.

10.3.5 Profitability structure of an extended chains

To express the profitability structure we recall that the profit function $P(L,D)$ is composed of the sum of marginal profit functions $\zeta_i(D_i,L_i), \zeta_i^j(D_i^j,L_i^j), \zeta^j(L^j)$ (Eq. (3.5)). We denote the **conditional-marginal** functions by $g_i(n)=\zeta_i(n,D_i^j), g^j(n)=\zeta^j(n,D_i^j)$ and $g^j(n)=\zeta^j(n)$, so that

$$P(L,D)=\underbrace{\sum_{i=1}^{m}\sum_{j=1}^{k} g_i^j(L_i^j)}_{\text{region \& type}}+\underbrace{\sum_{i=1}^{m} g_i(L_i)}_{\text{type}}+\underbrace{\sum_{j=1}^{k} g^j(L^j)}_{\text{region}}. \tag{10.7}$$

By the concavity assumption (see Section 3.2), the differential of these functions, $\Delta g(n)=g(n)-g(n-1)$, which are called the **marginal-differential** functions, are monotonically

non-increasing for $n \geq 1$. In the next lemma, we will use the region & type marginal-differential functions Δg_i^j and the region marginal-differential functions Δg^j.

The set of extended chains can be partitioned into five classes such that the profitability structure of each class is expressed slightly differently. One such class is the set of operations $O(j_1 \rightarrow j_2 \ldots \rightarrow j_n)$ (that neither adds a resource to j_1 nor removes one from j_n). For the sake of presentation brevity, we next show the profitability structure for this class of operations. In the Appendix (Appendix k.1), we show the profitability structure for all five classes.

Lemma 10.11. *Let o be an extended chain operation $O(j_1 \rightarrow j_2 \rightarrow \ldots \rightarrow j_n)$ that neither adds a resource to j_1 nor removes one from j_n. Suppose that between j_k to j_{k+1} it moves a resource of type i_k ($k = 1, 2 \ldots n-1$). Then the profitability of o over L is*

$$\Delta(o, L) = \Delta^+(j_1) + \sum_{k=1}^{n-1} \Delta_{i_k}(j_k \rightarrow j_{k+1}) + \Delta^-(j_n). \tag{10.8}$$

where,

$$\Delta_i(j_1 \rightarrow j_2) = \Delta g_i^{j_2}(L_i^{j_2} + 1) - \Delta g_i^{j_1}(L_i^{j_1}) \tag{10.9}$$

$$\Delta^+(j) = -\Delta g^j(L^j) \tag{10.10}$$

$$\Delta^-(j) = \Delta g^j(L^j + 1). \tag{10.11}$$

Intuitively, $\Delta_i(j_1 \rightarrow j_2)$ represents the change in the region & type part of Eq. (10.7) affected by the moving of type i resource from j_1 to j_2. The terms $\Delta^+(j), \Delta^-(j)$ represent the change in the region part of Eq. (10.7) affected by respectively removing and adding a resource to region j.

Proof. By the definition of profitability and Eq. (10.7), the profitability of o over L (denoted by $o(L)$) is equal to the sum of three terms: 1) the sum of region & type marginal differentials $g_i^j(o(L)_i^j) - g_i^j(L_i^j)$ (where $o(L)_i^j$ denotes the number of resources $o(L)$ contains of type i in region j) over all regions j and resource types i; 2) the sum of type marginal differentials $g_i(o(L)_i) - g_i(L_i)$ for all resource types i; and 3) the sum of region marginal differentials $g^j(o(L)^j) - g^j(L^j)$ over all regions j.

First, we consider the effect of a move operation over the region & type function term (sum of $g_i^j(o(L)_i^j) - g_i^j(L_i^j)$): Suppose that o moves a resource of type i from region j_1 to region j_2. Then o adds a resource of type i to region j_2, and removes a resource of type i from region j_1. Therefore, $o(L)_i^{j_2} = L_i^{j_2} + 1$ and $o(L)_i^{j_1} = L_i^{j_1} - 1$. For $j = j_2$, the term $g_i^j(o(L)_i^j) - g_i^j(L_i^j)$ equals $g_i^j(L_i^j + 1) - g_i^j(L_i^j) = \Delta g_i^j(L_i^j + 1)$. For $j = j_1$, the term

$g_i^j(o(L)_i^j) - g_i^j(L_i^j)$ equals $g_i^j(L_i^j - 1) - g_i^j(L_i^j) = -\Delta g_i^j(L_i^j)$. Thus, the move operation contributes to the region & type terms $\Delta g_i^{j_2}(L_i^{j_2} + 1) - \Delta g_i^{j_1}(L_i^{j_1}) = \Delta_i(j_1 \to j_2)$. Since for every $k = 1, 2 \ldots, n-1$ the operation o moves a resource of type i_k from j_k to region j_{k+1}, then o contributes to the region & type terms $\sum_{k=1}^{n} \Delta_{i_k}(j_k \to j_{k+1})$. For $(i,j) \neq (i_k, j_k)$ and $(i,j) \neq (i_k, j_{k+1})$ the number of type i resources in region j does not change, and $g_i^j(o(L)_i^j) - g_i^j(L_i^j) = 0$.

Next, consider the region function term (sum of $g^j(o(L)^j) - g^j(L^j)$): For every region $j \neq j_1, j_n$ the number of resources in region j does not change, and $g^j(o(L)^j) - g^j(L^j) = 0$. The operation o subtracts a resource in region j_1 and it adds a resource in region j_n. Thus, the sum over regions $g^j(o(L)^j) - g^j(L^j)$ over all regions j is $\Delta g^{j_n}(L^{j_n} + 1) - \Delta g^{j_1}(L^{j_1}) = \Delta^+(j) + \Delta^-(j)$, as required.

Lastly, consider the type function term (sum of $g^j(o(L)^j) - g^j(L^j)$). For these we observed that the total number of resources from each type does not change. Thus, it is equal to 0.

□

Remark 10.6. Lemma 10.11 assumes that the operation o is an extended chain operation, and by definition this means that it is a simple operation. The lemma does not hold if o is an $O(j_1 \to j_2 \to \ldots \to j_n)$ operation that is not simple. For example, suppose that o is an $O(j_1 \to j_2 \to j_1)$ operation that moves a resource of type i from region j_1 to j_2, and the same resource from j_2 to j_1. Then, according to Lemma 10.11, its profitability depends on the marginal-differential functions Δg, which are not necessarily equal to zero[7]. However operation o is the identical operation ($o(L) = L$ for every placement L), and thus its profitability is equal to zero.

10.3.6 The algorithm for finding a shortest profitable operation

As the profitability structure of an extended chain operation differs across operations of the five classes, the algorithm finds the shortest profitable operation for every particular class. Then, the algorithm will compare the different shortest profitable operations and will choose the best operation among them. Except for the fifth class, the implementation of the algorithm in each class is similar, with slight differences. The implementation for the fifth class is a little more complex and takes more time to compute, by a factor of k, where k is the number of regions (which is typically small), and can be seen in Appendix k.2. For the sake of presentation, we will show how the algorithm finds the

[7]For instance if $\Delta g^{j_1}(n) = n$ and $\Delta g_i^{j_1}(n) = \Delta g_i^{j_2}(n) = 0$, then the profitability of o is $(-1) \cdot (L^{j_1}) + 0 + (L^{j_1} + 1) = 1$

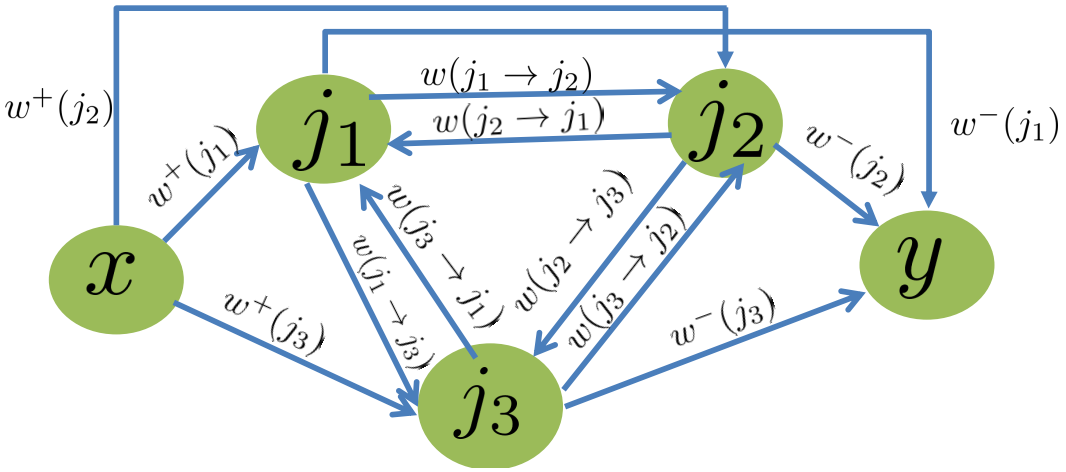


FIGURE 10.1: The graph with three regions j_1, j_2, j_3 and with the appropriate weight edges $w(j_1 \to j_2) = (-1) \cdot \max_i \Delta_i(j_1 \to j_2)$, $w^+(j) = (-1) \cdot \Delta^+(j)$, $w^-(j) = (-1) \cdot \Delta^-(j)$.

shortest profitable operation over the class of chain operations $O(j_1 \to j_2 \to \ldots \to j_n)$, which is the first class of operations. Our full algorithm is given in Appendix k.2.

Given the placement L, in the first stage, the algorithm creates a complete digraph of region vertices $1, 2 \ldots, k$. The graph G (depicted in Figure 10.1 for 3 nodes j_1, j_2, j_3) represents all extended chain operations $E(j_{l_1} \to j_{l_2} \to \ldots \to j_{l_n})$ such that between regions j_{l_m} and $j_{l_{m+1}}$ it moves the best resource type $i(j_{l_m} \to j_{l_{m+1}})$ between them. More formally, the algorithm defines over the edges (j_{l_1}, j_{l_2}) weights $w(j_{l_1} \to j_{l_2}) = (-1) \cdot \max_i \Delta_i(j_{l_1} \to j_{l_2})$, and saves $i(j_{l_1} \to j_{l_2}) = \arg\max_i \Delta_i(j_{l_1} \to j_{l_2})$ (i.e., the resource type that maximizes Eq. (10.9)). The reason we multiply $\Delta_i(j_{l_1} \to j_{l_2})$ by (-1) is to convert a max profitable problem into a shortest path problem (that requires finding minimum-length path, where we consider the edge weights as edge lengths).

The algorithm adds two nodes: a source node x and a sink node y, and connects for every region j edges (x, j) and (j, y) of respectively weight $w^+(j) = (-1) \cdot \Delta^+(j)$, $w^-(j) = (-1) \cdot \Delta^-(j)$ (defined in Eqs. (10.10), (10.11)). The graph G with its edge weights is depicted in Figure 10.1.

We observed that the path $P = (x, j_1, j_2, \ldots, j_{i_{min}-2}, y)$ in the graph corresponds to the best chain operation o of format $O(j_1 \to j_2, \ldots \to j_{i_{min}-2})$, and according to Lemma 10.11, the weight of the path P is equal to (-1) multiplied by the profitability of o. We will use this observation to find the shortest profitable operation.

After constructing G, our algorithm considers the edge weights as edge lengths and uses the Bellman-Ford algorithm [72], which computes in the i^{th} iteration the shortest

path between x and y of i edges at most[8]. It stops in the first iteration i_{min} where the shortest path with i_{min} edges has negative weight. If the algorithm finds that the shortest path (and its length) is $(x, j_1, j_2, \ldots, j_{i_{min}-2}, y)$, then the algorithm returns the chain operation $O(j_1 \rightarrow j_2, \ldots \rightarrow j_{i_{min}-2})$ that moves from region j_k to j_{k+1} a resource of type $i(j_k \rightarrow j_{k+1})$. According to the next claim, the algorithm finds a shortest profitable operation:

Claim 10.12. *The algorithm finds a shortest profitable operation over the class of chain operations* $O(j_1 \rightarrow j_2 \rightarrow \ldots \rightarrow j_n)$.

Proof. According to Lemma 10.11, the profitability of an operation of format $O(j_1 \rightarrow j_2 \rightarrow \ldots \rightarrow j_n)$ is less than or equal to the weight of the path $(x, j_1, j_2, \ldots, j_n, y)$ multiplied by -1. Also, the profitability of an operation that moves a resource of type $i(j_k \rightarrow j_{k+1})$ from j_k to j_{k+1} is exactly equal to the weight of the path $(x, j_1, j_2, \ldots, j_n, y)$ multiplied by -1. Therefore, the weight of the shortest path of length $n+2$ $((x, j_1, j_2, \ldots, j_{n-2}, y))$ is equal to the profitability of the best extended chain operation of length n $(O(j_1 \rightarrow j_2 \rightarrow \ldots \rightarrow j_n))$ multiplied by (-1).

Thus, there is a profitable extended chain operation of length n iff there is a shortest path of length $n+2$ with negative weight. Since the shortest profitable operation is an operation of minimum length, then it corresponds to a negative-weighted shortest path with a minimum number of edges, which is what the algorithm finds. □

10.3.7 Time complexity of SCO

The complexity of creating the graph is $O(k^2 \cdot m)$ where k is the number of regions and m is the number of resource types. This is because computing $w(j_1 \rightarrow j_2)$ takes $O(m)$. Using Bellman-Ford on a graph $G = (V, E)$ takes $O(VE)$ steps. Since our graph contains $O(k)$ vertices and $O(k^2)$ edges, then the running time of the second stage is $O(k^3)$. Thus, the time complexity of our algorithm over the class of chain operations that neither add a resource to j_1 nor subtract one from j_n not is $O(k^2(k+m))$.

The algorithm presented in this section finds the shortest profitable operation among the first class of operations. In Appendix k.2, we present four additional algorithms that find the shortest profitable operation in the other classes. Three of these algorithms, which correspond to the second, the third, and the fourth classes, have a similar implementation as the algorithm presented in this section. These algorithms run in $O(k^2(k+m))$ steps. Finding the shortest profitable operation among the operations of the fifth class, however,

[8] In the first iterations, when $i = 1$, there exists no path of length $\leq i$ between x and y. At these stages the length of the "shortest path" between x and y is computed as ∞. Once a path exists, this length receives a finite value.

is solved in a more expensive time complexity of $O(k^2(k^2+m))$ instead of $O(k^2(k+m))$. Thus, the full algorithm which finds and compares the shortest profitable operations in each class takes $O(k^2(k^2+m))$ steps. We can use a similar algorithm that, instead of using region nodes $1, 2 \ldots, k$, uses resource type nodes $1, 2, \ldots, m$. The complexity of such an algorithm is $O(m^2(k+m^2))$. Choosing the right algorithm as a function of the parameters (k, m) allows us to find the shortest profitable operation in $O(\min(k,m)^3(k+m))$ steps.

Since SCO runs for r iterations at most (r is the reposition bound) – then the complexity of SCO is $O(r\min(k,m)^3(k+m))$.

Remark 10.7. We assume that computing the marginal-differential functions $\Delta g(n)$ for every integer n takes $O(1)$. Otherwise, if it takes $O(e)$ time to compute these values – then the complexity of the algorithm is $O(r\min(k,m)^3(k+m)+re\cdot\min(k,m)^2(k+m))$.

10.3.8 Every shortest profitable operation is an extended chain – proof of Theorem 10.10

Proof of Theorem 10.10. We use the reduction to a min-cost flow described in Theorem 8.6 and Figure 8.1. The methodology of reduction to a graph flow problem allows us to conduct the analysis in the domain of graph theory. Given a flow f_L corresponding to placement L in the four-layer graph G^4, we use the **residual multigraph** $G^4_{f_L}$ that will allow one to represent all possible operations o (placement changes) on L by augmenting the flow along its cycles.

The residual multigraph $G^4_{f_L}$ is constructed as follows: 1) If the flow of an edge e in G^4 is $f_L(e)=0$, then add e to the residual multigraph $G^4_{f_L}$; the edge e in G^4 that was added to the residual multigraph $G^4_{f_L}$ is called **original edge**. 2) If the flow of an edge $e=(u,v)$ in G^4 is $f_L(u,v)=1$, then add its **reverse edge** (v,u) to the residual multigraph $G^4_{f_L}$. In such case, the original edge (u,v) **is not** added to the residual multigraph $G^4_{f_L}$. Given a weight function of the four-layer graph edges $w : E \to R^+$, we define the **residual** weight function over the edges of the residual multigraph $G^4_{f_L}$ as $w_{f_L}(e) = w(e)$ for every original edge and $w_{f_L}(v,u) = -w(u,v)$ for every reverse edge (v,u) in $G^4_{f_L}$. For example, the residual multigraph of Figure 8.1 is presented in Figure 10.2. The formal definition of a residual multigraph G_f for a general (multi) graph G and flow f is given in Appendix k.3.

When an operation o changes a placement L to placement $o(L)$, the flow corresponding to L, f_L, is changed to a flow corresponding to $o(L)$, $f_{o(L)}$. The flow value of both flows is equal to $|f_L| = |f_{o(L)}| = s$. It is well-known that when a flow f in a general graph G can change to another flow f' with the same flow value ($|f| = |f'|$), the change

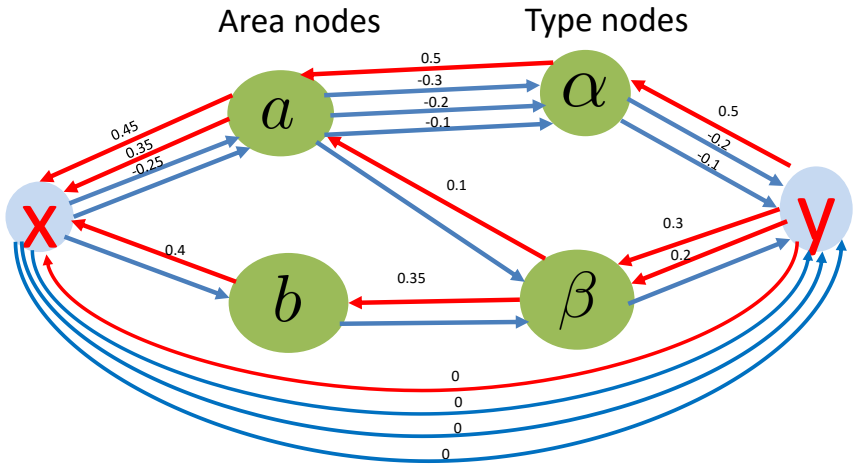


FIGURE 10.2: The residual multigraph $G^4_{f_L}$ where $L^a_\alpha = L^a_\beta = 1$ and $L^b_\alpha = 0, L^b_\beta = 1$ (the flow f_L is depicted in Figure 8.1). Red edges are reverse edges with flow $f_L(e) = 1$, and blue edges are original edges with flow $f_L(e) = 0$. Between every two nodes there are a total of $s(= 4)$ edges. We omit some blue edges from the graph for the sake of presentation.

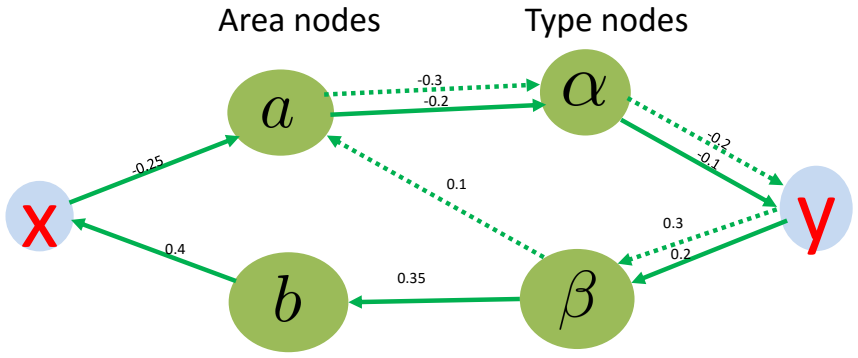


FIGURE 10.3: The residual operation multigraph $G(o, L)$, a subgraph of $G^4_{f_L}$ from Figure 10.2, where o is an operation that adds two α-type resources to region a, and removes two β-type resources, one from region a and one from region b. The dotted-line edges represent the lowest weighed cycle C in $G(o, L)$. An augmentation through C will add one α resource to a and remove one β resource from a (this operation is $o^*(o, L)$).

can materialize by augmenting the flow through cycles $C_1, C_2 \dots C_n$ in the residual multigraph G_f [See The Augmenting Cycle Theorem (Theorem k.4) in Appendix k.3]. We define the **residual operation multigraph** $G(o, L)$, a sub-graph of $G^4_{f_L}$, as the union of cycles such that augmenting 1 unit of flow along the cycles changes the flow f_L to $f_{o(L)}$ in $G^4_{f_L}$, i.e.,

$$G(o, L) = \bigcup C_i. \tag{10.12}$$

A detailed description of the structure (and how to build) $G(o, L)$ is given in Appendix k.4 (Observation k.5) and an example is given in Example 10.1 below. By

Claim 8.7, we derive that the weight of a flow f_L is equal to minus the profit of L, plus some constant c (see Claim 8.7). Thus, the sum of edge weights of the residual operation multigraph is equal to minus the operation profitability, i.e.,

$$\sum_{e \in G(o,L)} w_f(e) = -\Delta(o, L). \tag{10.13}$$

Next, we define a **lowest-weight cycle** C as a simple cycle with minimum weight in $G(o, L)$. Note that $G(o, L)$ might contain more than one lowest-weight cycle.

Example 10.1. Let $G^4_{f_L}$ be the residual graph depicted in Figure 10.2, which corresponds to the placement $L^a_\alpha = L^a_\beta = 1$ and $L^b_\alpha = 0, L^b_\beta = 1$. Note that the flow f_L is depicted in Figure 8.1. Suppose that o is an operation that adds two α-type resources to region a, and removes two β type resources, one from region a and one from region b. The residual operation multigraph $G(o, L)$ is depicted in Figure 10.3, where every original edge corresponds to adding a resource, while every reverse edge corresponds to removing a resource. Note that the residual multigraph comprises a collection of cycles, as stated above. Eventually, after augmenting flow through $G(o, L)$ we change the flow f_L corresponding to placement L to the flow corresponding to $o(L)$. The lowest-weight cycle C, i.e., the simple cycle with the lowest weight in $G(o, L)$, is depicted through dotted-line edges.

Next, given two operations, o and o', we say that o is a **sub-operation** of o' and denote $o' \subseteq o$ if o' uses a subset of unary operations (operations that either add or remove a single resource) from o'. For example, consider o in Figure 10.3, which adds two α-type resources to region a, and removes two β type resources, one from region a and one from region b. It has a sub-operation o' that adds one α-type resource to a and removes one β type from b. We can see that operation o' corresponds to augmenting the flow through the lowest-weight cycle C. Also, note o' is an extended chain of zero length (See Remark 10.5 in Section 10.3.4).

This leads to our first lemma used to prove Theorem 10.10:

Lemma 10.13. *For every operation o and placement L there exists a sub-operation of o, to be denoted by $o^*(o, L) \subseteq o$ that possesses the following two properties: 1) There is a lowest-weight cycle C such that augmenting flow through C corresponds to $o^*(o, L)$, i.e., it changes the flow f_L to $f_{o^*(o,L)(L)}$, and 2) operation $o^*(o, L)$ is an extended chain operation or a composition of two operation-disjoint extended chain operations.*

Proof. The detailed proof requires many details. For this reason we provide here the main arguments of the proof. The full formal proof can be seen in Appendix k.4.

The proof first characterizes the structure of the residual operation multigraph $G(o, L)$ (see Observation k.5). One such property is that for every region node j and resource type node i, the residual operation multigraph $G(o, L)$ (and therefore every lowest-weight cycle C) cannot contain both original edges from j to i and reverse edges from j to i. Now suppose C is a lowest-weight cycle in the residual operation multigraph $G(o, L)$. We define operation $o^*(o, L)$ such that for every region node j and resource type node i it does the following:

For every region node j and resource type node i do:

- If C contains original edges from j to i, then $o^*(o, L)$ adds a type i resource to region j.
- If C contains reverse edges from i to j, then $o^*(o, L)$ removes a type i resource from region j.
- If C does not contain edges between i and j, then $o^*(o, L)$ will not change the number of type i resources from region j.

We show that there is a lowest-weight cycle C', such that augmenting flow through C' changes the flow corresponding to placement L to the flow corresponding to placement $o^*(o, L)(L)$ (Claim k.6).

Next, we characterize all cycles in the residual operation multigraph $G(o, L)$ (Claim k.7) and show that every cycle can be one of six classes. Five of these classes represent extended chain operations, and one class represents a composition of two disjoint extended chain operations. For example, one of these classes contains all cycles of format $j_1 \to i_1 \to j_2 \to i_2, \ldots, \to j_n \to x \to j_1$ where j_k are regions and i_k are resource types. If the lowest-weight cycle C is a cycle of such a class, then operation O_c is the extended chain that moves a resource of type i_k from j_k to j_{k-1}. □

Having shown that for every operation o there exists a sub-operation (denoted by $o^*(o, L)$) corresponding to a lowest-weight cycle C, which is an extended chain operation (or composition of two disjoint extended chain operations), we can conclude the proof Theorem 10.10 by showing that if o_L is the shortest profitable operation of L, then $o_L = o'(o_L, L)$. This is proven in the following lemma:

Lemma 10.14. *Let L be a placement and let o_L be its shortest profitable operation. Then the following claims hold:*

1. *The operation o_L is equal to the operation $o'(o_L, L)$ chosen in Lemma 10.13.*
2. *The operation o_L is not a composition of two disjoint extended chain operations.*

Proof. First, we show that $o'(o_L, L)$ must be a profitable operation. Let C be the lowest-weight cycle C corresponding to $o'(o_L, L)$. Suppose that C is a non-negative cycle. Since C is a lowest-weight cycle in $G(o_L, L)$, all cycles $C_1, C_2, \ldots, C_n$ composing the residual operation graph $G(o, L) = \bigcup_{i=1}^{n} C_i$, have non-negative weights, i.e., $w_f(C_i) \geq 0$. Thus, the profitability of o_L, which is equal to minus the sum of cycles weights, i.e., $\Delta(o, L) = -\sum_{e \in G(o,L)} w_f(e) = -\sum_{i=1}^{n} w_f(C_i)$ (Eqs. (10.13),(10.12)), is non-positive – a contradiction, as o_L is a shortest profitable operation, and in particular, a profitable operation. Thus, the operation $o'(o_L, L)$, which corresponds to augmenting flow through C and its profitability is minus the weight of C, is profitable.

Suppose that $o'(o_L, L)$ is not equal to o. Then $o'(o_L, L)$ is a sub-operation of o that strictly uses fewer repositions than o, which is profitable. This forms a contradiction since o is a shortest profitable operation. Therefore $o = o^*(o, L)$, i.e., o is an extended chain or a composition of two disjoint extended chain operations.

To prove the second part of claim, suppose that o is a composition of two disjoint extended chain operations o_1, o_2. Then, one of these sub-operation operations is profitable (otherwise, o is not profitable) that uses fewer repositions than o – a contradiction as o is a shortest profitable operation. Therefore, o must be an extended chain operation. □

□

10.4 A Heuristic Solution to the Contained-Reposition Problem (ConRePP-2) – The SCC Algorithm

In addition to the SCO heuristics, we present a polynomial heuristic algorithm called **SCC** that solves the problem. It uses the reduction to the four-layer graph G_f^4 presented in Theorem 8.6, Section 8.4, where we have shown that an optimal min-cost flow solution in G_f^4 is a flow corresponding to the optimal placement that solves the Unconstrained-Reposition Placement Problem (SPP-4). One particular algorithm for solving the optimal min-cost flow problem, is the well-known Cycle-Canceling algorithm, which cancels in each iteration a negative cycle. Canceling a negative cycle reduces the cost of the flow associated with the placement, and thus increases the profit of the placement. We will utilize this property to modify a (previously determined) placement and adapt it to new demand distribution.

The Shortest Cycle-Canceling (SCC) algorithm we propose for the dynamic placement problem is a variation of the well-known Cycle-Canceling algorithm. To implement SCC, we combine the Cycle-Canceling algorithm with our proposed **Minimum-Length**

negative Cycle (MLnC) algorithm that uses dynamic programming for finding a **minimum-length negative cycle**. The minimum-length negative cycle is a negative cycle with the minimum number of edges l_{min} and among all negative cycles of length l_{min}, it is the one with the minimum cost.

SCC is a dynamic Cycle-Canceling algorithm, given the reposition bound r. SCC ensures that given an initial placement $L(t)$ computed at period t, it finds a placement $L(t)$ such that 1) the reposition cost is always less than or equal to r, i.e., $\sum_{i=1}^{m}\sum_{j=1}^{k}|L(t)_i^j - L(t+1)_i^j| < r$. 2) The placement cost deviation of $L(t+1)$. with respect to $D(t+1)$, is smaller than that of $L(t)$.

SCC finds a minimal (in the number of edges) cycle. As the augmenting a cycle with minimal number of edges can approximate the operation with minimal length, we can augment as many cycles as possible. Note that SCC and SCO are similar to one another.

10.4.1 A comparison between SCO and SCC

While SCO finds the shortest profitable operation corresponding to a placement L, SCC (See Section 10.4) finds the minimum-length negative cycle in the residual graph $G^8_{f_L}$ corresponding to L. We believe that there are many cases where the minimum-length negative cycle is the shortest profitable operation, and thus SCC can be a good approximation of SCO. In addition, SCC is easier to implement than the more complex SCO algorithm. Thus, we used SCC in our performance evaluation later (Section 10.6). Later, in Section 11.4, we evaluate the performance of SCO for the special case of a single region.

SCO runs faster than SCC: The time complexity of SCC takes $O(rmk(k+m)^2)$, where m is the number of resource types, k is the number of regions, and r is the reposition bound. The time complexity of SCO is $O(r \cdot \min(k,m)^3(k+m))$, and therefore SCO significantly runs faster than SCC.

10.4.2 Preliminaries: Cycle-Canceling algorithm

The well-known Cycle-Canceling algorithm solves the min-cost flow problem for general graphs. Given a flow f on a graph $G = (V, E)$, the Cycle-Canceling algorithm uses the well-known **residual graph** $G_f = (V, E_f)$. Every edge in the residual graph is associated with weight w_f and capacity c_f (also called the *residual capacity*). The residual graph G_f is constructed from G and f using the following steps: 1) Add to G_f edges from G, such that every edge $(v, v') \in E$ will have weight $w_f(v, v') = w(v, v')$ and

capacity $c_f(v,v') = c(v,v') - f(v,v')$. 2) Add the reverse edges of G. This means that if $(v,v') \in E$, then add edge (v',v) to G_f with weight $w_f(v',v) = -w(v,v')$ and capacity $c_f(v',v) = f(v,v')$. Note that for every edge e in G_f we have $c(e) \geq 0$. 3) Set the weight of every edge $e \in E_f$ with zero residual capacity ($c_f(e) = 0$) to $w_f(e) = \infty$.

The Cycle-Canceling algorithm works iteratively, and in the i^{th} iteration it finds a negative cycle in G_f. It executes the following steps in each iteration: 1) Find a negative cycle C in G_f using some negative cycle detection algorithm. If there is no negative cycle then the algorithm terminates. 2) We augment the flow by $\delta = \min c_f(e) | e \in E > 0$ units of flow through C. This means that if $(v,v') = e \in C$ is in the original graph (i.e., $e \in E$), then we update $f(e) \leftarrow f(e) + \delta$, and if the reverse edge is in G (i.e $(v',v) \in E$), then we update $f(v',v) \leftarrow f(v',v) - \delta$. 3) We update a new residual graph G_f according to the updated flow f.

The following theorem is established in [70]:

Theorem 10.15. *1) In each iteration of the Cycle-Canceling algorithm the cost of the flow decreases. 2) If there is no negative cycle in G_f then f is a min-cost flow. 3) The flow value $|f|$ does not change during the algorithm.*

Example 10.2. The flow f_L depicted in Figure 10.4 is corresponding to a placement L, where $L^a_\alpha = L^a_\beta = 1$, and $L^b_\beta = L^b_\alpha = 0$. Its residual graph, is depicted in Figure 10.5. The profit of f_L is equal to -2.2, and thus, according to Claim 8.7, the profit of L is equal to $P(L,D) = 2.2 + c$, where c is some constant. Edges with infinite weight, and other unnecessary blue edges, are omit from residual graph.

Cycle $C_1 = (x, b, \beta, y, x)$ is the minimum-length negative cycle, i.e., the negative cycle with a minimum number of edges (only 4), and among cycle with 4 edges – it has minimum weight (-0.95). Cycle C_1 is depicted in Figure 10.5 by dotted lines. It should be noted that although cycle $C_2 = (x, b, \beta, a, \alpha, y, x)$ is the one with the minimum cost (is equal to -1.15) it is not the minimum-length negative cycle, as it contains 6 edges.

By sending flow along C_1 we augment f_L by -0.95. This will, eventually, lead to a new flow $f_{L'}$, as depicted in Figure 10.6. The new flow $f_{L'}$ is corresponding to a placement L', where $L'^a_\alpha = L'^a_\beta = L'^b_\alpha = 1$ and $L^b_\alpha = 0$. This means, by augmenting the flow f_L to $f_{L'}$ we changed the corresponding placement from L to L', and simply added one resource of type α to region b.

The weight of $f_{L'}$ is equal to $w(f_L) + w(C_1) = -3.15$, and thus, according to Claim 8.7, the profit of of L' is $P(L',D) = c + 3.15 = P(L,D) - w(C_1)$. In other words, flowing along the minimum-length negative cycle C_1 augments the profit of placement by $-w(C)_1 = 0.95$.

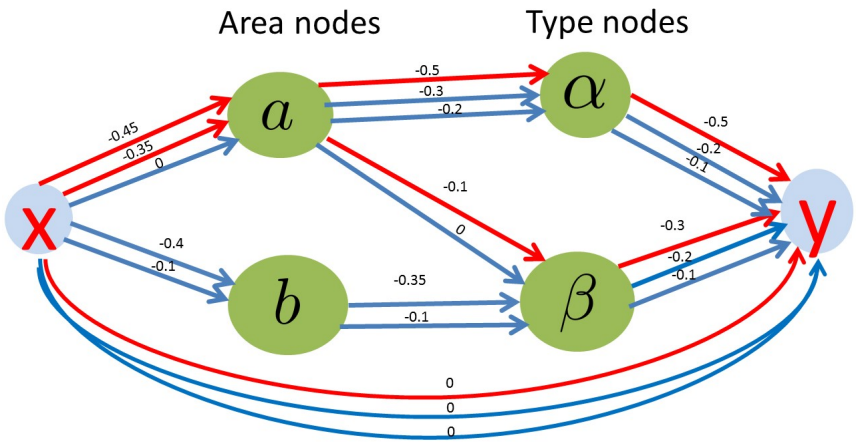


FIGURE 10.4: The flow f_L in the multigraph G^4 where $L^a_\alpha = L^a_\beta = 1$ and $L^b_\alpha = L^b_\beta = 0$. Red edges are edges with 1 unit of flow, and blue edges with 0 units of flow. Between every two nodes there are a total of $s(= 3)$ edges. We omit some blue edges from the graph for the sake of presentation.

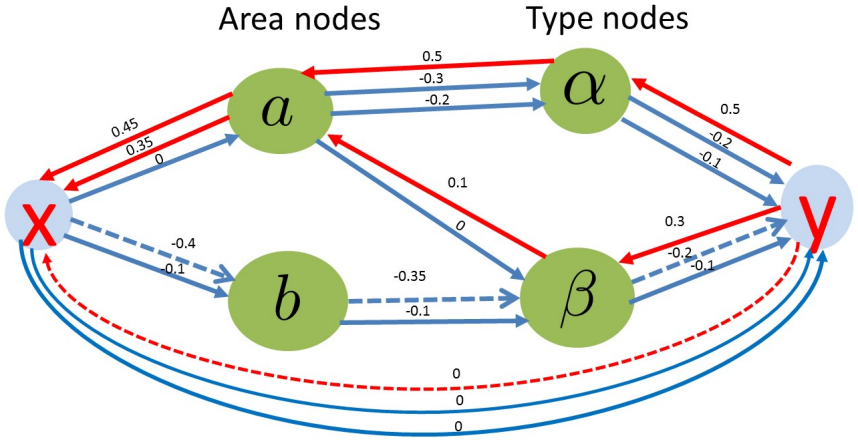


FIGURE 10.5: The residual graph G_{f_L}. The minimum-length negative cycle C_1 is depicted by dotted lines.

10.4.3 Minimum-Length negative Cycle (MLnC) algorithm

The implementation of the Minimum-Length negative Cycle (MLnC) algorithm is presented in Appendix j.

10.4.4 The Shortest Cycle-Canceling algorithm (SCC)

SCC is a dynamic Cycle-Canceling algorithm with a threshold parameter r. Initially, the algorithm builds the residual graph of the four-layer graph G^4_f with respect to the demand at time t, $D(t)$, and the flow f_{init} representing the placement at t, $L(t)$. In each iteration, SCC holds a flow f_{prev} of the previous iteration, where in the first iteration

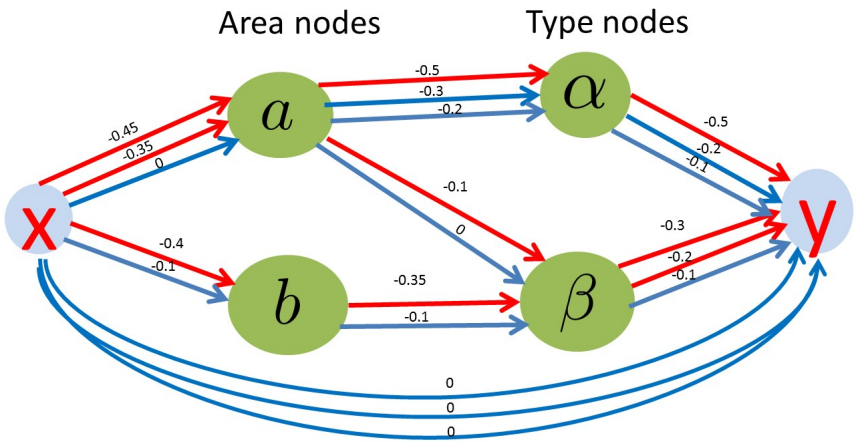


FIGURE 10.6: The augmented flow $f_{L'}$ in the four-layer graph G^4, after sending flow along C_1. The flow $f_{L'}$ is corresponding to the placement L' where $L'^a_\alpha = 2, L'^a_\beta = L'^a_\alpha = 1$ and $L^b_\alpha = 0$.

we set $f_{prev} = f_{init}$. SCC uses MLnC to find the minimum-length negative cycle in G_f, denoted as C. If there exists no cycle, then f_{prev} is a min-cost flow, and its associated placement $L^j_i = f_{prev}(a^i, t^j)$ must be the optimal-cost placement with respect to the demand $D(t+1)$.

We obtain f_{curr} from augmenting the flow through cycle C. We check whether the placement associated with f_{curr} violates the reposition cost constraint, i.e., $\sum_{i=1}^{m}\sum_{j=1}^{k} |L(t-1)^j_i - f_{curr}(a^i, t^j)| < r$. If so, then the algorithm sets the placement for the next period $L(t)$ to be the placement associated with the previous flow f_{prev}. Otherwise, the algorithm will set $f_{prev} = f_{curr}$ and will continue until either f_{curr} violates the reposition cost constraint or the algorithm runs for exactly r iterations.

The following theorem establishes the properties of SCC:

Theorem 10.16. *Let r be the threshold parameter of SCC. Then the following conditions hold: (1) SCC increases the profit of the placement in each iteration. (2) The reposition cost of SCC is r at most. (3) If there is no negative cycle, then SCC returns the optimal placement.*

Proof. **Proof of (1):** Since augmenting the flow by a negative cycle reduces the cost of its corresponding flow (according to Theorem 10.15) and the profit of a placement L is equal to a constant c minus the cost of its corresponding flow f_L (Claim 8.7) then SCC increases the profit of the placement in each iteration.
Proof of (2): Since SCC does not violate the reposition constraint, then the reposition cost of SCC is r at most.
Proof of (3): If there is no negative cycle, then the corresponding flow f_L is a min-cost flow (Theorem 10.15) and thus L is the optimal placement (according to Claim 8.7). □

SCC finds a minimal (in number of edges) cycle, so it can augment as many cycles as possible without violating the reposition cost constraint.

10.4.5 Time complexity of SCC

In a similar way to Sections 7.8 and 6.3, we can equivalently implement SCC over the bipartite-like graph G_f^B that contains $|V| = O(m+k)$ vertices (m number of resource types, k number of regions) and $|E| = O(mk)$ edges. Imitating the behavior of SCC over the bipartite-like graph will reduce the time complexity of an MLnC iteration to only $O(|V|^2|E|) = O(mk(k+m)^2)$. The formal details are presented in Appendix j. Since SCC can run for r iterations at most, the time complexity of SCC is $O(rmk(k+m)^2)$, which is polynomial.

Remark 10.8. We assume that calculating the marginal-differential values $\Delta g(n)$ for every integer n takes $O(1)$. Otherwise, if it takes $O(e)$ time to compute these values, then the complexity of the algorithm is $O(rmk(k+m)^2 + r \cdot e \cdot m \cdot k)$.

10.4.6 Practical Considerations and a General Algorithm

SCC, as designed above, aims to bound the reposition cost incurred in any operation of the algorithm. As such, SCC operates regardless of the placement cost deviation, and there might be situations where the application will yield resource repositions, whose placement cost deviation is tiny. Such repositions can be avoided using the Hybrid algorithm, as presented in the next section.

10.5 An Online Hybrid Multi-Period Algorithm (Hybrid)

In an online multi-period environment, one must decide on the optimal placement of resources over multiple periods, $t = 1, 2, 3, \ldots$, where at each period t one receives a prediction of the demand at period $t+1$ and needs to decide how to reposition the resources at time $t+1$ as a function of the demand at $D(t+1)$ and the placement at $t+1$.

Two challenges must be faced by such an algorithm: 1) The number of resources that can be repositioned between t and $t+1$ is usually bounded, due to physical or economical constraints. 2) It is preferable to overcome temporal fluctuations and to avoid back-and-forth repositions due to temporal changes in demand.

To address these objectives, the algorithm combines the SCC[9] algorithm developed in Section 10.4, and the sensitivity results developed in Chapter 8. When it senses (using the sensitivity results) a large deviation in the demand it conducts SCC and repositions as many resources as possible (up to the maximum number possible, r_{max}). When it senses small deviation in the demand, it conducts only a smaller number of repositions to avoid redundant fluctuations.

To this end, the Hybrid algorithm holds three threshold parameters ϵ, r_{min} and r_{max}, where $r_{min} < r_{max}$. At time t the Hybrid algorithm holds a reference demand $D_{ref}(t+1)$, which is equal to $D(\tau)$ for some $\tau < t+1$, where τ is the last time the algorithm conducted a large reposition. It also holds, as a reference, the placement at time t, $L(t)$. Given distributions $D_{ref}(t+1), D(t+1)$ and placement $L(t)$ the algorithm computes the placement at period $t+1$, $L(t+1)$. The algorithm determines whether demands $D(t+1)$ and $D_{ref}(t+1)$ are weakly ϵ-near. If they are – it conducts a "minimal" reposition by running SCC over placement $L(t)$ (with its associated flow), where the reposition cost threshold is set to r_{min}. Otherwise, it conducts a "maximal possible" reposition by running SCC over placement $L(t)$ with the reposition cost threshold set to r_{max} and then resets $\tau = t$.

Note that if we set $\epsilon = 0$, Hybrid is equivalent to SCC with r_{max}. If we set $\epsilon = \infty$, Hybrid is equivalent to SCC with r_{min}.

The complexity of Hybrid in the worst-case scenario is equal to the complexity of SCC that runs for r_{max} iterations, i.e., $O(r_{max}mk(k+m)^2)$ where the parameter m is the number of resource types and k is the number of regions.

10.6 Performance Evaluation of the Dynamic Algorithms

Remark 10.9. We evaluate the performance of SCO and compare it the performance of the parametrized placement solution in the next chapter (Section 11.4). The performance evaluation of this section includes only the performance of SCC and Hybrid, which are easer to implement.

In this section, we evaluate the performance of the dynamic algorithms presented in this chapter; for the sake of providing a scale of reference we compare them with the Proportional Mean placement – a replication strategy proposed in [11]. We simulate a mobile game application that requires real-time service by servers located on a geographically distributed cloud (e.g., Amazon EC2 servers). For the sake of a addressing a realistic case we use the price structure of Amazon EC2([80]).

[9] In [1], we present Hybrid over SCO.

10.6.1 Parameter Settings

We refer to our model (Chapter 3) for a full explanation of our simulation parameters. To achieve good performance the mobile application provider aims to receive service at the cloud servers located in proximity to its clients (users). To this end, it places the servers in $k = 3$ regions, located in the USA, Europe and Asia. The application provider offers two different applications ($m = 2$), each requiring a different type of platform from the service provider (e.g., either a Windows platform or a Red Hat Enterprise Linux (Linux) platform in the cloud). We assume that both types of cloud platforms can serve up to $B_{Windows} = B_{Linux} = 500$ users.

Revenue constants parameters	R_i^{loc}	R_i^{glo}
Windows	\$ 0.5	\$ 2
Linux	\$ 0.1	\$ 1.9

TABLE 10.1: Service costs used in simulations

On demand costs (p_i^j)	USA	Europe	Asia
Windows	\$ 0.14	\$ 0.133	\$ 0.161
Linux	\$ 0.137	\$ 0.137	\$ 0.158

TABLE 10.2: Amazon EC2 price system

The revenue constants R_i, R_i^j (see Eqs. (3.6),(3.7)) are, respectively, associated with locally serving a user requesting type i platform in region j and serving a user requesting type i platform across all regions. We assume that the revenue of locally serving a type i user is uniform across all regions. We denote this local constant by $R_i^{loc} := R_i^j$, for every platform type $i \in \{Windows, Linux\}$ and region $j \in \{USA, Europe, Asia\}$. We denote, for the sake of presentation, the other (global) service constant as $R_i^{glo} := R_i$. These revenue constants R_i^{glo}, R_i^{loc} can be defined as the Average Revenue Per User (ARPU) from granting a user request. The ARPU may vary across mobile applications as reported in [81]. The full details assumed for the revenue constants can be found in Table 10.1, where we use values similar to those reported in [81].

Owing to the cloud provider limitation on on-demand servers, the application provider cannot rent more than 20 servers per region, (similar to the limitations of Amazon EC2 [80]), i.e., we set the region costs to be $C^{USA}(x) = C^{Europe}(x) = C^{Asia}(x) = \infty$ for any $x > 20$.

We set the region&type cost functions $C_i^j()$ (See Eq. (3.6)) to be a linear functions, i.e., $C_i^j = p_i^j \cdot x$ where p_i^j are the prices of renting an on-demand instance of resource i in region j. In Table 10.2, we provide the on-demand costs p_i^j. The costs are based on the 2014 Amazon EC2 price system on the m3.medium VM instances [80], on Windows

and Linux platforms, over three regions: North Carolina (USA), Ireland (Europe) and Singapore (Asia). When using Amazon EC2 servers, there are no region or type costs and thus we can set the region and type cost functions to zero (See Eq. (10.3), Eq. (10.4)), i.e., $C^{USA}(x) = C^{Europe}(x) = C^{Asia}(x) = 0$ for every $x \leq 20$, and $C_{Windows}(x) = C_{Linux}(x) = 0$ for every x.

We set the total number of requests for resource type i in region j on time t, $D_i^j(t)$, to be a time-dependent Poisson distribution with parameter $\lambda_i^j(t)$ (as in [82] and [83]). Our dynamic scheduling uses an hourly based prediction, where in every region the average number of requests for Windows and Linux servers is the same i.e., $\lambda^j_{Linux}(t) = \lambda^j_{Windows}(t)$. We set the demand parameter to a periodic function that increases during day time and decreases during night time (usually, between 4 pm till 4 am) as in other studies (such as [83]). In addition, in some web applications (such as [8]) the arrival rate is considered to be unpredictable, with a larger variability due to some noise factor. To simulate the arrival rates, we add an Additive White Gaussian Noise to the arrival rate formula. Thus, the arrival rate is

$$\lambda_i^j(t) = 3000 \cdot \sin(\frac{t_j \cdot \pi}{24}) + 300 \cdot \zeta, \tag{10.14}$$

where ζ is a white noise, generated by the standard Gaussian (Normal) distribution, and t_j is the local time in region j.

Finally, given a period t, we compute the local time in the USA by $t_{USA} = (t \bmod 24)$, the local time in Europe by $t_{Europe} = ((t + 6) \bmod 24)$, and in Asia $t_{Asia} = ((t + 12) \bmod 24)$.

10.6.2 Performance of the dynamic algorithms over time-varying predicted demands

We evaluate the reposition cost (the number of repositions made by the algorithm) as well as the deviation of the placement profit (deviation from optimality) under an arrival rate of $\lambda_i^j(t)$ (as given in Eq. (10.14)) over a time range of $t = 0, 1, \ldots, 47$ and compare the following algorithms: 1) The optimal unconstrained placement[10]. 2)The SCC[11] algorithm (Section 10.4) with reposition cost $r = 4$. 4) The Hybrid algorithm (Section 10.5) with thresholds $\epsilon = \$2000$, $r_{min} = 2$, $r_{max} = 4$. 5) A placement strategy called **Proportional Mean**, as proposed in [11], where the number of type i servers

[10] At every time t an optimal placement is conducted over the demand $L(t)$. Recall that while the optimal reposition under reposition constraint is a hard problem (See Section 10.2), the optimal unconstrained placement (reposition constraint set to infinity) is solvable in polynomial time.

[11] In [1], we present these results over SCO, and we explained that the actual algorithm used in the simulation is an approximate implementation of SCO, i.e., SCC.

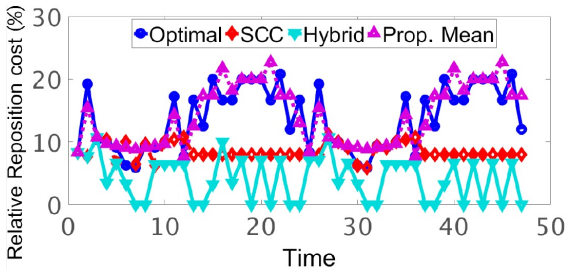


FIGURE 10.7: Relative Reposition Cost

allocated in every region j is proportional to the demand, i.e., there is a constant $\alpha > 0$ such that $L_i^j(t) = \alpha \cdot E(D_i^j(t))$. For our simulations we set $\alpha = 1.2$. For all applicable algorithms above, the placement allocated in time $t = 0$ is the optimal unconstrained placement with respect to demand $D(t) = D(0)$.

In Figure 10.7, we depict the **Relative Reposition Cost (RRC)**. The Relative Reposition Cost is defined as the reposition cost (the number of repositions made by the algorithm) normalized by the number of servers placed. The RRC represents the percentage of servers that were changed.

We observe the following results: 1) The Hybrid algorithm incurs the lowest reposition cost of 10% at most, and sometimes 0% (i.e., no reposition between successive periods). The Hybrid has the lowest average reposition cost of the only 5.7%, while the optimal unconstrained placement and Proportional Mean have 15.8% and 16.8%, respectively. This means that, the average reposition cost of the Hybrid algorithm is 65% lower than that of the other two algorithms. 2) Proportional Mean and the optimal unconstrained placement incur a large reposition cost, ranging between $10\% - 20\%$.3) SCC has (almost) a constant reposition cost, which is always around 10%. Its average reposition cost is 9.8%, which is larger than the Hybrid algorithm by a factor of 1.7.

Next we examine whether the fact that the Hybrid algorithm operates under reposition constraints, allows it to achieve close to optimal placements. To this end, in Figure 10.8 we depict the relative deviations of the placement profit which we called the **Relative Profit Deviation** for all the algorithms examined; the Relative Profit Deviation is defined as the deviation of placement profit normalized by the profit of the optimal unconstrained placement; the Relative Profit Deviation is the difference between the profit of an optimal (unconstrained) placement and the placement profit[12].

[12]Recall that as opposed to the optimal reposition problem under reposition constraint (as studied in this work) that is a hard problem (See Section 10.2), the optimal unconstrained placement is solvable in polynomial time.

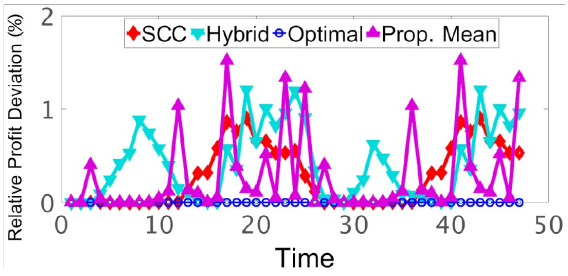


FIGURE 10.8: Relative deviation of the placement profit

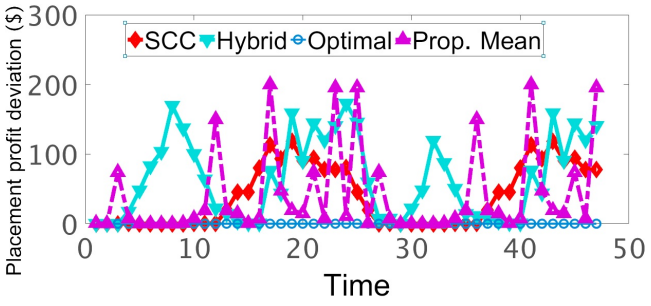


FIGURE 10.9: absolute deviation of the placement profit

We observe that the Relative Profit Deviation of all the dynamic algorithms studied is 2% at most. Note that the Hybrid placement algorithm achieves very efficient placements, whose relative deviation is bounded by 1.3% (i.e., its profit is at least 98.7% of the optimal placement profit), while obeying strict reposition constraints.

For the sake of completeness we depict in Figure 10.9 the absolute (non-relative) profit deviation achieved by the algorithms in these runs. We observe that: 1) SCC has the minimum profit deviation ($130 per hour at most) with low variability. 2) The Hybrid algorithm is better than Proportional Mean (deviations of Hybrid and Proportional Mean are bounded by $200 and $400 respectively). 3) Proportional Mean suffers from high variability compare to the Hybrid algorithm and SCC.

10.6.3 Performance Evaluation – conclusions

We observed that the Hybrid algorithm has the lowest number of repositions used. The average number of repositions it uses is lower by 65% than the number of repositions the optimal unconstrained placement uses, as well as Proportional Mean. The Relative Profit Deviation of the Hybrid algorithm is 1.3% at most, better than Proportional Mean.

Although SCC has Relative Profit Deviation smaller than that of the Hybrid algorithm, it uses more repositions. Proportional mean, as well as the optimal unconstrained placement, incur, as expected, high reposition costs.

10.7 Approximation algorithms - Future work and general discussion

Recall that in Section 10.2 we showed that ConRePP-2 is hard, i.e., there is no optimal solution in polynomial time[13]. Thus, one might consider approximating the problem, i.e., giving a constant-approximation solution for the problem. Unfortunately, we could not come up with any constant approximation algorithm of the problem, which runs in polynomial time.

The problem addressed in this chapter , namely, ConRePP-2, seems to be a non trivial problem, which brings up difficulties either in solving it or in proving hardness. For this reason, the issue of approximate algorithms is left open in this thesis for future work. Specifically the difficulty of ConRePP-2 is concluded from the following observations:

1. Solving our problem, namely ConRePP-2, even when assuming that the system contains only a single resource-type (ConRePP-1, described at Chapter 9), seems to be non-trivial. It took almost 20 pages (some of them were moved to the appendix) to solve ConRePP-1, which is much "easier" than ConRePP-2. Thus, even if there exists an approximation for ConRePP-2, we suspect that the proof of its optimality may be non-trivial.

2. To show hardness of ConRePP-2, we showed that a polynomial solution for ConRePP-2 will derive a polynomial solution for the Exact Perfect Matching Problem. [84] showed a Polynomial-Time Approximation Scheme (PTAS) for the Exact Perfect Matching. Thus, Exact Perfect Matching Problem can be approximated within a factor $1 + \epsilon$ for every $\epsilon > 0$, and we cannot use our reduction or a similar reduction described in Section 10.2 to guarantee a constant approximation hardness for ConRePP-2. We have tried (and we will try in future work) to prove that ConRePP-2 is hard to approximate by using an approximation-preserving reduction.

We conjecture that SCO, which solves optimally ConRePP-1 (Theorem 10.7), approximates the optimal solution of ConRePP-2. In future work, we will try to show whenever or not SCO approximates ConRePP-2.

[13] Polynomial in the reposition constant r, the number of regions k, and the number resource types m.

Chapter 11

Model Extensions

In this chapter, we study two extensions of the model studied in this thesis.

11.1 The Placement Problem with Reposition Costs

A related problem to the Constrained-Reposition Placement Problem (ConRePP-2, Chapter 10) and the general Static Placement Problem (SPP-4, Chapter 7) is whereby repositions are attributed with (linear) costs rather than being considered as a constraint. That is, in this problem one seeks an optimal reposition where each reposition incurs some (fixed) cost and these costs are added to the profit function. The generality of our profit function allows the operator to capture different operational costs, and in particular, it allows one to capture costs associated with the reposition of the resources between different time periods.

Suppose that the operator repositioned the placement $L(t) = \{L_i^j(t)\}$ at the beginning of period t, creating a new placement $L(t+1)$. The cost of adding a new type i resource to region j is equal to $\pi_i^j \geq 0$, while the cost of removing a type i resource from region j is $\theta_i^j \geq 0$. The reposition cost of type i in region j between periods t and $t+1$ is equal to

$$Rep(t)_i^j(L_i^j(t+1)) = \pi_i^j \cdot \underbrace{\max(L(t+1)_i^j - L(t)_i^j, 0)}_{\text{number of type } i \text{ resources added to region } j} + \theta_i^j \cdot \underbrace{\max(L(t)_i^j - L(t+1)_i^j, 0)}_{\text{number of type } i \text{ resources removed from region } j}. \tag{11.1}$$

The Placement Problem with Reposition Costs is formulated as follows: given a placement $L(t)$ of the previous period and the new demand $D(t+1)$, find the placement $L(t+1)$ that maximizes the profit $P(L(t+1), D(t+1))$ minus the reposition costs

i.e.,

$$\max_{L(t+1)} E_{D(t+1)}[P(L(t+1), D(t+1))] - \sum_{i=1}^{m}\sum_{j=1}^{k} Rep(t)_i^j(L_i^j(t+1)). \tag{11.2}$$

Similar to other problems presented in our modeling, we assume that the conditional-marginal functions of P, i.e., $g(n) = \zeta(n, D)$ are concave.

As we next formulate, the Placement Problem with Reposition costs can be reduced to the general Static Placement Problem (SPP-4) (without reposition costs), and it can be solved using the transformation to the min-cost flow problem as done in Chapter 7.

Theorem 11.1. *The Placement Problem with Reposition Costs can be reduced to the general Static Placement Problem (SPP-4).*

Proof. To show the reduction – we define a new profit function P' that is constructed to be equal to the profit P minus the reposition costs C_i^j. Such a profit function should be expressed as we did in the modeling chapter (Chapter 3). The profit function P' has the same model parameters as in P (i.e., the same local and global revenue parameters ($R_i^j > 0$ and $R_i > 0$), the same number of requests a type i resource can serve concurrently B_i etc.), with only one exception: the region+type local cost functions, C'^j_i includes the repositions costs, i.e., is equal to

$$C'^j_i(L_i^j) = C_i^j(L_i^j) + Rep(t)_i^j(L_i^j), \tag{11.3}$$

where $C_i^j(L_i^j)$ is the local cost function of the original profit function P, and $Rep(t)_i^j(L_i^j))$ is the reposition cost.

As we only change the local costs, the new profit function P' is equal to the original profit function P minus the reposition costs for every resource type i and region j, i.e., $\sum_{i=1}^{m}\sum_{j=1}^{k} Rep(t)_i^j(L_i^j))$.

Note that the optimal solution provided for the general Static Placement Problem (SPP-4), as described in Chapter 7, holds only if its conditional-marginal functions are concave. That means, we need to show that the conditional-marginal profit functions of P', denoted by $g'(n) = \zeta'(n, D)$, are concave. Since only the local costs C_i^j have changed, it is left to show that only the region+type marginal profit functions $g'^j_i(n)$ are concave. According to Eq. (11.4), these functions are equal to

$$g'^j_i(n) = \zeta'^j_i(n, D_i^j) = \zeta_i^j(n, D_i^j) - Rep(t)_i^j(n) = g_i^j(n) - Rep(t)_i^j(n). \tag{11.4}$$

where ζ_i^j, g_i^j are respectively the region+type marginal and conditional-marginal functions of the original profit function P, and $Rep(t)_i^j$ are the reposition costs.

Now, note that the negation of reposition cost function $-Rep(t)_i^j(L_i^j)$ can be shown to be a concave function, i.e., the differential of the reposition cost $-Rep(t)_i^j(L_i^j)$ is monotonically decreasing: according to Eq. (11.1), the differential is equal to $Rep(t)_i^j(L_i^j) - Rep(t)_i^j(L_i^j+1) = -\pi_i^j$ for $L_i^j \geq L(t)_i^j$ and $Rep(t)_i^j(L_i^j) - Rep(t)_i^j(L_i^j+1) = \theta_i^j$, otherwise. Since the reposition bounds π_i^j, θ_i^j are non-negative, then the negation of reposition cost functions $-Rep(t)_i^j$ are concave.

The marginal function $\zeta_i^j(L_i^j, D_i^j)$ of P is assumed to be concave. Thus, by Eq. (11.4), the marginal functions of P', $\zeta'^j_i(L_i^j, D_i^j)$ for $i = 1, 2 \ldots, m$ and $j = 1, 2, \ldots k$, are the sum of concave functions, and thus they must be concave.

□

Finally, in Chapter 7 we described an algorithm that solves the general Static Placement Problem (SPP-4) in $O(|L|km(k+m))$, where $|L|$ is the size of the optimal placement, k number of regions, and m number of resource types m.

11.2 The Parameterized Placement Problem – Can it be used to Solve the Constrained-Reposition Placement Problem (ConRePP-2)

The **Parameterized Placement Problem** (which we called in [1] the **Parameterized Unconstrained Problem**), is a special case of the Placement Problem with Reposition Costs, whose solutions can possibly be used to solve the Constrained-Reposition Placement Problem for particular reposition bounds r. Formally, the Parametrized Placement Problem is defined as the Placement Problem (SPP-4), where every reposition (add or subtract) incurs a cost of $\lambda \geq 0$. Given a reposition cost parameter λ, the Parametrized Placement Problem is to find the optimal placement $L_{par}(\lambda)$ with the larger parametrized profit, i.e., solves the following problem (similar to Eq. (11.2)):

$$\max_{L(t+1)} E_{D(t+1)}[P(L(t+1), D(t+1))] - \lambda \cdot \sum_{i=1}^{m} \sum_{j=1}^{k} |L(t+1)_i^j - L(t)_i^j|. \tag{11.5}$$

The Parametrized Placement Problem, can be solved using the same solution for the Placement Problem (SPP-4) with Reposition Costs.

Now, suppose that a solution $L_{par}(\lambda)$ has repositioned r_λ resources. Let L be a placement that repositioned r_λ resources at most. Then the profit $P(L, D(t+1))$ of L cannot be larger than the profit of $L_{par}(\lambda)$ (i.e., $P(L_{par}(\lambda), D(t+1)) \geq P(L, D(t+1))$); otherwise, L has a larger parametrized profit (described in Eq. (11.5)) than $L_{par}(\lambda)$ and $L(\lambda)$ is not an optimal placement for the Parametrized Placement Problem. Thus, we present the following corollary:

Corollary 11.2. *Suppose $L_{par}(\lambda)$ solves the Parametrized Placement Problem with parameter λ, and uses r_λ repositions. Then $L_{par}(\lambda)$ solves the Constrained-Reposition Placement Problem under r_λ repositions.*

Note that this corollary does not contradict the hardness of the Constrained-Reposition Placement Problem (presented in Section 10.2); given a general reposition bound r, it is hard to find a parametrized solution $L_{par}(\lambda)$ which uses exactly $r_\lambda = r$ repositions. However, the corollary may help us in attempting to find the optimal solution for particular constraint r, by solving the Parameterized Placement Problem for a variety of λ values and hoping that one of the resulting r_λ values will be close to r. A binary search over the lambda values could possibly expedite this search by utilizing monotonicity.

This corollary enables us to measure the effectiveness of every heuristic strategy for solving the Constrained-Reposition Placement Problem. Let $L_{par}(\lambda)$ and $L_A(\lambda)$ respectively denote the parametrized solution and the solution returned by an heuristic algorithm A under r_λ repositions. Using Corollary 11.2 we can evaluate the performance of A by exploring a set of reposition parameters λ and comparing for each of them to the profit of its placement $L_A(\lambda)$ with that of the optimal Constrained-Reposition Placement Problem $L_{par}(\lambda)$. This approach is used in Section 11.4, where we measure the effectiveness of *SCO*.

11.3 Modeling Extensions of Unsatisfied Requests

Our general model and placement problems (both the Static Placement Problem and the Constrained-Reposition Placement Problem) can be extended to account for special costs (negative profits) attributed to unsatisfied requests. We show that these problems can be reduced to the corresponding problem where unsatisfied requests do not incur cost, and we can use the same solutions to solve these problems.

Suppose that the (negative) profit of every type i unsatisfied request in region j is $U_i^j < 0$. As explained in the modeling chapter (Chapter 3), the number of type i requests satisfied by some resource in region j is equal to $\min(L_i^j \cdot B_i, D_i^j)$ (B_i – the number of requests a type i resource can serve concurrently). Thus, the number of the unsatisfied type

i requests in region j equals $D_i^j - \min(L_i^j \cdot B_i, D_i^j)$, and the (negative) profit of these requests is equal to $U_i^j \cdot (D_i^j - \min(L_i^j \cdot B_i, D_i^j))$.

To incorporate these unsatisfied requests in the profit function, we add these unsatisfied request costs to the region&type marginal profit function $\zeta_i^j(L_i^j, D_i^j)$. According to Eq. (3.6), this is equal to:

$$\zeta_i^j(L_i^j, D_i^j) = R_i^j \cdot E_{D_i^j}[\min(L_i^j \cdot B_i, D_i^j)] - C_i^j(L_i^j) + U_i^j \cdot E_{D_i^j}[D_i^j - \min(L_i^j \cdot B_i, D_i^j)], \quad (11.6)$$

or alternatively,

$$\zeta_i^j(L_i^j, D_i^j) = (R_i^j - U_i^j) \cdot E_{D_i^j}[\min(L_i^j \cdot B_i, D_i^j)] - C_i^j(L_i^j) + U_i^j \cdot E_{D_i^j}[D_i^j]. \quad (11.7)$$

The constant $U_i^j \cdot E_{D_i^j}[D_i^j]$ does not depend on the placement L. Also $R_i^j - U_i^j > R_i^j \geq 0$ is the (positive) profit of every type i request in region j that is satisfied.

Now let P be the profit function where unsatisfied requests incur costs and have the above model parameters. We will define a new profit function, denoted P', such that: 1) It does not include unsatisfied request costs, and 2) An optimal solution for either the Static Placement Problem or the Constrained Reposition Placement Problem with profit function P, is the same optimal solution for the corresponding problem with profit function P'. In other words, we show that the placement problems where the unsatisfied requests incur costs can be reduced to the same problems where the unsatisfied requests do not incur costs.

We define a profit function P' to have the same model parameters as P (i.e., the same global revenue parameters ($R_i > 0$), the same number of requests a type i resource can serve concurrently B_i etc.), with only two exceptions: 1) it does not include unsatisfied requests costs, and 2) the region+type local revenue parameter of P' is equal to $R'^j_i = R_i^j - U_i^j > 0$. One can verify that the profit P is equal to the profit P' plus a constant $\sum_i \sum_j U_i^j \cdot E_{D_i^j}[D_i^j]$, which does not depend on the placement L. Thus, the optimal solutions for the corresponding placement problems with a profit function P (where unsatisfied requests have impacts) are the same solution as the corresponding problem with profit function P' (where unsatisfied requests do not incur costs).

11.4 Comparing the Performance of SCO to the Parametrized (Unconstrained) Placement Solution

We use real demand traffic, taken from [5], to numerically evaluate SCO and compare it to the optimal solutions of the Parametrized Placement Problem described in Section 11.2. We have shown that if a parametrized solution $L(\lambda)$ uses r repositions then $L(\lambda)$ solves the Constrained-Reposition Placement Problem (ConRePP-2) under r repositions. We are interested in examining how SCO compares to these optimal solutions.

We consider a single-type system ($m = 1$) where the cost and revenue functions are linear and the revenue from serving a local request is 1.5 times larger than that of a remote request. With the lack of real demand data, we use data from [5] that reflects the demand of three data centers. Using that data we consider $k = 3$ regions where the demands at t are $D^1 \sim N(334, 115^2)$ (i.e., Normal demand with (mean, stdvar)= (400,100)), $D^2 \sim N(504, 100^2), D^3 \sim N(186, 55^2)$ and the demands at $t+1$ are shifted cyclically.

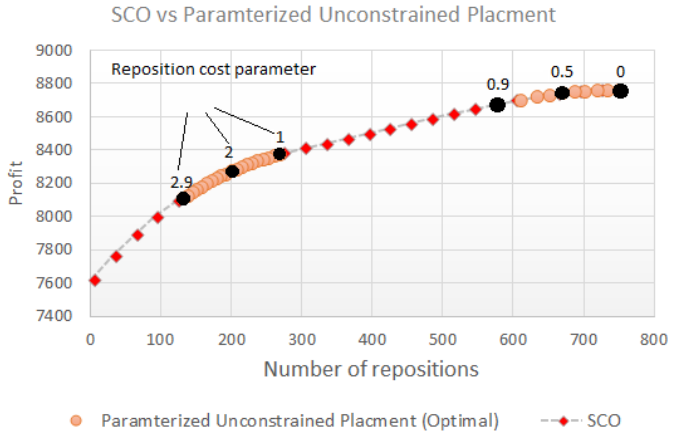


FIGURE 11.1: Performance of SCO compared with the parametrized (unconstrained) placement solutions. Reposition cost parameters values are indicated above the curve.

In Figure. 11.1, we depict the profit of SCO and compare it to the profit of the parametrized solutions. We run SCC for reposition parameters $r = 1 + 30 * ind$ for $0 \leq ind \leq 25$, and depict the profit of parametrized solution $L(\lambda)$ for reposition cost parameter $\lambda = ind{*}0.1$, where $0 \leq ind \leq 29$. We can see that SCC is the optimal strategy, and has similar performance to the parametrized solutions.

We note that the parametrized solutions do not cover all the optimal solutions of the Constrained-Reposition Placement Problem. For example, there is a large gap between the parametrized solutions $L(1)$ and $L(0.9)$. The profit and number of repositions that $L(0.9)$ uses are both significantly larger than those of $L(1)$. Thus, we see the need to

use our SCC algorithm that can provide (nearly-optimal) solutions for the Constrained-Reposition Placement Problem for any value r.

Bibliography

[1] Y. Rochman, H. Levy, and E. Brosh. Dynamic placement of resources in cloud computing and network applications. *Performance Evaluation(PEVA)*, in press. URL http://www.sciencedirect.com/science/article/pii/S0166531616302188.

[2] Amazon EC2. Amazon EC2 home page. http://aws.amazon.com/ec2, 2017.

[3] Microsoft Azure. Microsoft Azure home page. http://www.windowsazure.com, 2017.

[4] Lokad. Lokad – Probabilistic Forecasting. https://www.lokad.com/probabilistic-forecasting, 2017.

[5] Iyswarya Narayanan, Aman Kansal, Anand Sivasubramaniam, Bhuvan Urgaonkar, and Sriram Govindan. Towards a leaner geo-distributed cloud infrastructure. In *Proceedings of HotCloud*, 2014.

[6] V. Valancius, N. Laoutaris, L. Massoulie, C. Diot, and P. Rodriguez. Greening the Internet with Nano Data Centers. In *ACM CoNext*, Rome, Italy, Dec 2009.

[7] M. Cha, H. Kwak, P. Rodriguez, Y. Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *ACM Internet Measurement Conference*, New York, NY, USA, October 2007.

[8] Valeria Cardellini, Emiliano Casalicchio, Francesco Lo Presti, and Luca Silvestri. Sla-aware resource management for application service providers in the cloud. In *NCCA*, pages 20–27, 2011. URL http://dblp.uni-trier.de/db/conf/ncca/ncca2011.html#CardelliniCPS11.

[9] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein. Dynamic service placement in geographically distributed clouds. *IEEE Journal on Selected Areas in Communications*, 99:762–772, 2013.

[10] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C.M. Lau. Scaling Social Media Applications into Geo-Distributed Clouds. In *IEEE INFOCOM*, Orlando, Florida, USA, March 2012.

[11] S. Tewari and L. Kleinrock. Proportional replication in peer-to-peer networks. In *IEEE INFOCOM*, Barcelona, Spain, April 2006.

[12] Y. Rochman, H. Levy, and E. Brosh. Max percentile replication for optimal performance in multi-regional p2p vod systems. In *Proceeding of the 9th International Conference on Quantitative Evaluation of SysTems (QEST) 2012*, London, UK, September 2012.

[13] Y. Rochman, H. Levy, and E. Brosh. Resource placement and assignment in distributed network topologies. In *IEEE INFOCOM*, Turin, Italy, April 2013.

[14] Y. Rochman, H. Levy, and E. Brosh. Efficient resource placement in cloud computing and network applications- technical report. http://www.cs.tau.ac.il/~yuvalroc/technical%20reports/Yuval_effeicnt_resource.pdf, 2014. In preparation.

[15] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The cost of a cloud: Research problems in data center networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, December 2008.

[16] Yan Li, Sanjay Ranka, and Sartaj Sahni. In-advance path reservation for file transfers in e-science applications. *The Journal of Supercomputing*, 59(3):1167–1187, 2012. URL http://dblp.uni-trier.de/db/journals/tjs/tjs59.html#LiRS12.

[17] Hamed Pirsiavash, Deva Ramanan, and Charless C. Fowlkes. Globally-optimal greedy algorithms for tracking a variable number of objects. In *CVPR*, pages 1201–1208. IEEE, 2011.

[18] Y. Rochman, H. Levy, E. Brosh, and G. Gilboa-Freedman. Resource repositioning in distributed clouds - technical report. https://goo.gl/Tjntfw, 2015. Submitted.

[19] Christos H. Papadimitriou and Mihalis Yannakakis. The complexity of restricted spanning tree problems. *J. ACM*, 29(2):285–309, April 1982. ISSN 0004-5411.

[20] Yuster Raphael. Almost exact matchings. *Algorithmica*, 63(1):39–50, 2012.

[21] EgervÃąry Research Group on Combinatorial Optimization (EGRES). Exact matching in red-blue bipartite graphs- egervÃąry research group on combinatorial optimization (egres)., 2016. URL http://lemon.cs.elte.hu/egres/open/Exact_matching_in_red-blue_bipartite_graphs.

[22] Y. Rochman and A. Tamir. On sensitivity of dynamic min cost flows - technical report. https://goo.gl/SWsQ8V, 2016. To be Submitted.

[23] Y. Rochman, H. Levy, and E. Brosh. On dynamic placement of resources in cloud computing- technical report. http://www.cs.tau.ac.il/~yuvalroc/technical%20reports/Yuval_on_dynamic.pdf, 2015. In preparation.

[24] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management*, pages 119–128. IEEE, 2007. URL http://dblp.uni-trier.de/db/conf/im/im2007.html#BobroffKB07.

[25] Yasuhiro Ajiro and Atsuhiro Tanaka. Improving packing algorithms for server consolidation. In *Int. CMG Conference*, pages 399–406. Computer Measurement Group, 2007.

[26] M. Wang, X. Meng, and L. Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *2011 Proceedings IEEE INFOCOM*, pages 71–75, April 2011.

[27] A. Goel and P. Indyk. Stochastic load balancing and related problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 579–586, 1999.

[28] Galia Shabtai, Danny Raz, and Yuval Shavitt. Stochastic service placement. *CoRR*, abs/1503.02413, 2015.

[29] M. R. Garey and D. S. Johnson. " strong " np-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, July 1978.

[30] Danny Raz, Itai Segall, and Maayan Goldstein. Multidimensional resource allocation in practice. In *Proceedings of the 10th ACM International Systems and Storage Conference*, SYSTOR '17, New York, NY, USA, 2017.

[31] H. Xu and B. Li. Joint Request Mapping and Response Routing for Geo-distributed Cloud Services,. In *IEEE INFOCOM*, Turin, Italy, April 2013.

[32] Mansoor Alicherry and T. V. Lakshman. Network aware resource allocation in distributed clouds. In *INFOCOM*, pages 963–971. IEEE, 2012.

[33] Zhen Xiao, Weijia Song, and Qi Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE Trans. Parallel Distrib. Syst.*, 24(6):1107–1117, June 2013.

[34] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Gener. Comput. Syst.*, 28(5):755–768, May 2012.

[35] Anton Beloglazov and Rajkumar Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGrid 2010, 17-20 May 2010, Melbourne, Victoria, Australia*, pages 577–578, 2010.

[36] Toni Mastelic, Ariel Oleksiak, Holger Claussen, Ivona Brandic, Jean-Marc Pierson, and Athanasios V. Vasilakos. Cloud computing: Survey on energy efficiency. *ACM Comput. Surv.*, 47(2):33:1–33:36, December 2014. ISSN 0360-0300.

[37] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transaction On Networking*, 20(1):206–219, February 2012. ISSN 1063-6692.

[38] Long Gong, Yonggang Wen, Zuqing Zhu, and Tony Lee. Toward profit-seeking virtual network embedding algorithm via global resource capacity. In *INFOCOM IEEE Conference on Computer Communications*, pages 1–9, 2014.

[39] Yong Zhu and Mostafa H. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM*. IEEE, 2006.

[40] N. M. Mosharaf Kabir Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM. 28th IEEE International Conference on Computer Communications*, pages 783–791, 2009.

[41] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *SIGCOMM Comput. Commun. Rev.*, 38(2):17–29, March 2008.

[42] Amit Chakrabarti, Chandra Chekuri, Anupam Gupta, and Amit Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47(1):53–78, Jan 2007.

[43] Sivadon Chaisiri, Bu-Sung Lee, and Dusit Niyato. Optimal virtual machine placement across multiple cloud providers. In *APSCC*, 2009.

[44] Sivadon Chaisiri, Rakpong Kaewpuang, Bu-Sung Lee, and Dusit Niyato. Cost minimization for provisioning virtual servers in amazon elastic compute cloud. In *MASCOTS*, 2011.

[45] Lei Yu, Liuhua Chen, Zhipeng Cai, Haiying Shen, Yi Liang, and Yi Pan. Stochastic load balancing for virtual resource management in datacenters. In *IEEE Transactions On Cloud Computing*, 2016.

[46] Siva Theja Maguluri, R. Srikant, and Lei Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. In *Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012*, pages 702–710, 2012.

[47] Dimitris Hatzopoulos, Iordanis Koutsopoulos, George Koutitas, and Ward Van Heddeghem. Dynamic virtual machine allocation in cloud server facility systems with renewable energy sources. In *Communications (ICC), 2013 IEEE International Conference on*, pages 4217–4221. IEEE, 2013.

[48] Siva Theja Maguluri and R. Srikant. Scheduling jobs with unknown duration in clouds. *IEEE/ACM Trans. Netw.*, 22(6):1938–1951, December 2014.

[49] E. Sherzer, H. Levy, and G. Gilboa-Freedman. Plan your cloud on a rainy day, 2017. Submitted.

[50] Y. P. Zhou, T. Z. J. Fu, and D. M. Chiu. Statistical modeling and analysis of p2p replication to support vod service. In *IEEE INFOCOM*, Orlando, FL , USA, July 2011.

[51] Yan Chen, Randy H Katz, and John D Kubiatowicz. Dynamic replica placement for scalable content delivery. In *International Workshop on Peer-to-Peer Systems*, pages 306–318. Springer, 2002.

[52] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan. Optimal content placement for a large-scale vod system. In *ACM CoNEXT*, Philadelphia, USA, Dec 2010.

[53] J. Kangasharju, K. W. Ross, and D. A. Turner. Optimizing file availability in peer-to-peer content distribution. In *IEEE INFOCOM*, Anchorage, Alaska , USA, May 2007.

[54] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks. In *ACM SIGCOMM*, pages 177–190, 2002.

[55] Z. Drezner and H. W. Hamacher. *Facility Location: Applications and Theory*. Springer, 2002.

[56] Susan Hesse Owen and Mark S. Daskin. Strategic facility location: A review. *European Journal of Operational Research*, 111(3):423 – 447, 1998.

[57] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12(3):450–459, 1964.

[58] Charles Revelle. The maximum capture or âĂIJsphere of influenceâĂİ location problem: Hotelling revisited on a network*. *Journal of Regional Science*, 26(2): 343–358, 1986.

[59] Lawrence V. Snyder. Facility location under uncertainty: a review. *IIE Transactions*, 38(7):547–564, 2006.

[60] Yale T. Herer, Michal Tzur, and Enver YÃijcesan. The multilocation transshipment problem. *IIE Transactions*, 38(3):185–200, 2006.

[61] Deniz Özdemir, Enver Yücesan, and Yale T. Herer. Multi-location transshipment problem with capacitated production and lost sales. In *Proceedings of the 38th Conference on Winter Simulation*, WSC '06, pages 1470–1476. Winter Simulation Conference, 2006.

[62] Deniz Özdemir, Enver Yücesan, and Yale T. Herer. Multi-location transshipment problem with capacitated production. *European Journal of Operational Research*, 226(3):425–435, 2013.

[63] Deniz Özdemir, Enver Yücesan, and Yale T. Herer. Sample average approximation approach to multi-location transshipment problem with capacitated production. In *Winter Simulation Conference*, pages 2384–2394, 2009.

[64] William C. Jordan and Stephen C. Graves. Principles on the benefits of manufacturing process flexibility. *Management Science*, 41(4):577–594, 1995.

[65] Tal Raviv, Michal Tzur, and Iris A. Forma. Static repositioning in a bike-sharing system: models and solution approaches. *EURO Journal on Transportation and Logistics*, 2(3):187–229, 2013.

[66] Daniel Freund, Shane G. Henderson, and David B. Shmoys. Minimizing multimodular functions and allocating capacity in bike-sharing systems. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 186–198, 2017.

[67] Wikipedia. Expected value: Discrete distribution taking only non-negative integer value, 2017. URL http://en.wikipedia.org/wiki/Expected_value#Discrete_distribution_taking_only_non-negative_integer_values.

[68] B. Tan and L. Massoulié. Optimal content placement for peer-to-peer video-on-demand systems. In *IEEE INFOCOM*, Orlando, FL , USA, July 2011.

[69] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube traffic characterization: a view from the edge. In *ACM Internet Measurement Conference*, pages 15–28, New York, NY, USA, 2007. ACM.

[70] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[71] R. G. Busacker and P. J. Gowen. A procedure for determining a family of minimal cost network flow patterns. Oro technical report 15, Operational Research Office, Johns Hopkins University, Baltimore, MD, September 1961.

[72] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[73] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *ACM 19*, pages 248–264, 1972.

[74] S. Tewari and L. Kleinrock. On Fairness, Optimal Download Performance and Proportional Replication in Peer-to-Peer Networks. In *IFIP/TC6 Networking*, Waterloo, Ontario, Canada, May 2005.

[75] Dorit S. Hochbaum. Lower and upper bounds for the allocation problem and other nonlinear optimization problems. *Math. Oper. Res.*, 19(2):390–409, 1994.

[76] Dorit S. Hochbaum and J. George Shanthikumar. Convex separable optimization is not much harder than linear optimization. *J. ACM*, 37(4):843–862, October 1990.

[77] Eric Weisstein. Kolmogorov-smirnov test. http://mathworld.wolfram.com/Kolmogorov-SmirnovTest.html, 2017.

[78] Achim Klenke, Johannes Gutenberg universitaÌĹt Mainz, and Lutz Mattner. Stochastic ordering of classical discrete distributions, 2010.

[79] Wikipedia. Multiset:example multiset operations, 2017. URL https://en.wikipedia.org/wiki/Multiset#Example_multiset_operations.

[80] Amazon Pricing. Amazon EC2 pricing page. http://aws.amazon.com/ec2/pricing/, 2014.

[81] Mobile ARPU. Think Gaming: top paid games. http://thinkgaming.com/app-sales-data/top-paid-games/, 2014.

[82] Xiaoming Nan, Yifeng He, and Ling Guan. Optimal allocation of virtual machines for cloud-based multimedia applications. In *MMSP*, pages 175–180. IEEE, 2012.

[83] Boxun Zhang, Gunnar Kreitz, Marcus Isaksson, Javier Ubillos, Guido Urdaneta, Johan A. Pouwelse, and Dick H. J. Epema. Understanding user behavior in spotify. In *INFOCOM*, pages 220–224. IEEE, 2013.

[84] André Berger, Vincenzo Bonifaci, Fabrizio Grandoni, and Guido Schäfer. Budgeted matching and budgeted matroid intersection via the gasoline puzzle. In *Integer Programming and Combinatorial Optimization, 13th International Conference, IPCO 2008, Bertinoro, Italy, May 26-28, 2008, Proceedings*, pages 273–287, 2008.

Appendices

Appendix a

Chapter 5 – Proof of Claims

a.1 Proof of Claim 5.3

Claim 5.3. *Let L be a balanced placement, whose quantity vector is $\hat{L} = (L_1, L_2, \ldots L_m)$. Then, the following properties hold:*

1. *In every region j either $\left\lfloor \frac{L_i}{k} \right\rfloor$ or $\left\lceil \frac{L_i}{k} \right\rceil$ resources of resource type i are allocated for $1 \leq i \leq m$ and $1 \leq j \leq k$.*
2. *The number of regions with $\left\lfloor \frac{L_i}{k} \right\rfloor + 1$ resources is equal to $L_i \bmod k$.*
3. *The profit $P(L, D)$ is equal to:*

$$\sum_{i=1}^{m}[R_{glo}E(\min(L_i, D_i)) + R_{loc}\sum_{j=1}^{r_i} E(\min(\left\lceil \frac{L_i}{k} \right\rceil, D_i^j)) + R_{loc}\sum_{j=r_i+1}^{k} E(\min(\left\lfloor \frac{L_i}{k} \right\rfloor, D_i^j))], \tag{a.1}$$

where $r_i = L_i \bmod k$.

Proof of Claim 5.3. 1. Let us order the regions by the number of resources containing the i^{th} resource type, i.e., $L_i^1 \geq L_i^2 \geq \ldots \geq L_i^k$. Since $\sum_{j=1}^{k} L_i^j = L_i$ then by the pigeonhole principle we see that $L_i^1 \geq \left\lceil \frac{L_i}{k} \right\rceil$ and $L_i^k \leq \left\lfloor \frac{L_i}{k} \right\rfloor$. Let us examine two cases:

(a) If L_i is not divided by k, then we have $\left\lfloor \frac{L_i}{k} \right\rfloor + 1 = \left\lceil \frac{L_i}{k} \right\rceil$. Hence if $L_i^1 > \left\lceil \frac{L_i}{k} \right\rceil$ or $L_i^k < \left\lfloor \frac{L_i}{k} \right\rfloor$ then $L_i^1 - L_i^k \geq 2$. The placement L is balanced, and thus $L_i^1 = \left\lceil \frac{L_i}{k} \right\rceil$ and $L_i^k = \left\lfloor \frac{L_i}{k} \right\rfloor$. Therefore, for every region j we have $\left\lceil \frac{L_i}{k} \right\rceil = L_i^1 \geq L_i^j \geq L_i^k = \left\lfloor \frac{L_i}{k} \right\rfloor = \left\lceil \frac{L_i}{k} \right\rceil - 1$, i.e., L_i^j must be or $\left\lfloor \frac{L_i}{k} \right\rfloor$ or $\left\lceil \frac{L_i}{k} \right\rceil$.

(b) If L_i is divided by k, then we assume, by way of contradiction that $L_i^1 > \left\lfloor \frac{L_i}{k} \right\rfloor = \frac{L_i}{k}$. Therefore $L_i^k \geq \frac{L_i}{k}$, since L is balanced. But then $L_i = \sum_{j=1}^{k} L_i^j \geq L_i^1 + (k-1) \cdot L_i^k > \frac{L_i}{k} + (k-1) \cdot \frac{L_i}{k} = L_i$ – a contradiction. Therefore, $L_i^1 = \frac{L_i}{k}$. In a similar way, we can prove that $L_i^k = \frac{L_i}{k}$, and thus, for every $1 \leq j \leq k$ we have $L_i^j = \frac{L_i}{k}$.

2. For every j, let us define $L_i^j = \left\lfloor \frac{L_i}{k} \right\rfloor + \gamma_j$ when $\gamma_j \in \{0, 1\}$. Then

$$L_i = \sum_{j=1}^{k} L_i^j = k \cdot \left\lfloor \frac{L_i}{k} \right\rfloor + \sum_{j=1}^{k} \gamma_j.$$

Therefore $L_i - k \cdot \left\lfloor \frac{L_i}{k} \right\rfloor = \sum_{j=1}^{k} \gamma_j$. By definition of the modulus operation $L_i - k \cdot \left\lfloor \frac{L_i}{k} \right\rfloor$ is equal to $L_i \, mod \, k$. The expression $\sum_{j=1}^{k} \gamma_j$ is equal to the number of regions which have $\left\lfloor \frac{L_i}{k} \right\rfloor + 1$ resources. Thus, the number of regions with $\left\lfloor \frac{L_i}{k} \right\rfloor + 1$ resources is equal to $L_i \, mod \, k$, as required.

3. If L_i is divided by k, then every region has exactly $\frac{L_i}{k}$ resources and the claim is clear. If L_i is not divided by k then $\left\lfloor \frac{L_i}{k} \right\rfloor + 1 = \left\lceil \frac{L_i}{k} \right\rceil$ and therefore, from the second part of this claim, the number of regions that have $\left\lceil \frac{L_i}{k} \right\rceil$ resources is r. The rest of the regions have $\left\lfloor \frac{L_i}{k} \right\rfloor$ resources. Thus, Eq. (a.1) holds.

□

a.2 Proof of Claim 5.4

Claim 5.4. *Given a quantity vector $\hat{L} = (L_1, L_2, \ldots L_m)$, the following properties hold:*

1. *There exists a balanced placement of quantity vector $\hat{L}$.*
2. *Every two balanced placements with the same quantity vector $\hat{L}$, have the same profit.*

Proof of Claim 5.4. 1. We denote $r_i = L_i \, mod \, k$. Let L be the placement defined as followed:

$$L_i^j = \begin{cases} \left\lfloor \frac{L_i}{k} \right\rfloor + 1 & 1 \leq j \leq r_i \\ \left\lfloor \frac{L_i}{k} \right\rfloor & k \geq j > r_i \end{cases}.$$

By its definition, L is balanced. The number resources of type i is equal to $\left\lfloor \frac{L_i}{k} \right\rfloor \cdot k + r_i = L_i$, and therefore, the quantity vector of L is $\hat{L}$.

2. Let L and L' be two balanced placement with the same quantity vector denoted by $\hat{L}$. By Claim 5.3, L and L' have the same profit.

□

a.3 Proof of Lemma 5.8

Lemma 5.8. *The profit of L_B (as defined in Lemma 5.7), $P(L_B, D)$, is not smaller than that of L, $P(L, D)$. That is $P(L_B, D) \geq P(L, D)$.*

Proof of Lemma 5.8. To prove this lemma we want to show that for every unbalanced placement L there exists a quantity-equivalent placement with a profit that is higher than (or equal to) the profit L. To this end, we will make use of a potential function which is defined as follows: Let $L = \{L_i^j | 1 \leq i \leq m, 1 \leq j \leq k\}$ be an arbitrary placement. The *Potential function* f is defined as $f(L) = \sum_{i=1}^{m} \sum_{j=1}^{k} \sum_{p=1}^{L_i^j} p$.

Using this potential function we will show that for every unbalanced placement L there exists a quantity-equivalent placement L' such that:

1. $f(L) > f(L')$.
2. $P(L', D) \geq P(L, D)$.

We use the fact that L is not balanced, and thus there exists a resource type i and two regions j_1 and j_2, such that $L_i^{j_1} \geq L_i^{j_2} + 2$. We define placement L' to be the placement resulting from moving a resource of type i from region j_1 to region j_2. Of course, L' is quantity-equivalent to L.

Next we show that the first property holds: The removal of resource type i from region j_1 decreases the potential $f()$ by $L_i^{j_1}$, and the addition of resource type i to region j_2 increases the potential by $L_i^{j_2} + 1$. Therefore, we have $f(L') = f(L) - L_i^{j_1} + L_i^{j_2} + 1$. As $L_i^{j_1} \geq L_i^{j_2} + 2 > L_i^{j_2} + 1$ we derive $f(L') < f(L)$.

Claim 5.6 derives that the profit of L' is larger than (or equal to) the profit of L, $P(L', D) \geq P(L, D)$. Hence, the second property holds.

From both properties 1 and 2, we derive that there exists a chain $L_0 = L, L_1, \ldots$ of placements that are quantity-equivalent to L such that 1) $f(L_i) > f(L_{i-1})$ and 2) $P(L_i, D) \geq P(L_{i-1}, D)$ for every $i \geq 1$. Since the number of placements that are quantity-equivalent to L is finite, there is a placement in the equivalence class of L that must be a balanced placement, and whose profit is not smaller than that of L. Since

every two quantity-equivalent balanced placements have the same profit (as shown in Claim 5.4), the symmetrically-full balanced placement of L, L_B (derived in Lemma 5.7), which is quantity-equivalent to L, has a profit which is higher than (or equal) to that of L and the lemma follows.

□

Remark a.1. One may question whether the potential technique used in the proof of Lemma 5.8 is necessary and if the use of Claim 5.6 by itself is not sufficient to carry out the proof. To this end we believe that the use of the potential technique is needed in order to avoid "infinite cycles" of $\geq$ improvements.

a.4 Proof of Claim 5.9

Claim 5.9. *Given placement L, inserting a resource of type i_0 to region j_0, will increase the profit by:*

$$R_{glo} \cdot \Pr(D_{i_0} \geq L_{i_0} + 1) + R_{loc} \Pr(D_{i_0}^{j_0} \geq L_{i_0}^{j_0} + 1).$$

Proof of Claim 5.9. By Eq. (5.1), the profit of a placement is:

$$R_{glo} \sum_{i=1}^{m} E(\min(L_i, D_i)) + R_{loc} \sum_{i=1}^{m} \sum_{j=1}^{k} E(\min(L_i^j, D_i^j)).$$

Therefore, adding a resource of type i_0 to region j_0 increases L_{i_0} and $L_{i_0}^{j_0}$ by one. The profit of the new placement L' is equal to:

$$P(L', D) = R_{glo}[E(\min(L_{i_0} + 1, D_{i_0})) - E(\min(L_{i_0}, D_{i_0}))] + R_{loc}[E(\min(L_{i_0}^{j_0} + 1, D_{i_0}^{j_0})) - E(\min(L_{i_0}^{j_0}, D_{i_0}^{j_0}))].$$

Hence, by Theorem 3.1, we derive that the profit of the placement is increased by

$$P(L', D) - P(L, D) = R_{glo} \Pr(D_{i_0} \geq L_{i_0} + 1) + R_{loc} \Pr(D_{i_0}^{j_0} \geq L_{i_0}^{j_0} + 1),$$

as required.

□

a.5 Proof of Claim 5.10

Claim 5.10. *Let $\hat{L} = (L_1, L_2, \dots L_m)$ be a quantity vector and let $\hat{L}'$ be a quantity vector such that $L'_{i_0} = L_{i_0} + 1$ and $L'_j = L_j$ for all $j \neq i_0$. Then the increase of the balanced profit, i.e., $P_B(\hat{L}') - P_B(\hat{L})$, is*

$$R_{glo} \cdot \Pr(D_{i_0} \geq L_{i_0} + 1) + R_{loc} \Pr(\tilde{D}_{i_0} \geq \left\lfloor \frac{L_{i_0}}{k} \right\rfloor + 1). \tag{a.2}$$

Proof of Claim 5.10. Let L_B be a balanced placement of $\hat{L}$. By Claim 5.3 there is a region j_0 containing $L_{i_0}^{j_0} = \left\lfloor \frac{L_{i_0}}{k} \right\rfloor$ resource of type i_0. Let L'_B be the placement implied by adding a resource of type i_0 to region j_0. The placement L'_B has the following properties:

1. L'_B is balanced: Follows immediately as L_B is balanced and from the selection of the added resource.
2. The quantity vector of L'_B is $\hat{L}'$: The number of type i resources has been incremented by one.
3. The expected profit difference between L'_B and L_B is given by Eq. (a.2): Using the demand symmetry (see Section 5.1) this follows from Claim 5.9, where $D_{i_0}^{j_0}$ is replaced by $\tilde{D}_{i_0}$.

Therefore, the increase of the balanced profit will be as in Eq. (5.8). □

Appendix b

Chapter 6 – Proof of Claims

b.1 Proof of Theorem 6.1

Theorem 6.1. *Let L be a feasible placement that minimizes $E(C^L)$ among feasible placements with s resources (i.e., $|L| = s$). Then L is an optimal placement (i.e., solves SPP-3).*

Proof of Theorem 6.1. In our proof, we use Eq. (3.9) to imply that:

$$E(\min(L_i^j, D_i^j)) = \sum_{n=1}^{L_i^j} \Pr(D_i^j \geq n), \tag{b.1}$$

$$E(\min(L_i, D_i)) = \sum_{n=1}^{L_i} \Pr(D_i \geq n). \tag{b.2}$$

In the first step of the proof, we will show that there is an optimal feasible placement (solves SPP-3) that contains s resources, i.e., in every region j the placement contains s^j resources (s^j is the total storage value). By Section 6.1, an optimal placement L_{opt} should maximize the profit $P(L, D)$ among all placements with the feasibility constraint, i.e., the number of resources in area j is not larger than s^j ($L^j \leq s^j$).

Suppose there exists an area j_0 such that L_{opt} does not contain s^{j_0} resources (i.e., $L_{opt}^{j_0} < s^{j_0}$). By Eqs. (b.1), (b.2), we derive that adding $s^{j_0} - L_{opt}^{j_0}$ resources in region j_0 cannot decrease the expected values of $E(\min(L_i^j, D_i^j))$ and $E(\min(L_i, D_i))$. Hence, by Eq. (6.1), we derive that there is a feasible optimal placement that contains s resources.

In the second step, we prove that every feasible placement L with s resources has an alternative cost of $E(C^L) = s(R_{glo} + R_{loc}) - P(L, D)$. The number of resources in L is

equal to $s = \sum_{i=1}^{m}\sum_{j=1}^{k} L_i^j = \sum_{i=1}^{m} L_i$ and therefore using Eq. (b.1), we derive that:

$$R_{glo}\sum_{i=1}^{m}\sum_{n=1}^{L_i}(1-\Pr(D_i \geq n)) = \\ R_{glo}\sum_{i=1}^{m} L_i - R_{loc}\sum_{i=1}^{m}\sum_{n=1}^{L_i}\Pr(D_i \geq n)) = \\ = R_{glo}s - R_{glo}\sum_{i=1}^{m}\min(D_i, L_i). \tag{b.3}$$

In similar way we prove:

$$R_{loc}s - R_{loc}\sum_{i=1}^{m}\sum_{j=1}^{k}\min(D_i^j, L_i^j) = R_{loc}\sum_{i=1}^{m}\sum_{j=1}^{k}\sum_{n=1}^{L_i}(1-\Pr(D_i \geq n)). \tag{b.4}$$

Eqs. (b.4), (b.3) yield that $E(C^L) = s(R_{glo} + R_{loc}) - P(L, D)$.

Finally, let L' be a feasible placement minimizing the alternative cost $E(C^L)$ among all feasible placements with s resources. The profit of L' is equal to $P(L', D) = s(R_{glo} + R_{loc}) - E(C^L)$. The expression $s(R_{glo} + R_{loc})$ is constant and thus L' maximizes the profit among all feasible placements with s resources. Since there is a feasible optimal placement (for the placement problem, SPP-3) that contains s resources, then placement L' is also a feasible optimal placement for the placement problem (SPP-3).

□

b.2 Proof of Lemma 6.2

Lemma 6.3. *Let L be a feasible placement with n resources. Then there exists an integer flow f_L (i.e., $f_L(e)$ is integer for all edges $e \in E$) called the* ***corresponding flow of*** *L with flow value $|f_L| = n$ that obeys the following conditions: (1) ${f_L}^{in}(a^j, t^i) = L_i^j$, ${f_L}^{in}(t^i) = L_i$ for all area j and resource type i. (2) The flow f_L sends 1 unit of flow on the edges between layer-3 $((a^j, t_i))$ and layer-4 $((a^j, t_i, r))$ of costs $R_{loc}(1 - \Pr(D_i^j \geq r))$ for $1 \leq r \leq L_i^j$. (3) The flow f_L sends 1 unit of flow on the edges between layer-5 (t_i) and layer-6 $((t_i, r))$ of costs $R_{glo}(1 - \Pr(D_i \geq r))$ for $1 \leq r \leq L_i$. Moreover, the cost of f_L is equal to the alternative cost of L, i.e., $w(f_L) = E(C^L)$.*

Proof of Lemma 6.2. To show the existence of a flow corresponding to L, we define the values of $f_L(e)$ for every edge e as follows: 1) The flow between layer-1 node (source) x and layer-2 node (area) a^j is equal to the number of resources in the j^{th} region L^j. 2) The flow between layer-2 nodes a^j and 3-layer nodes (a^j, t^i) is L_i^j. 3) The flow between layer-3 nodes (a^j, t^i) and layer-4 nodes (a^j, t^i, r) equals 1 iff $r \leq L_i^j$ and otherwise 0. 4) The flow between layer-4 nodes (a^j, t^i, r) and layer-5 (resource type) nodes t^i is 1 iff $r \leq L_i^j$, and otherwise 0. 5) The flow between resource type nodes t^i and 6-layer nodes (t^i, r) is 1 iff $r \leq L_i$, and otherwise 0. 6) The flow between 6-layer nodes (t^i, r) and the sink node y is 1 iff $r \leq L_i$, and otherwise 0. One may verify that f_L satisfies the capacity constraint and conservation of flows properties, conditions (1) and (2), and its flow value is equal to n.

To prove that the cost of f_L is equal to $E(C^L)$, we can see that the weight of f_L equals $R_{loc} \sum_{n=1}^{L_i^j}(1-\Pr(D_i^j \geq n)) + R_{glo} \sum_{n=1}^{L_i}(1-\Pr(D_i \geq n))$, which is exactly the alternative cost of L, $E(C^L)$. □

b.3 Proof of Lemma 6.3

Lemma 6.3. *Let f be an integer flow in G^7. Let L be a placement such that $L_i^j = f^{in}(a^j, t^i) = f_i^j$. Then the cost of f is not smaller than the cost of a corresponding flow of L, i.e., $w(f) \geq w(f_L)$. Moreover, $w(f) = w(f_L)$ iff f is a flow corresponding to L.*

Proof of Lemma 6.3. Given a vertex v in a given network graph G, a flow r on $G = (V, E)$, the outcome weight of v with respect to r, denoted as $w_r^{out}(v)$, is the flow cost outcomes from this vertex i.e.,

$$w_r^{out}(v) \doteq \sum_{(v,u)\in E} w(v,u) r(v,u). \tag{b.5}$$

If there is no edge originating from v, then we define $w_r^{out}(v) = 0$. One may check that given a flow g on general graph $G = (V, E)$ the weight of g equals $w(g) = \sum_{v\in V} w_g^{out}(v)$. To prove that the weight of f_L is not larger than the weight of f, we show that for every vertex v its outcome weight with respect to f_L is not larger than its outcome weight with respect to f, i.e., $w_f^{out}(v) \geq w_{f_L}^{out}(v)$. Note that only 3-layer nodes $v = (a^j, t^i)$ and 5-layer nodes $v = (t^i)$ have non-zero weights. Thus, it is left to show the equation for 3-layer and 5-layer nodes.

For a 3-layer node $v = (a^j, t^i)$, the outcome weight of $w_f^{out}(v)$ is equal to the sum of f_i^j elements from the set T_i^j, where

$$T_i^j = \{R_{loc}(1 - \Pr(D_i^j \geq 1)), R_{loc}(1 - \Pr(D_i^j \geq 2)), \ldots, R_{loc}(1 - \Pr(D_i^j \geq s))\}. \quad \text{(b.6)}$$

The f_i^j minimal elements from T_i^j are $R_{loc}(1-\Pr(D_i^j \geq 1)), R_{loc}(1-\Pr(D_i^j \geq 2)), \ldots, R_{loc}(1-\Pr(D_i^j \geq f_i^j))$, and the sum of these elements is equal to $w_{f_L}^{out}(v)$. Thus the outcome weight $w_{f_L}^{out}(v)$ cannot be larger than $w_f^{out}(v)$. Thus we have shown that $w_f^{out}(v) \geq w_{f_L}^{out}(v)$ for every vertex $v \in V_3$. In a similar way, we prove that every 5-layer node $v \in V_5$ satisfies $w_f^{out}(v) \geq w_{f_L}^{out}(v)$.

Now, suppose that the cost of f is equal to the cost of f_L. Since there is no vertex v whose outcome weight with respect to f_L is larger than its outcome weight with respect to f, then we derive that $w_f^{out}(v) = w_{f_L}^{out}(v)$ for every 3-layer and 5-layer vertex.

Let $v = (a^j, t^i)$ be a 3-layer node. Since $w_f^{out}(v) = w_{f_L}^{out}(v)$ then $w_f^{out}(v)$ is equal to the sum of the f_i^j minimal elements from the set T_i^j. In addition, f sends flow through edges with costs $R_{loc}(1-\Pr(D_i^j \geq 1)), R_{loc}(1-\Pr(D_i^j \geq 2)), \ldots, R_{loc}(1-\Pr(D_i^j \geq f_i^j)) = R_{loc}(1 - \Pr(D_i^j \geq L_i^j))$. In a similar way, we show that f sends flow through edges with costs $R_{glo}(1 - \Pr(D_i \geq 1)), R_{glo}(1 - \Pr(D_i \geq 2)), \ldots, R_{glo}(1 - \Pr(D_i \geq L_i))$. Thus f is a flow corresponding to L.

□

b.4 Proof of Corollary 6.4

Corollary 6.4. *Let f_{opt} be an integer min-cost flow of G^7. Let L be the placement created from the values of the flow in (region, type) nodes $L_i^j = f_{opt}^{in}(a^j, t^i)$. Then the flow f_{opt} is corresponding to L.*

Proof of Corollary 6.4. If f_{opt} is not a flow corresponding to L, then, according to Lemma 6.3, the cost of f_L is strictly smaller than the cost of f (i.e., $w(f) > w(f_L)$). However, the flow value of f_{opt} is equal to the flow value of f. Thus, f_{opt} cannot be a min-cost flow – a contradiction. □

b.5 Proof of Corollary 6.5

Corollary 6.5. *Let f_{opt} be an integer min-cost flow of G^7 with flow value $|f_{opt}| = s$. Let L be the placement of the flow in (region, type) nodes $L_i^j = f_{opt}^{in}(a^j, t^i)$. Then L solves the placement problem.*

Proof of Corollary 6.4. The integer min-cost flow f_{opt} has a required flow of s. Thus, the flow between source x and an area node a^j should be equal to the edge capacity, i.e., s^j. Therefore, the placement L contains $L^j = \sum_{i=1}^{m} L_i^j = \sum_{i=1}^{m} f_i^j = s^j$ resources in region j. The placement L is a feasible placement that contains s resources. According to Corollary 6.4, $f_{opt} = f_L$ is a flow corresponding to L.

Let $L' = \{L'^j_i\}$ be a feasible placement of s resources that minimizes the alternative cost $E(C^{L'})$. Using Lemma 6.3, we derive that the cost corresponding flow of L and L' are respectively equal to the alternative cost of L and L', i.e., $w(f_L) = E(C^L)$ and $w(f_{L'}) = E(C^{L'})$. The flow f_L is a min-cost flow (of flow value s), and therefore $E(C^L) = w(f_L) \leq w(f_{L'}) = E(C^{L'})$. The placement L' minimizes the alternative cost $E(C^{L'})$ – thus, L also minimizes the alternative cost. Finally, by Theorem 6.1 we derive that L solves the placement problem (SPP-3). □

b.6 Proof of Lemma 6.8

Lemma 6.8. *Let f be a flow that the SSP algorithm calculates in some iteration over the residual graph G_f^7. Then, there is a shortest path between x and y, denoted by p_{opt}, which has a decomposition formula as follows:*

$$p_{opt} = x \overset{\min}{\to} a^{j_1} \overset{\min}{\to} t^{i_1} \overset{\min}{\to} a^{j_2} \ldots \overset{\min}{\to} a^{j_e} \overset{\min}{\to} t^{i_e} \overset{\min}{\to} y, \tag{b.7}$$

where x, y are respectively the sink and source of G_f^7, and a^{j_l}, t^{i_l} are, respectively, area and resource type nodes, for all $1 \leq l \leq e$.

Proof of Lemma 6.8. First, we use the following two claims that are proved in the literature: 1) [73] proved that in each iteration of the SSP the resulted graph does not contain a negative cycle. 2) [72] proved that if a graph $G = (V, E)$ does not contain negative cycles, then between two vertices $a, b \in V$ there exists a shortest path which does not contain cycles (namely, the path is *simple*). By 1) and 2) we obtain that there exists a simple shortest path p' between source x and sink y in G_f^7 (i.e., if $p' = (v_0 = x, v_1, v_2 \ldots, v_r, v_{r+1} = y)$, then for all $i_1 < i_2$ we have $v_{i_1} \neq v_{i_2}$). Note that

the length of every simple path does not exceed the number of vertices $|V'|$ in G_f^7, and thus the length of p' is finite.

We will prove that path p' satisfies the following conditions:

(a) Path p' begins with a source-to-area path.
(b) If an area node $v = a^j$ is in p', then v begins an area-to-type path in p'.
(c) If a resource type node $v = t^i$ is in p', then it begins a type-to-area or type-to-sink path in p'.

If we show that p' satisfies these conditions, then the monotone paths in p' are minimal, otherwise p' is not the shortest path between x and y. Since the sink node y appears once in p' (as the path p' is simple) and the length of p is finite, this shows that p' can be split into special minimal paths $x \to a^1 \to t^1 \to \ldots \to t^e \to y$, where $v^i \to v^j$ are minimal paths between v^i and v^j. We define $p_{opt} = p_{opt} = x \overset{\min}{\to} a^{j_1} \overset{\min}{\to} t^{i_1} \overset{\min}{\to} a^{j_2} \ldots \overset{\min}{\to} a^{j_e} \overset{\min}{\to} t^{i_e} \overset{\min}{\to} y$. Note that p_{opt} might be different from p' if there is more than one minimal path between two nodes (See Remark 6.3), although the weight of p_{opt} and p' are equal to each other and thus p_{opt} is a minimal path.

The proof that p' satisfies condition (a) follows immediately. To show that p' satisfies condition (b), let a^j be an area node in p'. Since the edges in the residual graph G_f^7 are either original or reverse edges from G^7, then the neighbors of a^j in G_f^7 are layer-3 nodes (a^j, t^i) and the layer-1 source node x. If the next node in p' after a^j is the source node x, then path p' contains x at least twice, when its first appearance is in the beginning of the path. Thus, in such case, p' contains a cycle, which contradicts the definition of p'. Therefore, the next node in p' after a^j must be a layer-2 node (a^j, t^i).

The neighbors of (a^j, t^i) in G_f^7 are layer-4 nodes (a^j, t^i, r) or the layer-2 node a^j. If the next node in p' after (a^j, t^i) is a^j, then p' contains a^j twice, which contradicts the path simplicity. In a similar way, it can be proved that the node after (a^j, t^i, r) in p' is t^i (otherwise (a^j, t^i) appears twice). Finally, we show that a^j begins an area-to-type path in p' and therefore p' satisfies condition (b).

Let t^i be a resource type node. Then the node after t^i in p' denoted by v can be either a layer-6 node (t^i, r) or a layer-4 node (a^j, t^i, r). If v is a layer-6 node $v = (t^i, r)$, then t^i begins a type-so-sink path, otherwise t^i appears twice in p'. If v is a layer-4 node $v = (a^j, t^i, r)$, then t^i begins a type-to-area path from t^i to a^j; otherwise, p' is not a simple path. Thus we show p' satisfies condition (c), and the lemma is completed. □

b.7 Proof of Corollary 6.9

Corollary 6.9. *Let f be a flow that SSP calculates in some iteration over the residual graph G_f^7. Then $p_{opt} = x \overset{\min}{\to} a^{j_1} \overset{\min}{\to} t^{i_1} \overset{\min}{\to} a^{j_2} \ldots \overset{\min}{\to} a^{j_e} \overset{\min}{\to} t^{i_e} \overset{\min}{\to} y$ is a shortest path in G_f^7 iff $\hat{p}_{opt} = (x, a^{j_1}, t^{i_1}, a^{j_2} \ldots, t^{i_e}, y)$ is a shortest path in G_f^B. Moreover, p_{opt} and $\hat{p}_{opt}$ have the same weight.*

Proof of Corollary 6.9. Given a graph G and a path p in G, we denote by $w(p, G)$ the weight of p in G.
$\Rightarrow$: Let $\hat{p'}$ be a shortest path in G_f^B between x and y, corresponding to a path p' in G_f^7. The path p_{opt} is the one with the minimum weight in G_f^7. Thus, $w(p_{opt}, G_f^7) \leq w(p', G_f^7)$. The weight of every path p in the seven-layer residual graph G_f^7 is equal to the weight of its corresponding path $\hat{p}$ in the bipartite-like graph G_f^B. Therefore, the weight of $\hat{p'}$ is not larger than the weight of $\hat{p}_{opt}$ in the bipartite-like graph G_f^B, i.e., $w(\hat{p}_{opt}, G_f^B) \leq w(\hat{p'}, G_f^B)$. Since $\hat{p'}$ is a shortest path in G_f^B, then $\hat{p}_{opt}$ is the shortest path in G_f^B as required.

$\Leftarrow$: Let p' be a shortest path in the seven-layer residual graph G_f^7 between x and y, corresponding to a path $\hat{p'}$ in G_f^B. The path $\hat{p}_{opt}$ is the one with the minimum weight in G_f^7. Thus, $w(\hat{p}_{opt}, G_f^B) \leq w(\hat{p'}, G_f^B)$. The weight of every path p in the seven-layer residual graph G_f^7 is equal to the weight of their corresponding path $\hat{p}$ in the bipartite-like graph G_f^B. Therefore, the weight of p' is not larger than the weight of p_{opt} in the seven-layer residual graph G_f^B, i.e., $w(p_{opt}, G_f^7) \leq w(p', G_f^7$. Since p' is a shortest path in G_f^7, then p_{opt} is the shortest path in G_f^B as required.

The paths p_{opt} and $\hat{p}_{opt}$ have the same weight by the definition of p_{opt}.

□

b.8 Proof of Lemma 6.10

Lemma 6.10. *Let f be a flow in G^7 corresponding to a placement L. The minimal path weights in G_f^7 are computed as follows:*
(a) $w_f(x \overset{\min}{\to} a^j) = 0$ *if* $L^j < s^j$ *and otherwise* ∞.
(b) $w_f(a^j \overset{\min}{\to} t^i) = R_{loc}(1 - \Pr(D_i^j \geq L_i^j + 1))$ *if* $L_i^j < s$ *and otherwise* ∞.
(c) $w_f(t^j \overset{\min}{\to} a^i) = -R_{loc}(1 - \Pr(D_i^j \geq L_i^j))$ *if* $L_i^j > 0$ *and otherwise* ∞.
(d) $w_f(t^i \overset{\min}{\to} y) = R_{loc}(1 - \Pr(D_i \geq L_i + 1))$ *if* $L_i < s$ *and otherwise* ∞.

Proof of Lemma 6.10. There is a single source-to-area path between x and a^j, which is the edge connecting the nodes. If $f(s, a^j) = f^j < s^j = c(s, a^j)$, then the weight of the

edge $w_f(s, a^j)$ is equal to the original edge weight in G^7 i.e., $w_f(s, a^j) = w(s, a^j) = 0$. If $f(s, a^j) = f^j < s^j = c(s, a^j)$ then $w_f(s, a^j) = \infty$ by the definition of the residual graph G_f. Thus part (a) of lemma is proved.

To show part (b) of the lemma, let a^j and t^i respectively denote area and resource type nodes. All edges in an area-to-type path between a^j and t^i except the edges between (a^j, t^i) and (a^j, t^i, r) have capacity ∞ and weight 0. Thus, the weight of a minimal area-to-type path $w_{f_L}(a^j \overset{\min}{\rightarrow} t^i)$ is equal to the minimal edge weight between layer-3 (i.e., (a^j, t^i)) and layer-4 (i.e., (a^j, t^i, r)) vertices in G^7_f.

Let e be an edge of minimum weight between layer-3 and layer-4 vertices. If e has a cost of $R_{loc}(1 - \Pr(D^j_i \geq r))$ for $1 \leq r \leq L^j_i$, then f flows through e. Thus, the residual capacity of e equals $c_f(e) = c(e) - f(e) = 1 - 1 = 0$, and therefore its weight is equal to infinity ($w_f(e) = \infty$).

If e has a cost of $R_{loc}(1-\Pr(D^j_i \geq r))$ for $L^j_i+1 \leq r \leq s$ then f does not send flow through e. Thus the residual capacity of e is equal to $c_f(e) = c(e)-f(e) = 1-0 = 1$, and therefore its weight in G_f is equal to the one in G^7, i.e., $w_f(e) = w(e)$. Hence, $w_f(e)$ is equal to either $R_{loc}(1 - \Pr(D^j_i \geq L^j_i + 1)), R_{loc}(1 - \Pr(D^j_i \geq L^j_i + 2)), \ldots, R_{loc}(1 - \Pr(D^j_i \geq s))$. Among these edges, the one with the minimum cost must be $R_{loc}(1 - \Pr(D^j_i \geq L^j_i + 1))$. Thus, the weight of the minimal area-to-type path $w_{f_L}(a^j \overset{\min}{\rightarrow} t^i)$ is equal to $R_{loc}(1 - \Pr(D^j_i \geq L^j_i + 1))$. In the special case where $L^j_i = s$, all edges between a^j and t_i have zero residual capacity, and infinite weight in G_f. Thus, in such case, $w_{f_L}(a^j \overset{\min}{\rightarrow} t^i) = \infty$.

To prove part (c), let a^j and t^i respectively denote area and resource type nodes. All edges in a type-to-area path between t^i and a^j in the residual graph G^7_f are reverse edges. If f does not send flow through the path $a^j - (a^j, t^i) - (a^j, t^i, r) - t^i$ then the reverse edges corresponding to the path (i.e., $(a^j, t^i) - a^j$, $(a^j, t^i, r) - (a^j, t^i)$ and $t^i - (a^j, t^i, r)$) have zero residual capacity, and thus they have an infinite weight. If f sends flow along the path $a^j - (a^j, t^i) - (a^j, t^i, r) - t^i$ then the reverse edges corresponding to the path have non-zero residual capacity, and the weight of the path is equal to the weight of the edge connecting (a^j, t^i) and (a^j, t^i, r).

The flow f sends flow through edges with costs $R_{loc}(1 - \Pr(D^j_i \geq 1)), R_{loc}(1 - \Pr(D^j_i \geq 2)), \ldots, R_{loc}(1 - \Pr(D^j_i \geq L^j_i))$. The cost of their respective reverse edges in the residual graph is equal to $-R_{loc}(1 - \Pr(D^j_i \geq 1)), -R_{loc}(1 - \Pr(D^j_i \geq 2)), \ldots, -R_{loc}(1 - \Pr(D^j_i \geq L^j_i))$. Among these edges, the edge with the minimum weight is equal to $-R_{loc}(1 - \Pr(D^j_i \geq L^j_i))$. Thus, the weight of the minimal type-to-area path $w_{f_L}(t^i \overset{\min}{\rightarrow} a^j)$ is equal to $-R_{loc}(1 - \Pr(D^j_i \geq L^j_i))$. In the special case where $L^j_i = 0$, all reverse edges between a^j and t_i have zero residual capacity, and infinite weight in G_f. Thus, in such case, $w_{f_L}(t^i \overset{\min}{\rightarrow} a^j) = \infty$.

The proof of (d) is similar to the proof of (b).

□

b.9 Proof of Lemma 6.11

Lemma 6.11. *Let f be a flow corresponding to L that SSP computes in some iteration. Suppose that SSP augments the flow by the shortest path (found in Step 2 of SSP) $p_{opt} = x \overset{\min}{\rightarrow} a^{j_1} \overset{\min}{\rightarrow} t^{i_1} \overset{\min}{\rightarrow} a^{j_2} \dots \overset{\min}{\rightarrow} a^{j_e} \overset{\min}{\rightarrow} t^{i_e} \overset{\min}{\rightarrow} y$. Then, SSP updates the flow f to another flow f' that is corresponding to a placement L', where L' is defined as follows: 1) $L'^{j_l}_{i_l} \leftarrow L^{j_l}_{i_l} + 1$ for $1 \leq l \leq e$. 2) $L'^{j_{l+1}}_{i_l} \leftarrow L^{j_{l+1}}_{i_l} - 1$ for $1 \leq l \leq e-1$. 3) $L'^{j}_{i} \leftarrow L^{j}_{i}$ for $(i,j) \neq (i_l, j_l), (i_l, j_{l+1})$.*

Proof of Lemma 6.11. Since the capacity of every edge between (a^j, t^i) and (a^j, t^i, r) is equal to 1 unit, then we augment the flow by only 1 unit of flow in each iteration of SSP. The edges in every minimal area-to-type path (i.e., $a^{j_l} \overset{\min}{\rightarrow} t^{i_l}$) were from the original edges of G^7, and therefore augmenting through p_{opt} increases the flow of node (a^{j_l}, t^{i_l}) by 1 unit of flow, i.e., $f'^{j_{l+1}}_{i_l} \leftarrow f^{j_{l+1}}_{i_l} + 1$. In contrast, type-to-area paths (i.e., $t^{i_l} \overset{\min}{\rightarrow} a^{j_{l+1}}$) are concatenation of reverse edges, and therefore augmenting through p_{opt} cancels the flow of these reverse edges, and thus $f'^{j_{l+1}}_{i_l} \leftarrow f^{j_{l+1}}_{i_l} - 1$ for $1 \leq l \leq e-1$. Since the new updated flow f' is corresponding to a some placement L' (See Corollary 6.7), then $L'^{j}_{i} = f^{j}_{i}$. Therefore, the lemma is followed. □

Appendix c

Chapter 7 – Proof of Claims

c.1 Proof of Theorem 7.4

Theorem 7.4. *Let G be an acyclic graph with arbitrary edge weights w. In the i^{th} iteration of SSP-NE, the flow f_i obtained at the end of the i^{th} iteration satisfies the reduced weight optimality conditions with respect to node potential π_i. Thus, f_i is a min-cost flow.*

Proof of Theorem 7.4. We will prove the claim by induction on the iteration number i. In the initial step ($i = 0$) of the SSP algorithm, all original edges (v_1, v_2) in the residual graph G_f have reduced weights that are equal to $w_f^\pi(v_1, v_2) = w(v_1, v_2) + d(v_1) - d(v_2)$. The shortest path between the source x and v_2 is not strictly longer than the shortest path between the source x and v_1 concatenated with the edge (v_1, v_2), i.e., $w(v_1, v_2) + d(v_1) \geq d(v_2)$. Therefore, this implies that $w_f^\pi(v_1, v_2) \geq 0$. Every reverse edge e in the residual graph G_f has zero residual capacity (as the initial flow f is set to zero for every edge), and thus has an infinite reduced weight (i.e., $w_f^\pi(v_1, v_2) = \infty$). We have shown that in the initial step of the SSP-NE, π satisfies the non-negative reduced weight condition.

Denote the node potentials function in the i^{th} iteration as π_i. [70] indicated in Lemmas 9.11 and 9.12 that if π_{i-1} satisfies the non-negative reduced weight condition, then so does π_i. Thus, f_i satisfies the reduced weight optimality conditions and thus it must be a min-cost flow, as required. □

c.2 Proof of Corollary 7.6

Corollary 7.6. *Let f_i be the flow that SSP-NE finds at the end of the l^{th} iteration. Let p_l be the shortest path in the l^{th} iteration of SSP-NE. Suppose that the shortest path p_{l_0} is the first non-negative path found by SSP-NE, i.e., $w_{f_{l_0-1}}(p_{l_0-1}) < 0 \leq w_{f_{l_0}}(p_{l_0})$. Then, f_{l_0} has minimum weight over all possible min-cost flows (i.e., $f_{l_0} = \arg\min_{f \text{ is min-cost flow}} w(f)$). Moreover, among min-cost flows with minimum weight, the flow value of f_{i_0} is minimal. Thus, the placement L associated with the flow f_{l_0} (i.e., $L_i^j = f_{l_0}^{in}(a^j, t_i)$) solves the placement problem.*

Proof of Corollary 7.6. As SSP-NE augments the cost of flow by the weight of path (i.e., $w(f_l) = w_{f_{l-1}}(p_{l-1}) + w(f_{l-1})$), we derive that $w(f_l) < w(f_{l-1})$ for $l \leq l_0$ and $w(f_l) \geq w(f_{l-1})$ for $l \geq l_0 + 1$. Hence, f_{l_0} has minimum weight over all possible min-cost flows (i.e., $f_{l_0} = \arg\min_{\text{f is min-cost flow}} w(f)$), and among min-cost flows with minimum weight, the flow value of f_{l_0} is minimal. Finally, the optimality of SSP-NE (Theorem 7.4) implies that the placement L associated with the flow f_{l_0} (i.e., $L_i^j = f_{l_0}^{in}(a^j, t_i)$) solves the placement problem. □

c.3 Proof of Corollary 7.7

Corollary 7.7. *G-BG returns the optimal solution L that solves the placement problem (SPP-4).*

Proof of Corollary 7.7. As proven in Appendix e, G-BG imitates the behavior of SSP-NE over the eight-layer graph G^8. By Corollary 7.6 we show that SSP-NE finds the min-cost flow among all feasible flows f over G^8. By Corollary 7.2 this shows that the retrieved flow f_L is corresponding to a placement L, where L solves the placement problem (SPP-4). □

Appendix d

Correctness of Reduction – The General Static Placement Problem

To show the correctness of the reduction, we use the following lemmas, which are similar to Lemmas 6.2, 6.3:

Lemma d.1. *Let L be a feasible placement with n resources. Then there exists an integer flow f_L (i.e., $f_L(e)$ is integer for all edges $e \in E$) called the **corresponding flow of** L with flow value $|f_L| = n$ that obeys the following conditions: (1) ${f_L}^{in}(a^j, t^i) = L_i^j$, ${f_L}^{in}(t^i) = L_i$ for all area j and resource type i. (2) The flow f_L sends 1 unit of flow on the edges between the source x and layer-2 nodes $((a^j, r))$ with costs $-\Delta g^j(r)$ for $1 \leq r \leq L_i$. (3) The flow f_L sends 1 unit of flow on the edges between layer-4 $((a^j, t_i))$ and layer-5 $((a^j, t_i, r))$ of costs $-\Delta g_i^j(r)$ for $1 \leq r \leq L_i^j$. (4) f_L sends 1 unit of flow through edges between layer-6 (t_i) and layer-7 $((t_i, r))$ of costs $-\Delta g_i(r)$ for $1 \leq r \leq L_i$. (5) f_L sends 1 unit of flow through edges between the source x and layer-2 nodes $((a^j, r))$ of costs $-\Delta g_i(r)$ for $1 \leq r \leq L_i$. Moreover, the cost of f_L equals the constant c minus the profit of L, i.e., $w(f_L) = c - P(L, D)$.*

Lemma d.2. *Let f be an integer flow in G^8. Let L be a placement such that $L_i^j = f^{in}(a^j, t^i) = f_i^j$. Then, the cost of f is not smaller than the cost of a corresponding flow of L, i.e., $w(f) \geq w(f_L)$. Moreover, $w(f) = w(f_L)$ iff f is a flow corresponding to L.*

Proof of Lemmas d.1. For the sake of presentation, alternatively, a path $(v_1, v_2, v_3 \dots v_n)$ can be denoted as $v_1 - v_2 \dots v_n$. To show existence, we define the corresponding flow f_L as follows:

- The disjoint paths $x-(a^j,n)-a^j$, $(a^j,t_i)-(a^j,t_i,n)-t_i$, $t_i-(t_i,n)-y$ are assigned respectively with 1 unit of flow if $L^j \geq n$, $L_i^j \geq n$ and $L_i \geq n$; otherwise, the corresponding path is assigned with zero units of flow.
- The flow between (a^j,t_i), and (a^j,t_i) is L_i^j.

One can verify that 1) f preserves the capacity constraint and conservation of flows, 2) the flow value of f_L is equal to $|f_L| = n$. In addition, according to Eq. (7.6), the weight of f_L is equal to the constant c minus the profit of L, i.e., $w(f_L) = c - P(L,D)$ □

Proof of Lemmas d.2. Given a vertex v in a given network graph G, and a flow r on $G=(V,E)$ we define its **outcome weight of** v with respect to r, as the flow cost outcomes from this vertex, i.e.,

$$w_r^{out}(v) \doteq \sum_{(v,u)\in E} w(v,u)r(v,u). \tag{d.1}$$

If there is no edge originating from v, then we define $w_r^{out}(v)=0$. One may check that given a flow g on general graph $G=(V,E)$ the weight of g equals $w(g)=\sum_{v\in V} w_g^{out}(v)$.

To prove that the weight of f_L is not larger than the weight of f, we show that for every vertex v, its outcome weight with respect to f_L is not larger than its outcome weight with respect to f, i.e., $w_f^{out}(v) \geq w_{f_L}^{out}(v)$. Note that only the source x, four-layer nodes $v=(a^j,t^i)$ and 6-layer nodes $v=(t^i)$ have non-zero outcome weights. Thus, it is left to show the equation for these nodes.

For a four-layer node $v=(a^j,t^i)$, the outcome weight of $w_f^{out}(v)$ is equal to the sum of L_i^j elements from the set T_i^j, where

$$T_i^j = \{-\Delta g_i^j(1), -\Delta g_i^j(2), \ldots\}. \tag{d.2}$$

The L_i^j minimal elements from T_i^j are $-\Delta g_i^j(1), -\Delta g_i^j(2), \ldots, -\Delta g_i^j(L_i^j)$. Also, the sum of these elements is equal to $w_{f_L}^{out}(v)$. Since $w_{f_L}^{out}(v)$ is equal to the sum of the minimal L_i^j in T_i^j, they cannot be larger than $w_f^{out}(v)$.

In a similar way, we prove that every 6-layer node v satisfies $w_f^{out}(v) \geq w_{f_L}^{out}(v)$.

To prove that $w_f^{out}(v) \geq w_{f_L}^{out}(v)$ for the source node $v=x$ we use a different technique. We define $w_f(a^j)$ as the **weight value of region** j to be equal to the sum of edge weights connecting the source x to the 2-layer node (a^j,n) that f passes by, i.e., $w_f(a^j) = \sum_{n=1}^{\infty} w_f(e^j(n))f(e^j(n))$, where $e^j(n)$ is the edge connecting the source x and the 2-layer node (a^j,n). Note that the outcome weight of x is equal to the sum of the weight values

of region of region j, i.e., $w_f^{out}(x) = \sum_{j=1}^{n} w_f(a^j)$. The weight value $w_f(a^j)$ is equal to the sum of f^j elements from the set T^j, where

$$T^j = \{-\Delta g^j(1), -\Delta g^j(2), \ldots\}. \tag{d.3}$$

The L^j minimal elements from T^j are $-\Delta g^j(1), -\Delta g^j(2), \ldots, -\Delta g^j(L^j)$. Also, the sum of these elements is equal to $w_{f_L}(a^j)$. Since $w_{f_L}(a^j)$ is equal to the sum of the minimal L^j in T^j, they cannot be larger than $w_f(a^j)$, i.e., $w_f(a^j) \geq w_{f_L}(a^j)$. Thus the outcome weight of x with respect to f_l is not larger than the outcome weight of x with respect to f_L, i.e., $w_f^{out}(x) \geq w_{f_L}^{out}(x)$.

Now, suppose that the weight of f equals the weight of f_L. Since $w_f(v) \geq w_{f_L}(v)$ for every node v, we derive that $w_f(v) = w_{f_L}(v)$ for every node v. Now, let $v = (a^j, t_i)$ be a 3-layer node. Then $w_f(v)$ is equal to the sum of minimal elements from T_i^j. Thus f must send flow through edges between four-layer nodes $v = (a^j, t^i)$ and 5-layer nodes $v = (a^j, t^i, r)$ with costs $-\Delta g_i^j(1), -\Delta g_i^j(2), \ldots, -\Delta g_i^j(L_i^j)$. In a similar way, we can show that f sends flow through edges between the source x and 2-layer nodes $v = (a^j, r)$ with costs $-\Delta g^j(1), -\Delta g^j(2), \ldots, -\Delta g^j(L^j)$, and that f flow through edges between 6-layer nodes t_i and seven-layer nodes $v = (t_i, r)$ with costs $-\Delta g_i(1), -\Delta g_i(2), \ldots, -\Delta g_i(L_i)$. Thus f is a flow corresponding to L. □

d.1 Proof of Corollaries 7.1, 7.2

Finally, we show the proof of Corollaries 7.1, 7.2 as follows:

Corollary 7.1. *Let f_{opt} be an integer min-cost flow of G^8. Let L be the placement created from the values of the flow in (region, type) nodes $L_i^j = f_{opt}^{in}(a^j, t^i)$. Then f_{opt} is corresponding to L.*

Proof of Corollary 6.4. If f_{opt} is not a flow corresponding to L, then according to Lemma d.2 the cost of f_L is strictly smaller than the cost of f (i.e., $w(f) > w(f_L)$). However, the flow value of f_{opt} is equal to the flow value of f. Thus, f_{opt} cannot be a min-cost flow – a contradiction. □

Corollary 7.2. *Suppose that n_{opt} is the minimum number of resources an optimal solution can contain. Let f_{opt} be the min-cost flow in G^8 with a required flow of $|f| = n_{opt}$. Let L be a placement whose quantities are equal to the flow in nodes (a^j, t_i), i.e., $L_i^j = f_{opt}^{in}(a^i, t^j)$. Then L solves the placement problem (SPP-4).*

Proof of Corollary 7.2. The integer min-cost flow f_{opt} has a required flow of n_{opt}. Therefore the placement L_i^j contains $|L| = \sum_{i=1}^{m} L_i^j = \sum_{i=1}^{m} f_i^j = n_{opt}$ resources. According to Corollary 6.4, $f_{opt} = f_L$ is a flow corresponding to L.

Let $L' = \{L'^j_i\}$ be a feasible placement of n_{opt} resources that maximizes the profit and solves the placement problem (SPP-4). Using Lemma 6.3, we derive that the cost of the corresponding flow of L and of L are respectively equal to a constant minus the profit cost of L and L', i.e., $w(f_L) = c - P(L, D)$ and $w(f_{L'}) = c - P(L', D)$. The flow f_L is a min-cost flow (of flow value n_{opt}), and therefore $P(L, D) = c - w(f_L) \geq c - w(f_{L'}) = P(L', D)$. The placement L' maximizes the profit, and thus, L maximizes the profit as well, i.e., L solves the placement problem (SPP-4). □

Appendix e

The Bipartite-like Graph for the SSP-NE Algorithm

e.1 Formal Definition of Monotone Paths

The weight of the edges in the bipartite-like graph is equal to the weight of their respective minimal paths in the residual graph G_f^8. Formally, this is defined as follows:

Definition e.1. Let f be a flow, and let v_i and v_j respectively represent a node in layer-i and a node in layer-j of G_f^8, for $1 \leq i \neq j \leq 8$. An edge $e = (v_i, v_j) \in G_f^8$ is called a **forward edge** iff $i < j$. If $e = (v_i, v_j)$ is not a forward edge, then it is called a **backward edge**. A path p in G_f^8 is called a **forward path** iff all the edges composing the path are forward edges. Similarly, a path p in G_f^8 is called a **backward path** iff all the edges composing the path are backward edges. A path p is **monotone** if it is a forward or a backward path. The shortest monotone path between u and v in G_f^8 is called the **minimal path** between u and v, and is denoted by $u \overset{\min}{\rightarrow} v$.

Remark e.1. If there are multiple monotone paths between v and u, then the minimal path $u \overset{\min}{\rightarrow} v$ is selected to be one of them.

e.2 Path Composition

Similar to Lemma 6.8 and Corollary 6.9, we show that the weight of every shortest path in G_f^B is equal to the weight of its respective path in G_f^8, as follows:

Lemma e.1. *Let f be a flow that the SSP algorithm calculates in its j^{th} iteration over the eight-layer residual graph G_f^8. Let v_0, v_n be two vertices in G_f^8 that are also included*

in the bipartite-like graph G_f^B (i.e., v and v' are either the source node, the sink node, an area node or a type node). Then, there is a shortest path between v_0 and v_n, denoted by p_{opt}, which has a decomposition formula as follows:

$$p_{opt} = v_0 \overset{\min}{\rightarrow} v_1 \ldots \overset{\min}{\rightarrow} v_{n-1} \overset{\min}{\rightarrow} v_n, \tag{e.1}$$

where v_l for $l = 1, 2, \ldots n$ are vertices in G_f^8 that are also included in the bipartite-like graph G_f^B, and (v_i, v_{i+1}) are edges in the bipartite-like graph G_f^B.

Corollary e.2. *Let v_n be a vertex in G_f^8 that is also included in the bipartite-like graph G_f^B, and let x be the source node. Suppose that f is a flow that the SSP-NE algorithm calculates in its j^{th} iteration over the eight-layer residual graph G_f^8. Then $p_{opt} = x \overset{\min}{\rightarrow} v_1 \overset{\min}{\rightarrow} \ldots \overset{\min}{\rightarrow} v_n$ is a shortest path in G_f^8 iff $\hat{p}_{opt} = (x, v_1, v_2, \ldots, v_n)$ is a shortest path in the bipartite-like graph G_f^B. Moreover, p_{opt} and $\hat{p}_{opt}$ have the same weight, and the distance between x and v_n in eight-layer G_f^8 is equal to the distance between x and v_n in the bipartite-like graph G_f^B.*

Proof of Lemma e.1. This proof is similar to the one shown in Lemma 6.8.

Let $p_{opt} = (v_0, v_1, u_2, \ldots, v_n)$ be a shortest path in the eight-layer graph G_f^8. By Theorem 6.6, all edges in G_f^8 have non-negative reduced weights. Thus, as shown by [70], Property 2.5(a), the graph does not contain negative cycles. WLOG the optimal path p_{opt} does not contain cycles, otherwise the non-negative cycles can be omitted from p_{opt}, minimizing the cost of the path.

We will show that every node $v_{l_1} \neq v_n$ in the path p_{opt} is followed by a minimal path $(v_{l_1}, v_{l_1+1}, \ldots, v_{l_2})$, $l_2 > l_1$, where (v_{l_1}, v_{l_2}) is an edge on the bipartite-like graph G_f^B. This will show that p_{opt} can be split to minimal paths, and the proof of the lemma is completed.

Let $v_{l_1} \neq v_n$ in the path p_{opt} and in the bipartite-like graph G_f^B. Based on the structure of v_{l_1}, and the fact that p_{opt} does not contain cycles, we will show that v_{l_1} is followed by a corresponding minimal path:

If v_{l_1} is the source x – then the next minimal path can be either the forward path sink (x, y), where y is the sink, or the forward path $x - (a^j, n) - a^j$ where a^j is an area node.

If v_{l_1} is an area node a^j – then the next minimal path can be either the forward path $a^j - (a^j, t_i) - (a^j, t_i, n) - t_i$, where t_i is a resource type node, or the backward path

$a^j - (a^j, n) - x$ where x is the source node.

If v_{l_1} is a type node t_i – then the next minimal path can be either the forward path $t_i - (t_i, n) - y$, where y is the sink, or the backward path $t_i - (a^j, t_i, n) - (a^j, t_i) - a^j$ where a^j is an area node.

If v_{l_1} is the sink y – then the next minimal path can be either the backward path (x, y), where x is the source, or the backward path $y - (t_i, n) - t_i$ where t_i is a resource node.

□

Proof of Corollary e.2. The proof is similar to the one shown in Corollary 6.9.

Given a graph G and a path p in G, we denote $w(p, G)$ the weight of p in G.
$\Rightarrow$: Let $\hat{p'}$ be a shortest path in G_f^B between v_0 and v_n, corresponding to a path p' in G_f^8. The path p_{opt} is the one with the minimum weight in G_f^8. Thus, $w(p_{opt}, G_f^8) \leq w(p', G_f^8)$. The weight of every path p in the eight-layer residual graph G_f^8 is equal to the weight of its corresponding path $\hat{p}$ in the bipartite-like graph G_f^B. Therefore, the weight of $\hat{p'}$ is not larger than the weight of $\hat{p}_{opt}$ in the bipartite-like graph G_f^B, i.e., $w(\hat{p}_{opt}, G_f^B) \leq w(\hat{p'}, G_f^B)$. Since $\hat{p'}$ is a shortest path in G_f^B, then $\hat{p}_{opt}$ is the shortest path in G_f^B as required.

$\Leftarrow$: Let p' be a shortest path in the eight-layer residual graph G_f^8 between v_0 and v_n, corresponding to a path $\hat{p'}$ in G_f^B. The path $\hat{p}_{opt}$ is the one with the minimum weight in G_f^8. Thus, $w(\hat{p}_{opt}, G_f^B) \leq w(\hat{p'}, G_f^B)$. The weight of every path p in the eight-layer residual graph G_f^8 is equal to the weight of their corresponding path $\hat{p}$ in the bipartite-like graph G_f^B. Therefore, the weight of p' is not larger than the weight of p_{opt} in the eight-layer residual graph G_f^B, i.e., $w(p_{opt}, G_f^8) \leq w(p', G_f^8$. Since p' is a shortest path in G_f^8, then p_{opt} is the shortest path in G_f^B as required.

The paths p_{opt} and $\hat{p}_{opt}$ have the same weight by the definition of p_{opt}. □

e.3 Computing the Edge Weights in the Bipartite-like Graph

Given a min-cost flow f_L corresponding to a placement L, the weights of the bipartite-like graph edges $w_f(u \overset{\min}{\rightarrow} v)$ can be computed given the marginal-differential functions Δg (See Section 7.3.3) and the placement quantities L_i^j, as we show in the next lemma:

Lemma e.3. *Let f_L be a flow corresponding to a placement L that SSP-NE calculates in some iteration. The weights of the bipartite-like graph edges in G_f^B can be computed as follows:*

(a) $w_f(x \overset{\min}{\to} a^j) = -\Delta g^j(L^j + 1)$.

(b) $w_f(a^j \overset{\min}{\to} x) = \Delta g^j(L^j)$ *if* $L^j > 0$ *and otherwise* ∞.

(c) $w_f(a^j \overset{\min}{\to} t^i) = -\Delta g_i^j(L_i^j + 1)$.

(d) $w_f(t^j \overset{\min}{\to} a^i) = \Delta g_i^j(L_i^j)$ *if* $L_i^j > 0$ *and otherwise* ∞.

(e) $w_f(t^i \overset{\min}{\to} y) = -\Delta g_i(L_i + 1)$.

(f) $w_f(y \overset{\min}{\to} t^i) = \Delta g_i(L_i)$ *if* $L_i > 0$ *and otherwise* ∞.

The correctness of the lemma is based on the monotonicity of the marginal-differential functions Δg, and is similar to Lemma 6.10.

Proof of Lemma e.3. We will show parts (c) and (d) of the lemma. Other parts can be similarly proven.

To show part (c), let a^j and t^i respectively denote area and resource type nodes. All edges in an area-to-type path between a^j and t^i except the edges between (a^j, t^i) and (a^j, t^i, r), have an infinite capacity and zero weight. Thus, the weight of a minimal area-to-type path $w_{f_L}(a^j \overset{\min}{\to} t^i)$ is equal to the minimal edge weight between layer-4 (i.e., (a^j, t^i)) and layer-5 (i.e., (a^j, t^i, r)) vertices in G_f^8.

Let e be an edge of minimum weight between layer-4 and layer-5 vertices. If e has a cost of $-\Delta g_i^j(r)$ for $1 \leq r \leq L_i^j$, then the flow f sends flow through e. Thus, the residual capacity of e equals $c_f(e) = c(e) - f(e) = 1 - 1 = 0$, and therefore its weight in the residual graph is equal to infinity ($w_f(e) = \infty$).

If e has a cost of $-\Delta g_i^j(r)$ for $L_i^j + 1 \leq r$ then f does not send flow through e. Thus the residual capacity of e is equal to $c_f(e) = c(e) - f(e) = 1 - 0 = 1$, and therefore its weight in G_f is equal to the edge weight in G^8, i.e., $w_f(e) = w(e)$. Thus, $w_f(e)$ is equal to either $-\Delta g_i^j(L_i^j + 1), -\Delta g_i^j(L_i^j + 2), \ldots$. Among these edges, the one with the minimum cost must be $-\Delta g_i^j(L_i^j + 1)$ (as Δg_i^j is monotonically decreasing and $-\Delta g_i^j$ is monotonically increasing). Thus, the weight of the minimal area-to-type path $w_{f_L}(a^j \overset{\min}{\to} t^i)$ is equal to $-\Delta g_i^j(L_i^j + 1)$.

To show part (d), let a^j and t^i respectively denote area and resource type nodes. All edges in a type-to-area path between t^i and a^j in the residual graph G_f^8 are reverse edges. If f does not send flow through the path $a^j - (a^j, t^i) - (a^j, t^i, r) - t^i$ then the reverse edges

corresponding to the path (i.e., $(a^j, t^i) - a^j$, $(a^j, t^i, r) - (a^j, t^i)$ and $t^i - (a^j, t^i, r)$) have zero residual capacity, and thus they have an infinite weight. If f sends flow through the path $a^j - (a^j, t^i) - (a^j, t^i, r) - t^i$ then the reverse edges corresponding to the path have non-zero residual capacity, and the weight of the path is equal to the weight of the edge connecting (a^j, t^i) to (a^j, t^i, r).

The flow f sends flow through edges with costs $-\Delta g_i^j(1), -\Delta g_i^j(2), \ldots, -\Delta g_i^j(L_i^j)$. The cost of their respective reverse edges in the residual graph are equal to $\Delta g_i^j(1), \Delta g_i^j(2), \ldots, \Delta g_i^j(L_i^j)$. Among these edges, the edge with the minimum weight equal to $\Delta g_i^j(L_i^j)$ (as Δg_i^j is monotonically decreasing). Thus, the weight of the minimal type-to-area path $w_{f_L}(t^i \overset{\min}{\rightarrow} a^j)$ equals $\Delta g_i^j(L_i^j)$. In the special case where $L_i^j = 0$, all reverse edges between a^j and t_i have zero residual capacity, and infinite weight in G_f. Thus, in such case, $w_{f_L}(t^i \overset{\min}{\rightarrow} a^j) = \infty$. □

e.4 SSP-NE Update of the Placement Quantities

We can characterize how SSP-NE updates the placement quantities L_i^j, similar to the one described in Lemma 6.11, as follows:

Lemma e.4. *Let f be a flow corresponding to L that SSP computes in some iteration. Suppose that SSP augments the flow by the shortest path (found in Step 2 of SSP) $p_{opt} = x \overset{\min}{\rightarrow} a^{j_1} \overset{\min}{\rightarrow} t^{i_1} \overset{\min}{\rightarrow} a^{j_2} \ldots \overset{\min}{\rightarrow} a^{j_e} \overset{\min}{\rightarrow} t^{i_e} \overset{\min}{\rightarrow} y$. Then SSP updates the flow f to another flow f' that is corresponding to a placement L', where L' is defined as follows: 1) $L'^{j_l}_{i_l} \leftarrow L^{j_l}_{i_l} + 1$ for $1 \leq l \leq e$. 2) $L'^{j_{l+1}}_{i_l} \leftarrow L^{j_{l+1}}_{i_l} - 1$ for $1 \leq l \leq e-1$. 3) $L'^j_i \leftarrow L^j_i$ for $(i,j) \neq (i_l, j_l), (i_l, j_{l+1})$. Moreover, computing the quantities of L' takes linear time (i.e., it takes $O(|V^B| + |E^B|) = O(mk)$, where m is the number of resource types, k is the number of regions and $G^B = (V^B, E^B)$ is the bipartite-like graph).*

The proof is exactly the same as that of Lemma 6.11.

Proof of Lemma e.4. Since the capacity of each edge between (a^j, t^i) and (a^j, t^i, r) is equal to 1, we augment 1 unit of flow in each iteration of SSP. The edges in every minimal area-to-type path (i.e., $a^{j_l} \overset{\min}{\rightarrow} t^{i_l}$) were from the original edges of G^8, and therefore augmenting through p_{opt} increases the flow of node (a^{j_l}, t^{i_l}) by 1, i.e., $f'^{j_{l+1}}_{i_l} \leftarrow f^{j_{l+1}}_{i_l} + 1$. In contrast, type-to-area paths (i.e., $t^{i_l} \overset{\min}{\rightarrow} a^{j_{l+1}}$) are concatenation of reverse edges, and therefore augmenting through p_{opt} cancels the flow of these reverse edges, and thus $f'^{j_{l+1}}_{i_l} \leftarrow f^{j_{l+1}}_{i_l} - 1$ for $1 \leq l \leq e-1$. Since the new updated flow f' is corresponding to a some placement L' (See Corollary 6.4), then $L'^j_i = f^j_i$. Therefore, the lemma is followed. □

Appendix f

The Four-Layer Graph G^4, and the Proof of Claim 8.10

f.1 The Reduction to a four-layer Graph

f.1.1 Preliminaries: Introduction to the min-cost flow problem

This subsection is similar to Section 6.2.2. We start describing the **min-cost flow problem** [71], which is a generalization of the notable max flow problem (see [72]). In the problem, one considers a directed graph $G = (V, E)$ where every edge $e \in E$ has a non-negative integer *capacity* $c(e)$ and a real-value *weight* $w(e)$ (also called *cost*). The graph must contain two different nodes: a source node x and a sink node y. An *x-y-flow* $f : E \to R^+$ is defined on the graph edges $(v, v') \in E$ in the same way as defined in the max-flow problem. That means, the flow must satisfy the following properties: 1) *Capacity constraint:* for each edge e, we have $0 \leq f(e) \leq c(e)$. 2) *Conservation of flows:* for every vertex $v \in V \setminus \{x, y\}$ we have $\sum_{(v',v)\in E} f(v', v) = \sum_{(v,v')\in E} f(v, v')$. In addition to the standard definitions, we define the *flow in node* $v \neq x, y$ as the income flow (and by conservation of flows, the outcome flow) to (from) node v. We denote it by $f^{in}(v)$, which equals $f^{in}(v) = \sum_{(v,v')\in E} f(v, v') (= \sum_{(v',v)\in E} f(v', v) = f^{out}(v))$. The *flow value* of f, as defined in the max-flow problem, is $|f| = \sum_{(x,v)\in E} f(x, v) = \sum_{(v,y)\in E} f(v, y)$. The *weight (or cost) of flow* f is $w(f) = \sum_{e\in E} f(e)w(e)$.

Given a parameter n, the classic *minimum-cost flow* problem is to find a flow f_{opt} of value n that has a minimum weight among all flows of value n. This means that for every flow f' where $|f'| = |f_{opt}| = n$ we have $w(f_{opt}) \leq w(f')$. It is well-known that if the capacities $c(e)$ of a min-cost flow network are integers, then there exists a min-cost

flow f where the flow in every edge is integer; a proof can be found in [70], Theorem 9.10.

In this chapter, we generalized the min-cost flow problem, so that it is defined given a multigraph $G = (V, E)$, i.e., there are multiple edges between every two vertices in G.

f.1.2 The four-layer graph G^4

To devise the construction of G^4 (Theorem 8.6 and Figure 8.1) we recall that the profit function $P(D, L)$ is composed of the sum of marginal profit functions $\zeta_i(D, L_i) = \zeta_i(D_i, L_i), \zeta_i^j(D, L_i) = \zeta_i^j(D_i^j, L_i^j), \zeta^j(L^j)$ (Eq. (3.5). That means

$$P(L, D) = \underbrace{\sum_{i=1}^{m}\sum_{j=1}^{k}\zeta_i^j(L_i^j, D)}_{\text{area \& type}} + \underbrace{\sum_{i=1}^{m}\zeta_i(L_i, D)}_{\text{type}} + \underbrace{\sum_{j=1}^{k}\zeta^j(L^j)}_{\text{area}}. \tag{f.1}$$

The marginal profit functions ζ_i^j, ζ_i depend on the demand D, while ζ^j does not depend on the demand D. We denote the **differential** of a discrete function ζ as $\Delta\zeta(n, D) = \zeta(n+1, D) - \zeta(n, D)$. In addition, we recall that the size of every placement is bounded by the storage constant s, which can be as large as one wishes.

The four-layer multigraph G^4 (graph with parallel edges between vertices) is composed of a source x, region nodes $j_1, j_2, \ldots j_k$, resource type nodes $i_1, i_2 \ldots i_m$ and a sink y. Between every two nodes of successive layers we connect s edges, all with flow capacity $c(e) = 1$. The weight of edges between the source x and a region node j are minus the area j marginal function differential, i.e., $(-1)\cdot\Delta\zeta^j(0), (-1)\cdot\Delta\zeta^j(1), \ldots, (-1)\cdot\Delta\zeta^j(s-1)$, and the edges are denoted respectively by $j^1, j^2, \ldots, j^s$ or by $(x, j)^1, (x, j)^2, \ldots, (x, j)^s$. The weight of edges between region nodes j and resource type node i are minus the area-j type i marginal function differential (-1), $(-1)\cdot\Delta\zeta_i^j(0, D), (-1)\cdot\Delta\zeta_i^j(1, D), \ldots, (-1)\cdot\Delta\zeta_i^j(s-1, D)$ denoted by $(j, i)^1, (j, i)^2, \ldots, (j, i)^s$. The weight of edges between resource type node i and the sink y are minus the type i marginal function differential. Finally, we then connect the source x to the sink y with s edges of flow capacity $c(e) = 1$ and weight $w(e) = 0$. In Figure f.1, we depict the four-layer multigraph.

f.1.3 The corresponding flow

For every placement L we define its **corresponding flow** f_L in the four-layer graph G^4 in the following way: 1) For every region j and resource type i, we set the flow on the

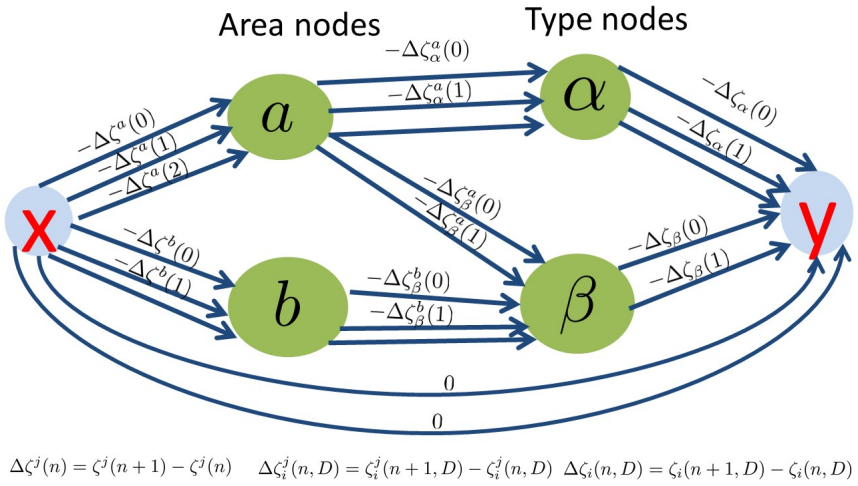


FIGURE F.1: The four-layer multigraph G^4. We omit some edges from the graph for the sake of presentation.

edges $(j,i)^1, (j,i)^2, \dots (j,i)^{L^j_i}$ to 1. 2) For every region j we set the flow on the edges $j^1, j^2, \dots j^{L^j}$ to 1. 3) For every resource type i we set the flow on the edges $i^1, i^2, \dots i^{L_i}$ to 1. 4) The flow between x and y is equal to $s - \sum_j L^j$. Note the required flow f_L is equal to the storage constant s. An example can be seen in Chapter 8, Figure 8.1.

f.2 Proof of Claim 8.7

Claim 8.7. *Let L be a placement. Then its profit is equal to a constant c minus the weight of its corresponding flow f_L, i.e.,*

$$P(D,L) = c - W(f_L, w(D)), \tag{f.2}$$

where c depends neither on the placement L nor the demand D, and $W(f, w(D))$ is the flow weight of the flow f using the edge weights w resulting from the demand D.

Proof of Claim 8.7. The proof follows from the fact that the sum of differential weights $(-1)\cdot\Delta\zeta^j(0,D), (-1)\cdot\Delta\zeta^j(1,D), \dots, (-1)\cdot\Delta\zeta^j(L^j-1)$ forms a telescopic series. The formal proof is presented below.

Let j be a region node, and denote the flow weight between x and j as $w_f(x,j) = \sum_{e \text{ is an edge connecting } x \text{ to } j} f(e)w(e)$. Since f_L sends 1 unit of flow through edges $j^1, j^2, \dots j^{L^j}$ with differential weights $(-1)\cdot\Delta\zeta^j(0), (-1)\cdot\Delta\zeta^j(1), \dots, (-1)\cdot\Delta\zeta^j(L^j-1)$, then

$$w(x,j) = \sum_{n=0}^{L^j_i - 1} (-1)\cdot\Delta\zeta^j(n) \underbrace{=}_{\text{definition of } \Delta g} \sum_{n=0}^{L^j-1} (\zeta^j(n) - \zeta^j(n+1)) \underbrace{=}_{\text{telescopic series}} \zeta^j(0) - \zeta^j(L^j),$$

and therefore,

$$\zeta^j(L^j) = \zeta^j(0) - w(x, j). \tag{f.3}$$

According to Eq. (3.8), $\zeta^j(0) = -C^j(0)$ is constant, and depends neither on the placement L nor on the demand D.

For every region node j and resource type node i, we can similarly define the flow between j and i, i.e., $w_f(j, i)$ and the flow between i and y, i.e., $w_f(i, y)$. We similarly derive that

$$\zeta_i^j(L_i^j, D) = \zeta_i^j(0, D) - w_f(j, i), \tag{f.4}$$

$$\zeta_i(L_i, D) = \zeta_i(0, D) - w_f(i, y), , \tag{f.5}$$

and according to Eqs. (3.7)),(3.6)), $\zeta_i(0, D) = -C_i(0), \zeta_i^j(0, D) = -C_i^j(0)$ are constants that depend neither on the placement L nor the demand D.

The weight of f_L is equal to the sum of weights $w_f(x, j), w_f(j, i), w_f(i, y)$ over all region nodes j and resource type nodes i. In addition, according to Eq. (f.1), the profit $P(D, L)$ is equal to the sum of $\zeta_i^j(L_i^j, D), \zeta_i(L_i, D), \zeta^j(L^j)$ over all region nodes j and resource type nodes i. If we define the constant c to be the sum of $\zeta_i^j(0, D), \zeta_i(0, D), \zeta^j(0)$ then summing up Eqs. (f.3),(f.4),(f.5) over all region nodes j and resource type nodes i will result in the required result.

□

f.3 Proof of Claim 8.8

Claim 8.8. *There exists a placement L whose corresponding flow f_L is the min-cost flow of G^4 with edge weights $w(D)$ corresponding to demand D and with required flow $|f| = s$. Moreover, L is the optimal placement (i.e., solves SPP-4) for demand D.*

Proof of Claim 8.8. The correctness of the claim stems from the follows three facts: 1) The marginal profit functions g are concave functions (See Section 3.2), i.e., the marginal-differential functions Δg are monotonically non-increasing functions. 2) If the capacities of a min-cost flow network are integer, then its min-cost flow f is integer (that is, the flow on every edge is integer). The latter is a well-known theorem (the Integrability Property in [70], Theorem 9.10). 3) Claim 8.7 proven above. Below we present the formal proof of the Claim.

First, the multigraph G^4 contains integer capacities, and there is a min-cost flow f_{min} of required flow $|f| = s$, with integer flow values. Note that every edge $e \neq (x, y)$ in G^4 has capacity $c(e) = 1$. Thus, the flow f_{min} in each edge can be either 0 or 1.

Let j be a region node and i be a resource type node. We denote the n-indexed edges $j^n, (j,i)^n, i^n$ respectively as the edge connecting x to j, the edge connecting j to i, and the edge connecting i to y with respective weights of $(-1) \cdot \Delta\zeta^j(n-1), (-1) \cdot \Delta\zeta_i^j(n-1, D), (-1) \cdot \Delta\zeta_i(n-1, D)$. The marginal profit functions $\zeta^j, \zeta_i^j, \zeta_i$ are concave, and $\Delta\zeta^j, \Delta\zeta_i^j, \Delta\zeta_i$, are monotonically non-increasing functions in n. Thus, the weights of $j^n, (i,j)^n, i^n$ are monotonically non-decreasing in n, and a min-cost flow f_{min} will prefer to send flow through edges with a possible index to minimize the weight of the flow, i.e., there exists L_i^j, L^j, L_i such that we send flow through edges $j^1, j^2 \ldots j^{L^j}, (j,i)^1, (j,i)^2, \ldots (j,i)^{L_i^j}, i^1, i^2, \ldots i^{L_i}$ (and their filler vertices) 1 unit of flow, and do not send flow through the other edges. By conservation of flow we note that $L^j = f_{min}(j) = \sum_{i=1}^m f_{min}(i,j) = \sum_{i=1}^m L_i^j$ and $L_i = f_{min}(i) = \sum_{j=1}^k f_{min}(i,j) = \sum_{j=1}^k L_i^j$. The flow through edge (x, y) equals $s - \sum_{j=1}^k L^j$. Thus, f_{min} is simply the associated flow of placement $L_{opt} = \{L_i^j\}$.

To prove that $L_{opt} = \{L_i^j\}$ is the optimal unconstrained placement, let L' be a placement with its associated flow $f_{L'}$. From the definition of a min-cost flow, the weight of $f_{L_{opt}}$ is smaller than the flow of $f_{L'}$. By Claim 8.7, we derive that L_{opt} has larger profit than L', thus L_{opt} is the optimal unconstrained placement. □

f.4 Proof of Claim 8.10

Claim 8.10. *Let $w = w(t)$ and $w' = w(t+1)$ be the edge weights of G^4 with respect to demands $D(t)$ and $D(t+1)$ respectively. Then the difference of weights $|w(e) - w'(e)|$ is bounded by the demand distance $d(D(t), D(t+1))$, i.e.,*

$$\sum_{e \in E} |w'(e) - w'(e)| \leq d(D(t), D(t+1)). \tag{f.6}$$

Proof of Claim 8.10. To compute these weight values, we use the following formula to compute the partial expectation (i.e.,$E_X(\min(n, X))$) for every positive discrete random variable X:

$$E_X(\min(n, X)) = \sum_{k=1}^{n} \Pr(X \geq k). \tag{f.7}$$

The correctness of the equation is shown in Theorem 3.1.

Using Eq f.7 we can now compute the weights of G^4 edges. In Section 8.1, we expressed the marginal profit functions by the model parameters, which is equal to

$$\begin{aligned}\zeta_i^j(L_i^j, D_i^j) &= R_i^j \cdot E_{D_i^j}[\min(L_i^j \cdot B_i, D_i^j)] - C_i^j(L_i^j)\\ \zeta_i(L_i, D_i) &= R_i \cdot E_{D_i}[\min(L_i \cdot B_i, D_i)] - C_i(L_i)\\ \zeta^j(L^j) &= -C^j(L^j).\end{aligned} \tag{f.8}$$

Using Eqs. (f.8), (f.7) we observe that the edge weights of the n^{th} edge respectively connecting the source x to region node j (denoted by $(x,j)^n$), region node j and resource type node i (denoted by $(j,i)^n$), resource type node i to sink y (denoted by $(i,y)^n$) are equal to

$$-\Delta\zeta_i^j(n-1, D) = \Delta C_i^j(n-1) - R_i^j \cdot \sum_{k=(n-1)\cdot B_i+1}^{n\cdot B_i} \Pr(D_i^j \geq k), \tag{f.9}$$

$$-\Delta\zeta_i(n-1, D) = \Delta C_i(n-1) - R_i \cdot \sum_{k=(n-1)\cdot B_i+1}^{n\cdot B_i} \Pr(D_i \geq k) \tag{f.10}$$

$$-\Delta\zeta^j(n-1) = \Delta C^j(n-1). \tag{f.11}$$

Let $w = w(t)$ and $w' = w(t+1)$ be the edge weights of the same n^{th} edge in G^4 with respect to demand $D(t)$ and $D(t+1)$. Then, the difference of weights $|w(e) - w'(e)|$ on the n^{th} edge is equal to

$$|w(e) - w'(e)| = \begin{cases} R_i^j \cdot |\sum_{k=(n-1)\cdot B_i+1}^{n\cdot B_i} \Pr(D_i^j(t) \geq k) - \Pr(D_i^j(t+1) \geq k)| & e = (i,j)^n,\\ R_i \cdot |\sum_{k=(n-1)\cdot B_i+1}^{n\cdot B_i} \Pr(D_i(t) \geq k) - \Pr(D_i(t+1) \geq k)| & e = i^n,\\ 0 & \text{Otherwise.}\end{cases} \tag{f.12}$$

Let j be a region node and i a resource type node. Then summing the weight difference over all edges $(j,i)^n$ is

$$\sum_{n=1}^{\infty} |w((i,j)^n) - w'((i,j)^n)| \underbrace{\leq}_{\text{using triangular inequality}} R_i^j \cdot \sum_{k=1}^{\infty} |\Pr(D_i^j(t) \geq k) - \Pr(D_i^j(t+1) \geq k)|$$

$$\underbrace{=}_{\Pr(X\geq n)=1-\Pr(X\leq n-1)} R_i^j \cdot \sum_{k=0}^{\infty} |\Pr(D_i^j(t) \leq k) - \Pr(D_i^j(t+1) \leq k)| = R_i^j \cdot d(D_i^j(t), D_i^j(t+1)). \tag{f.13}$$

Similarly, the sum of weight difference over all edges i^n is

$$\sum_{n=1}^{\infty} |w(i^n) - w'(i^n)| \underbrace{\leq}_{\text{using triangular inequality}} R_i \cdot \sum_{k=1}^{\infty} |\Pr(D_i(t) \geq k) - \Pr(D_i(t+1) \geq k)|$$
$$\underbrace{=}_{\Pr(X \geq n) = 1 - \Pr(X \leq n-1)} R_i \cdot \sum_{k=0}^{\infty} |\Pr(D_i(t) \leq k) - \Pr(D_i(t) \leq k)| = R_i \cdot d(D_i(t), D_i(t+1)). \tag{f.14}$$

If we sum up Eqs. (f.13), (f.14) over all regions j and resource types i, then we have

$$\sum_{e \in E} |w(e) - w'(e)| \underbrace{=}_{Eq.\ (f.12)} \sum_{i=1}^{m} \sum_{n=1}^{s} |w(i^n) - w'(i^n)| + \sum_{j=1}^{k} \sum_{i=1}^{m} \sum_{n=1}^{s} |w((j,i)^n) - w'((j,i)^n)|$$
$$\underbrace{\leq}_{\text{Eqs. (f.13), (f.14)}} \sum_{i=1}^{m} R_i \cdot d(D_i(t), D_i(t+1)) + \sum_{j=1}^{k} \sum_{i=1}^{m} R_i^j \cdot d(D_i^j(t), D_i^j(t+1)) \underbrace{=}_{\text{Definition of demand distance}} d(D(t), D(t+1)), \tag{f.15}$$

as required. □

Appendix g

Chapter 9 – Proof of Claims

g.1 Proof of Observation 9.1

Observation 9.1. *Given a placement L, the profitability of every unary or move operation can be computed as follows:*

$$\Delta(a_j, L) = \Delta g_j^{local}(L_j + 1) + \Delta g^{global}(|L| + 1) \tag{g.1}$$

$$\Delta(s_j, L) = -\Delta g_j^{local}(L_j) - \Delta g^{global}(|L|) \tag{g.2}$$

$$\Delta(b_{j_1 \to j_2}, L) = -\Delta g_{j_1}^{local}(L_{j_1}) + \Delta g_{j_2}^{local}(L_{j_2} + 1), \quad \textit{for} \quad j_1 \neq j_2 \tag{g.3}$$

Proof of Observation 9.1. **To prove Eq. (g.1):** let $L' = a_j(L)$ be the placement derived by adding to L a single resource into region j. Thus, the quantities of L' are equal to $L'_i = L_i$ for $i \neq j$, $L'_j = L_j + 1$ and $|L'| = |L| + 1$. According to Eq. (9.4), the profit of L' is

$$P(L') \underbrace{=}_{\text{Eq. (9.4)}} c + \sum_{i=1}^{k} \sum_{n=1}^{L'_i} \Delta g_i^{local}(n) + \sum_{n=1}^{|L'|} \Delta g^{global}(n)$$

$$\underbrace{=}_{\text{Definition of } L'} c + \sum_{i \neq j} \sum_{n=1}^{L_i} \Delta g_i^{local}(n) + \sum_{n=1}^{L_j+1} \Delta g_j^{local}(n) + \sum_{n=1}^{|L|+1} \Delta g^{global}(n)$$

$$= c + \sum_{i \neq j} \sum_{n=1}^{L_i} \Delta g_i^{local}(n) + \sum_{n=1}^{L_j} \Delta g_j^{local}(n) + \Delta g_j^{local}(L_j+1) + \sum_{n=1}^{|L|} \Delta g^{global}(n) + \Delta g^{global}(|L|+1)$$

$$\underbrace{=}_{\text{Changing order of sum}} c + \sum_{i=1}^{k} \sum_{n=1}^{L_i} \Delta g_i^{local}(n) + \sum_{n=1}^{|L|} \Delta g^{global}(n) + \Delta g_j^{local}(L_j+1) + \Delta g^{global}(|L|+1)$$

$$\underbrace{=}_{\text{Eq. (9.4)}} P(L) + \Delta g_j^{local}(L_j+1) + \Delta g^{global}(|L|+1). \quad \text{(g.4)}$$

Thus, $\Delta(a_j, L) = P(L') - P(L) = \Delta g_j^{local}(L_j+1) + \Delta g^{global}(|L|+1)$ as required.

To prove Eq. (g.2): let $L' = s_j(L)$ be the placement derived by subtracting a single resource from L in region j. Thus, the quantities of L' are equal to $L'_i = L_i$ for $i \neq j$, $L'_j = L_j - 1$ and $|L'| = |L| - 1$. According to Eq. (9.4), the profit of L' is

$$P(L') \underbrace{=}_{\text{Eq. (9.4)}} c + \sum_{i=1}^{k} \sum_{n=1}^{L'_i} \Delta g_i^{local}(n) + \sum_{n=1}^{|L'|} \Delta g^{global}(n)$$

$$\underbrace{=}_{\text{Definition of } L'} c + \sum_{i \neq j} \sum_{n=1}^{L_i} \Delta g_i^{local}(n) + \sum_{n=1}^{L_j-1} \Delta g_j^{local}(n) + \sum_{n=1}^{|L|-1} \Delta g^{global}(n)$$

$$= c + \sum_{i \neq j} \sum_{n=1}^{L_i} \Delta g_i^{local}(n) + \sum_{n=1}^{L_j} \Delta g_j^{local}(n) - \Delta g_j^{local}(L_j) + \sum_{n=1}^{|L|} \Delta g^{global}(n) - \Delta g^{global}(|L|) =$$

$$\underbrace{=}_{\text{Changing order of sum}} c + \sum_{i=1}^{k} \sum_{n=1}^{L_i} \Delta g_i^{local}(n) + \sum_{n=1}^{|L|} \Delta g^{global}(n) - \Delta g_j^{local}(L_j) - \Delta g^{global}(|L|)$$

$$\underbrace{=}_{\text{Eq. (9.4)}} P(L) - \Delta g_j^{local}(L_j) - \Delta g^{global}(|L|). \quad \text{(g.5)}$$

Thus, $\Delta(s_j, L) = P(L') - P(L) = -\Delta g_j^{local}(L_j) - \Delta g^{global}(|L|)$ as required.

To prove Eq. (g.3): let $L' = s_{j_1}(L)$ be the placement derived by subtracting a single resource from L from region j_1, and let $L'' = a_{j_2}(L') = b_{j_1 \to j_2}(L)$ be the placement of moving a resource from region j_1 to region j_2 ($j_1 \neq j_2$) in L. The quantities of L' are

equal to $L'_i = L_i$ for $i \neq j_1$, $L'_{j_1} = L_{j_1} - 1, |L'| = |L| - 1$. Thus the profitability of $b_{j_1 \to j_2}$ over L is

$$\begin{aligned}
\Delta(b_{j_1\to j_2}, L) &= P(L'') - P(L) = P(L'') - P(L') + P(L') - P(L) \\
&\underbrace{=}_{\text{definition of profitability}} \Delta(a_{j_2}, L') + \Delta(s_{j_1}, L) \\
&\underbrace{=}_{\text{Eqs. (g.1), (g.2)}} \Delta g^{local}_{j_1}(L'_{j_1}+1) + \Delta g^{global}(|L'|+1) - \Delta g^{local}_{j_2}(L_{j_2}) - \Delta g^{global}(|L|) \\
&\underbrace{=}_{\text{Definition of } L'} \Delta g^{local}_{j_1}(L_{j_1}+1) + \Delta g^{global}(|L|) - \Delta g^{local}_{j_2}(L_{j_2}) - \Delta g^{global}(|L|) \\
&= \Delta g^{local}_{j_1}(L_{j_1}+1) - \Delta g^{local}_{j_2}(L_{j_2}). \quad \text{(g.6)}
\end{aligned}$$

as required.

□

g.2 Proof of Claim 9.11

Claim 9.11. *Suppose $C_G = (o^1, \ldots, o^n)$ is a U&Me-like sequence. Suppose that both operations o^i and o^{i+1} are either addition, subtraction or move operations. Then, the i^{th} relative profitability is not smaller than that of $i+1$, i.e., $Rel(i, C) \geq Rel(i+1, C)$.*

Proof of Claim 9.11. We denote $L = R(i, C_G)$ the repositioned placement after conducting the first i operations on C_G.

In case o^i and o^{i+1} are both addition operations: Suppose that the sequence C_G adds to L a single resource to region j_1 and afterwards adds a single resource from region j_2. By Eq. (9.5), the relative profitability of adding to j_1 is equal to $Rel(i, C_G) = \Delta g^{local}_{j_1}(L_{j_1}+1) + \Delta g^{global}(|L|+1)$. There can be two cases:

If $j_1 = j_2$: The relative profitability of adding to j_2 is $Rel(i+1, C_G) = \Delta g^{local}_{j_1}(L_{j_1}+2) + \Delta g^{global}(|L|+2)$. Since the marginal-differential functions Δg are monotonically non-increasing, we derive that $Rel(i+1, C_G) = \Delta g^{local}_{j_1}(L_{j_1}+2) + \Delta g^{global}(|L|+2) \leq \Delta g^{local}_{j_1}(L_{j_1}+1) + \Delta g^{global}(|L|+1) = Rel(i, C_G)$, as required.

If $j_1 \neq j_2$: The relative profitability of adding to j_2 is equal to $Rel(i+1, C_G) = \Delta g^{local}_{j_2}(L_{j_2}+1) + \Delta g^{global}(|L|+2)$. Since C_G is a greedily ordered sequence (Observation 9.10), we derive that $\Delta g^{local}_{j_1}(L_{j_1}+1) \geq \Delta g^{local}_{j_2}(L_{j_2}+1)$ (otherwise, operation o^i cannot add to region j_1). Since the marginal-differential functions Δg are monotonically

non-increasing, then $\Delta g^{global}(|L|+1) \geq \Delta g^{global}(|L|+2)$. Therefore, we derive that $Rel(i+1, C_G) = \Delta g^{local}_{j_2}(L_{j_2}+1) + \Delta g^{global}(|L|+2) \leq \Delta g^{local}_{j_1}(L_{j_1}+1) + \Delta g^{global}(|L|+1) = Rel(i, C_G)$, as required

In case o^i and o^{i+1} are both subtraction operations: Suppose that the sequence C_G subtracts a resource from region j_1 and afterwards subtracts one from region j_2. By Eq. (9.6), the relative profitability of subtracting from j_1 is equal to $Rel(i, C_G) = -\Delta g^{local}_{j_1}(L_{j_1}) - \Delta g^{global}(|L|)$. There can be two cases:

If $j_1 = j_2$: The relative profitability of subtracting from j_2 is $Rel(i+1, C_G) = -\Delta g^{local}_{j_1}(L_{j_1}-1) - \Delta g^{global}(|L|-1)$. Since the marginal-differential functions Δg are monotonically non-increasing, we derive that $Rel(i+1, C_G) = -\Delta g^{local}_{j_1}(L_{j_1}-1) - \Delta g^{global}(|L|-1) \leq -\Delta g^{local}_{j_1}(L_{j_1}) - \Delta g^{global}(|L|) = Rel(i, C_G)$, as required.

If $j_1 \neq j_2$: The relative profitability of subtracting from j_2 is equal to $Rel(i+1, C_G) = -\Delta g^{local}_{j_2}(L_{j_2}) - \Delta g^{global}(|L|-1)$. Since C_G is a greedily ordered sequence (Observation 9.10), we derive that $-\Delta g^{local}_{j_1}(L_{j_1}) \geq -\Delta g^{local}_{j_2}(L_{j_2})$ (otherwise, operation o^i cannot subtract from region j_1). Since the marginal-differential functions Δg are monotonically non-increasing, then $-\Delta g^{global}(|L|) \geq -\Delta g^{global}(|L|-1)$. Therefore, we derive that $Rel(i+1, C_G) = -\Delta g^{local}_{j_2}(L_{j_2}) - \Delta g^{global}(|L|-1) \leq -\Delta g^{local}_{j_1}(L_{j_1}) - \Delta g^{global}(|L|) = Rel(i, C_G)$, as required

In case o^i and o^{i+1} are both move operations: Suppose that the sequence C_G applies move operations $o^i = b_{j_1 \to j_2}$ and then move operation $o^{i+1} = b_{j_3 \to j_4}$. By Eq. (9.7), the relative profitability of o^i is equal to $Rel(i, C_G) = \Delta g^{local}_{j_2}(L_{j_2}+1) - \Delta g^{local}_{j_1}(L_{j_1})$. We denote $L' = o^i(L)$ the placement after applying o^i over L. By Eq. (9.7), the relative profitability of o^i is $Rel(i, C_G) = \Delta g^{local}_{j_4}(L'_{j_4}+1) - \Delta g^{local}_{j_3}(L'_{j_3})$. We will prove that $\Delta g^{local}_{j_2}(L_{j_2}+1) \geq \Delta g^{local}_{j_4}(L'_{j_4}+1)$, and in a similar way we can show that $-\Delta g^{local}_{j_1}(L_{j_1}) \geq -\Delta g^{local}_{j_3}(L'_{j_3})$.

Since C_G is a non-canceling sequence, o^i cannot remove resources from regions j_4, j_2, i.e., $j_1 \neq j_2, j_4$. Thus, there can be only two cases:

If $j_2 = j_4$: Then $L'_{j_4} = L_{j_4}+1$ and thus $\Delta g^{local}_{j_2}(L_{j_2}+1) = \Delta g^{local}_{j_4}(L_{j_4}+1) \geq \Delta g^{local}_{j_4}(L_{j_4}+2) = \Delta g^{local}_{j_4}(L'_{j_4}+1)$.

If $j_2 \neq j_4$: Then $L'_{j_4} = L_{j_4}$. Since C_G is a greedily ordered sequence, we derive that $\Delta g^{local}_{j_4}(L'_{j_4}+1) = \Delta g^{local}_{j_4}(L_{j_4}+1) \leq \Delta g^{local}_{j_2}(L_{j_2}+1)$.

□

g.3 Proof of Claim 9.12

Claim 9.12. *Let L be a placement, and G a multiset of non-canceling unary operations. Suppose that G does not contain any profitable unary operations on L. Let m be a move operation, composed of a subtraction and an addition operation from G. Let $m(L)$ be the placement derived by conducting m over L. Then G does not contain any profitable unary operations on $m(L)$.*

Proof of Claim 9.12. Suppose that $m = b_{j_2 \to j_1}$ adds to region j_1 and subtracts from region j_2. We denote $L' = m(L)$. Let a_{j_3} be an addition operation in G. Then $j_3 \neq j_2$, otherwise G includes a canceling operation that adds to and subtracts from region $j_2 = j_3$. Therefore, $L'_{j_3} = L_{j_3}$ if $j_3 \neq j_1$ and otherwise $L'_{j_3} = L_{j_3} + 1$. Either way, we have $L'_{j_3} \geq L_{j_3}$. Also, the number of resources in L is equal to the number of resources in L', i.e., $|L| = |L'|$. Therefore, we derive by Eq. (9.7) that

$$\Delta(a_{j_3}, m(L)) = g^{local}_{j_3}(L'_{j_3} + 1) + g^{global}(|L'| + 1) \underbrace{\leq}_{\Delta g \text{ are monotonically non-decreasing}} g^{local}_{j_3}(L_{j_3} + 1) + g^{global}(|L| + 1) =$$

$$= \Delta(a_{j_3}, L) \underbrace{\leq}_{G \text{ does not contain any profitable unary operation on } L} 0. \quad \text{(g.7)}$$

This means that, a_{j_3} is not profitable over $m(L)$. In a similar way, we can prove that if s_{j_3} is a subtraction operation in G, then it is not profitable over $m(L)$. Thus, we have shown that G does not contain any profitable unary operation on $m(L)$.

□

g.4 Proof of Claim 9.15

Claim 9.15. *Let G be a valid multiset with a U&Me-like sequence C_G. In the following cases, we show a multiset G' that is better than G:*

(A) If C_G is a prefix of the U&Me sequence $C_{U\&Me}$, then $G_{U\&Me}$ is better than G.

(B) If the U&Me sequence $C_{U\&Me}$ is a prefix of C_G, then $G_{U\&Me}$ is better than G.

(C) Suppose that C_G contains non-profitable operations, and let G' be G where these non-profitable operations are removed. Then G' is better than G.

(D) Suppose G contains canceling operations, and let G' be the valid multiset G where canceling operations are removed. Then G' is better than G.

Proof of Claim 9.15:

Proof of (A): Let $C_G = (o_G^1, o_G^2, \ldots, o_G^{|C_G|})$ and $C_{U\&Me} = (o_{U\&Me}^1, o_{U\&Me}^2, \ldots, o_{U\&Me}^{|C_{U\&Me}|})$ respectively denote the U&Me-like sequence of G and the U&Me sequence. Since C_G is a prefix of $C_{U\&Me}$ the operations $o_G^i = o_{U\&Me}^i$ satisfies for every $1 \leq i \leq |C_G|$. It is obvious that the potential of G, which is non-zero, is strictly larger than the U&Me multiset $G_{U\&Me}$, which has zero potential. The U&Me algorithm chooses profitable operations, and in particular $o_{U\&Me}^{|C_G|+1}, o_{U\&Me}^{|C_G|+2}, \ldots o_{U\&Me}^{|C_{U\&Me}|}$ are all profitable operations with positive relative profitability. Conducting these profitable operations on L_G derives the U&Me placement, $L_{U\&Me}$, and therefore, we derive that $P(L^G) \leq P(L_{U\&Me})$.

Proof of (B): Since the first potential of G is non-zero, then we derive that $\pi(G) > (0,0,0) = \pi(G_{U\&Me})$. It is left to show that the profit of the placement corresponding to G is not larger than the U&Me placement, i.e., $P(L^G) \leq P(L_{U\&Me})$. Let $C_G = (o_G^1, o_G^2, \ldots, o_G^{|C_G|})$ and $C_{U\&Me} = (o_{U\&Me}^1, o_{U\&Me}^2, \ldots, o_{U\&Me}^{|C_{U\&Me}|})$ respectively denote the U&Me-like sequence of G and the U&Me sequence. Since U&Me sequence $C_{U\&Me}$ is a prefix of C_G, i.e., $o_G^i = o_{U\&Me}^i$ for every $1 \leq i \leq |C_{U\&Me}|$, then we derive that the U&Me unary multiset $G_{U\&Me} =$ is a subset of G.

According to the U&Me algorithm, the U&Me algorithm stops in the following cases:

(1) The U&Me algorithm has repositioned r resources.

(2) The U&Me algorithm has repositioned $r-1$ resources, and the last operation in the sequence $o_{U\&Me}^{|C_{U\&Me}|}$ is a move operation.

(3) There is no profitable move operation.

In case (1), $G_{U\&Me}$ contains exactly r unary operations. Since G is a valid multiset, i.e., it contains r operations at most, and $G_{U\&Me}$ is a subset of G, we derive that $G_{U\&Me} = G$, which contradicts our assumption that G has zero potential.

In case (2), since the multisets $G \neq G_{U\&Me}$ are different from each other, we derive that $G_{U\&Me}$ contains exactly $r-1$ unary operations, and G contains r operations. The U&Me sequence $C_{U\&Me}$ is a prefix of C_G, and therefore we derive that 1) the length of C_G is equal to the length of $C_{U\&Me}$ plus 1, i.e., $|C_G| = |C_{U\&Me}| + 1$ operations of C_G. 2) The last operation in G, $o_G^{|C_{U\&Me}|+1}$, must be a unary operation. 3) The last U&Me operation $o^{|C_{U\&Me}|_{U\&Me}} = o^{|C_{U\&Me}|_G}$ is a move operation. By these properties we derive that the last operation in C_G, $o_G^{|C_{U\&Me}|+1}$, must be a non-profitable unary operation (according to Claim 9.13). Since, conducting the non-profitable operation $o_G^{|C_G|}$ on the U&Me placement $L_{U\&Me}$ derives the placement L^G corresponding to G, then we derive that $P(L^G) \leq P(L_{U\&Me})$.

In case (3), by the third property (C) in Corollary 9.14 we derive the required result.

Proof of (C): Removing canceling operations from G does not change the corresponding placement, i.e., $L^G = L^{G'}$. Therefore, the profit of both placements is identical, i.e., $P(L^G) = P(L^{G'})$. Also, since the U&Me multiset $G_{U\&Me}$ does not contain canceling operations, then the first potential $\pi_1(G) = |G \setminus G_{U\&Me}|$ can only be reduced, i.e., $\pi_1(G) > \pi_1(G')$. This implies that $\pi(G) >_L \pi(G')$, and thus, G' is better than G.

Proof of (D): Removing non-profitable operations from G can only increase the placement profit, i.e., $P(L^G) \leq P(L^{G'})$, according to Claim 9.13. Also, the U&Me multiset $G_{U\&Me}$ does not contain non-profitable operation. Thus, we derive that the first potential $|G \setminus G_{U\&Me}|$ can only be reduced, i.e., $\pi_1(G) > \pi_1(G')$, which implies $\pi(G) >_L \pi(G')$. Thus, G' is better than G.

□

g.5 Proof of Observation 9.16

Observation 9.16. *Either one of the following conditions holds: 1) The first uncommon operation in $C_{U\&Me}$ adds to region $j_{U\&Me}$, and the first uncommon operation in C_G does not add to that region. 2) The first uncommon operation in $C_{U\&Me}$ subtracts from region $j_{U\&Me}$ and the first uncommon operation in C_G does not subtract from that region.*

Proof of Observation 9.16. Suppose, by way of contradiction, that both conditions do not hold. The proof is divided into several cases, where every case leads to a contradiction.

a) The first uncommon operation in $C_{U\&Me}$ and the first uncommon operation in C_G are both unary operations. As the condition in the observation does not hold, we derive that these operations are identical, i.e., $o^{n+1}_{U\&Me} = o^{n+1}_G$ – a contradiction, as the first uncommon operation in $C_{U\&Me}$, $o^{n+1}_{U\&Me}$, differs than the first uncommon operation in C_G, i.e., $o^{n+1}_{U\&Me} \neq o^{n+1}_G$.

b) The first uncommon operation in $C_{U\&Me}$ and the first uncommon operation in C_G are both move operations. The contradiction is derived similarly to case a).

c) The first uncommon operation in $C_{U\&Me}$ is a move operation, and the first uncommon operation in C_G is a unary operation. The contradiction is derived similar to case a).

d) The first uncommon operation in $C_{U\&Me}$ is a unary operation, and the first uncommon operation in C_G is a move operation. Suppose, without loss of generality, that the first uncommon operation in $C_{U\&Me}$ is an addition operation that adds to region j ($o^{n+1}_{U\&Me} = a_j$). In addition, there exists another region j' where the first uncommon operation in C_G is a move operation that moves a single resource from region j' to j (i.e, $o^{n+1}_G = b_{j'\to j} = s_{j'} \circ a_j$).

Let $R(n, C_{U\&Me}) = R(n, C_G) = L_{common}$ be the repositioned placement of applying all common operations $o^1_G = o^1_{U\&Me}, o^2_G = o^2_{U\&Me}, \ldots, o^n_G = o^n_{U\&Me}$ on the initial placement L. We denote the multiset corresponding to uncommon operation in G by $G^{n+1} = mult(o^{n+1}_G, o^{n+2}_G, \ldots)$. The first uncommon operation in C_G (o^{n+1}_G) is a move operation, and therefore, according to Claim 9.13, there is no profitable unary operation when o^{n+1}_G was selected. Formally, this means that there is no profitable unary operation in G^{n+1} over L_{common}. In particular, the operation that adds a resource to region j, a_j, is not profitable over L_{common} – a contradiction, as the first uncommon operation in $C_{U\&Me}$, i.e., $o^{n+1}_{U\&Me} = a_j$, is a U&Me operation, and thus it must be profitable over L_{common}.

□

g.6 Proof of Claim 9.17

Claim 9.17. *Suppose that $u_{j_{U\&Me}} = a_{j_{U\&Me}}$. Then there is no uncommon operation in C_G that adds to region $j_{U\&Me}$, i.e., $a_{j_{U\&Me}} \notin uncommon(G)$.*

Proof of Claim 9.17. Suppose otherwise, i.e., $a_{j_{U\&Me}} \in uncommon(G)$. We denote L_{common} as the placement of applying the common operations on C_G and $C_{U\&Me}$ on L_{init} (i.e., $L_{common} = (o^1_G \circ o^2_G \circ \ldots \circ o^n_G)(L_{init})$, where $o^i_G = o^i_{U\&Me}$ is a common operation for $1 \le i \le n$). From the greediness of the U&Me algorithm, the unary or the move operation with the highest profitability over L_{new_init} is the first uncommon operation $o^{n+1}_{U\&Me}$, which, by definition, adds to region $j_{U\&Me}$. In either of these cases, the best addition operation over L_{common} must add to region $j_{U\&Me}$[1]. Since the U&Me-like sequence $C_G = (o^1_G \circ o^2_G \circ \ldots)$ is a greedily ordered sequence (Observation 9.10), then the first uncommon operation o^{n+1}_G is either the unary or the move operation with the highest profitability in the multiset of uncommon operations $uncommon(G) = mult(o^{n+1}_G, o^{n+2}_G, \ldots)$. There are three cases, which in every case we show a contradiction:

[1]The move operation with the highest profitability is composed of the addition and subtraction operation with the highest profitability, according to Observation 9.9

o_G^{n+1} **is an addition or a move operation.** In both cases, o_G^{n+1} adds to the region with the highest addition profitability in $uncommon(G)$ over $L_{new_{init}}$ (Observation 9.9). Since the highest addition operation in $uncommon(G)$ over L_{common} adds to region $j_{U\&Me}$, then the first uncommon operation of C_G adds to region $j_{U\&Me}$ – a contradiction to our assumption that the first uncommon operation of C_G, o_G^{n+1}, does not add to region $j_{U\&Me}$.

o_G^{n+1} **is a subtraction operation, and** $o_{U\&Me}^{n+1}$ **is a unary operation.** In this case, $o_{U\&Me}^{n+1}$ is the addition operation $a_{j_{U\&Me}}$. According to the greediness of the U&Me algorithm, the profitability of $a_{j_{U\&Me}}$ over L_{common} is larger than the profitability of o_G^{n+1} over L_{common}. Thus, o_G^{n+1} is not the most profitable unary operation of $uncommon(G)$ – a contradiction, as o_G^{n+1} is, by definition, the most profitable unary operation in $uncommon(G)$.

o_G^{n+1} **is a subtraction operation, and** $o_{U\&Me}^{n+1}$ **is a move operation.** We prove in Corollary 9.14 that if the U&Me algorithm selects a move operation, then there is no profitable unary operation over the repositioned placement. In particular, there is profitable unary operation over L_{common}, and the subtraction operation o_G^t is not profitable – a contradiction, as all operations in the sequence C_G are assumed to have positive profitability.

□

g.7 Proof of Claim 9.18

Claim 9.18. *Suppose that $uncommon(G)$ contains an addition operation that adds a resource to region j_{rep}. Suppose that the first uncommon operation in the U&Me sequence adds to region $j_{U\&Me}$, while the first uncommon operation in G sequence does not. Let G' be the unary multiset G, where the addition to region j_{rep} is replaced by an addition to region $j_{U\&Me}$, i.e., $G' = (G \setminus \{a_{j_{rep}} = u_G\}) \uplus \{a_{j_{U\&Me}}\}$. Then G' is a valid multiset that satisfies $P(L^G) \leq P(L^{G'})$.*

Proof of Claim 9.18. By definition, $L^{G'}$ is simply L^G where a resource was moved from region j_{rep} to $j_{U\&Me}$, i.e., $L^G = b_{j_{rep} \to j_{U\&Me}}(L^{G'})$. Thus, it is sufficient to prove that $b_{j_{rep} \to j_{U\&Me}}$ has non-negative profitability over L^G.

According to Eq. (9.7), the move operation $b_{j_{rep} \to j_{U\&Me}}$ has non-negative profitability over $L^{G'}$ if and only if $g^{local}_{j_{U\&Me}}(L_{j_{U\&Me}} + 1) \geq g^{local}_{j_{rep}}(L_{j_{rep}})$. We denote for convenience L^{common} of the repositioned placement of applying all the common operations on L_{init} i.e., $L^{common} = R(n, C_G) = R(n, C_{U\&Me})$ where n is the number of common operations.

By definition, conducting the uncommon operations $o_G^{n+1}, o_G^{n+2}, \ldots$ over L^{common} derives L^G.

By definition, there exists an uncommon operation that adds to region j_{rep}. Since G contains non-canceling operations, then conducting the uncommon operations $o_G^{n+1}, o_G^{n+2}, \ldots$ over L^{common} will increase the number of resources in region j_{rep} by one resource at most, i.e.,

$$L^G_{j_{rep}} \geq L^{common}_{j_{rep}} + 1. \tag{g.8}$$

According to Claim 9.17, there the uncommon operations $o_G^{n+1}, o_G^{n+2}, \ldots$ do not add to region $j_{U\&Me}$ and therefore,

$$L^G_{j_{U\&Me}} \leq L^{common}_{j_{U\&Me}}. \tag{g.9}$$

Since the U&Me algorithm chooses to add to region $j_{U\&Me}$ in the t^{th} iteration, then the profitability of adding a resource to region $j_{U\&Me}$ is larger than the profitability of adding a resource to region j_{rep}, i.e.,

$$g^{local}_{j_{U\&Me}}(L^{common}_{j_{U\&Me}} + 1) \geq g^{local}_{j_{rep}}(L^{common}_{j_{rep}} + 1). \tag{g.10}$$

Since the marginal-differential functions Δg is monotonically non-increasing, we derive the desired result:

$$g^{local}_{j_{U\&Me}}(L^G_{j_{U\&Me}} + 1) \underbrace{\geq}_{\text{Eq. (g.9) and } \Delta g \text{ is monotonically non-increasing}} g^{local}_{j_{U\&Me}}(L^{common}_{j_{U\&Me}} + 1) \underbrace{\geq}_{\text{Eq. (g.10)}} g^{local}_{j_{rep}}(L^{common}_{j_{rep}} + 1)$$

$$\underbrace{\geq}_{\text{Eq. (g.8) and } \Delta g \text{ is monotonically non-increasing}} g^{local}_{j_{rep}}(L_{j_{rep}}). \tag{g.11}$$

□

g.8 Proof of Claim 9.19

Claim 9.19. *Suppose that uncommon(G) contains only subtraction operations. Suppose that the first uncommon operation in the U&Me sequence adds to region $j_{U\&Me}$. Suppose that the first uncommon operation in C_G subtracts from region j_{rep}. Let G' be the unary multiset G, where the subtraction of j_{rep} is replaced by an addition of $j_{U\&Me}$,*

i.e., $G' = (G \setminus \{s_{j_{rep}} = u_G\}) \uplus \{a_{j_{U\&Me}}\}$. *Then,* G' *is a valid multiset that satisfies* $P(L^G) \leq P(L^{G'})$.

Proof of Claim 9.19. We denote $C_{G'}$ to be the sequence C_G where the first uncommon operation, i.e., $o_G^{n+1} = s_{j_{rep}}$, is replaced by the unary operation $a_{j_{U\&Me}}$ (i.e., $o_{G'}^{n+1} = a_{j_{U\&Me}}$ and $o_{G'}^i = o_G^i$ for $i \neq n+1$). Of course, the multiset corresponding to the unary operations of G is G'. If we prove that the relative profit can only increase, i.e., $Rel(k, C_G) \leq Rel(k, C_{G'})$ for every k, then, according to Observation 9.8, we derive that $P(L^G) \leq P(L^{G'})$, as required.

Let $k \geq 1$ be a natural number. We denote $L^{k-1} = R(k-1, C_G)$ and $L'^{k-1} = R(k-1, C_{G'})$ be the repositioned placement of conducting the first $k-1$ operations of C_G and $C_{G'}$ over the initial placement L_{init}. Suppose that the number of common operations in C_G is n. Let L^{common} be the repositioned placement of conducting common operations over the initial placement L_{init}. There are three major cases:

If $k < n+1$**:** The first n operations in both C_G and $C_{G'}$ are the same operations. Thus, we derive that $L^{k-1} = L'^{k-1}$ and $o_G^k = o_{G'}^k$ for every $k < n+1$. Therefore $Rel(k, C_G) = \Delta(o_G^k, L^{k-1}) = \Delta(o_{G'}^k, L'^{k-1}) = Rel(k, C_{G'})$.

If $k = n+1$**:** First, since the first n operations in both C_G and $C_{G'}$ are the common operations, we derive that $L_n = L'_n = L^{common}$. Suppose, by way of contradiction, that $o_{U\&Me}^{n+1}$ is a move operation. During the move phase of the U&Me algorithm, there is no profitable unary operation over the repositioned placement (Corollary 9.14). Thus, there is no profitable unary operation over L^{common}. The first uncommon operation in C_G, i.e., o_G^{n+1}, is a subtraction operation, and thus it is not profitable over L^{common} – a contradiction that G contains only profitable operations. Since $o_{U\&Me}^{n+1}$ adds to region $j_{U\&Me}$, we derive that $o_{U\&Me}^{n+1} = a_{j_{U\&Me}}$ is a unary operation. According to the U&Me greediness, $o_{U\&Me}^{n+1} = a_{j_{U\&Me}}$ is more profitable than every other unary operation over L^{common}. This includes the first uncommon operation in C_G i.e., $o_G^{n+1} = s_{j_{rep}}$. Therefore,

$$Rel(k, C_G) = \Delta(o_G^{n+1}, L_n) \underbrace{=}_{o_{G'}^{n+1} = o_{U\&Me}^{n+1} = a_{j_{U\&Me}}} \Delta(s_{j_{rep}}, L^{common}) \leq$$
$$\Delta(a_{j_{U\&Me}}, L^{common}) = \Delta(o_{G'}^{n+1}, L'_n) = Rel(k, C_{G'}), \quad \text{(g.12)}$$

as required.

If $k > n+1$**:** We denoted $L^{k-1} = R(k-1, C_G)$ and $L'^{k-1} = R(k-1, C_{G'})$ be the repositioned placements of conducting the first $k-1$ operations of C_G and $C_{G'}$ over the initial placement L_{init}. All operations in both C_G and $C_{G'}$ are identical,

except for the $n+1$ operation. While $o^n_{G'}$ adds a resource to region $j_{U\&Me}$, o^n_G subtracts from region j_{rep}. Thus, we can derive the quantities of the repositioned placement $L^{k-1} = R(k-1, C_G)$ and L'^{k-1} as follows:

$$\begin{aligned} L^{k-1}_j &= L'^{k-1}_j \text{ (for every region } j \neq j_{U\&Me}, j_{rep}). \\ L^{k-1}_{j_{U\&Me}} &= L'^{k-1}_{j_{U\&Me}} - 1. \\ L^{k-1}_{j_{rep}} &= L'^{k-1}_{j_{rep}} - 1. \\ |L^{k-1}| &= |L'^{k-1}| - 2. \end{aligned}$$

Thus, we derive that

$$\begin{aligned} L^{k-1}_j &\leq L'^{k-1}_j \text{ (for every region } j). \\ |L^{k-1}| &\leq |L'^{k-1}|. \end{aligned} \tag{g.13}$$

Since $k > n+1$ we observed that: 1) The k^{th} operation in C_G and $C_{G'}$ are equal to each other ($o^k_G = o^k_{G'}$). 2) The k^{th} operation in C_G is an uncommon with the U&Me sequence $C_{U\&Me}$. Since the multiset of all uncommon operations $uncommon(G)$ contains only subtraction operations, then we derive that o^k_G is a subtraction operation that subtracts from some region j_0., i.e., $o^k_G = o^k_{G'} = s_{j_0}$.

Using the Eqs. (g.13), (9.6) and the fact that the marginal-differential function Δg is monotonically non-increasing, we derive the derived result:

$$\begin{aligned} Rel(k, C_G) = \Delta(o^k_G, L^{k-1}) &= \Delta(s_{j_0}, L^{k-1}) \\ &\underbrace{=}_{\text{Eq. (9.6)}} -\Delta g^{local}_{j_0}(L^{k-1}_{j_0}) - \Delta g^{global}(|L^{k-1}|) \\ &\underbrace{\leq}_{\text{Eq. (g.13) and } \Delta g \text{ are monotonically non-increasing}} -\Delta g^{local}_{j_0}(L'^{k-1}_{j_0}) - \Delta g^{global}(|L'^{k-1}|) \\ &\underbrace{=}_{\text{Eq. (9.6)}} \Delta(s_{j_0}, L'^{k-1}) \\ &= \Delta(o^k_{G'}, L'^{k-1}) = Rel(k, C_{G'}). \end{aligned} \tag{g.14}$$

□

g.9 Proof of Claim 9.20

Claim 9.20. *Suppose that uncommon(G) contains a unary operation u_G. Let G' be the unary multiset G where u is replaced by an addition of $j_{U\&Me}$, i.e., $G' = (G \setminus \{u_G\}) \uplus \{a_{j_{U\&Me}} = u_{U\&Me}\}$. Then, the potential of G is lexicographically greater than the potential of G'. More formally, we show that:*

(1) The number of operations not in the U&Me multiset cannot increase, i.e., $\pi_1(G) = |G \setminus G_{U\&Me}| \leq |G' \setminus G_{U\&Me}| = \pi_1(G')$.

(2) The number of uncommon operation is strictly reduced, i.e., $\pi_2(G) = |uncommon(G)| > |uncommon(G')| = \pi_2(G')$.

Proof of Claim 9.20.

Proof of (1):

We will prove that replacing u with the addition of $a_{j_{U\&Me}}$ cannot decrease the first potential. Formally, we will show that

$$G' \setminus G_{U\&Me} = (G \setminus G_{U\&Me}) \setminus \{u\}, \tag{g.15}$$

this will eventually imply that

$$\pi_1(G') = |G' \setminus G_{U\&Me}| \leq |G \setminus G_{U\&Me}| = \pi_1(G). \tag{g.16}$$

Suppose that a unary operation u_0 appears $x_G, x_{G'}$ and $x_{U\&Me}$ times in the multisets G, G' and $G_{U\&Me}$, respectively. We will prove that the multiplicity of u_0 in $G' \setminus G_{U\&Me}$ is equal to u multiplicity in $(G \setminus G_{U\&Me}) \setminus \{u\}$.

In our proof, we will now use the well-known fact that if an element x appears x_A and x_B times in multisets A and B, then the element x appears $x_A + x_B$ and $\max(0, x_A - x_B)$ times in $A \uplus B$ and in $A \setminus B$, respectively. We refer to [79] for more information. Note we will divide our proof into three cases:

If $u_0 \neq u, a_{j_{U\&Me}}$: If $u_0 \neq u, a_{j_{U\&Me}}$, then u_0 appears $x_G = x_{G'}$ in G and G', and also $\max(0, x_G - x_{U\&Me}) = \max(0, x_{G'} - x_{U\&Me})$ in both $G' \setminus G_{U\&Me}$ and in $(G \setminus G_{U\&Me}) \setminus \{u\}$.

If $u_0 = u$: The operation u appears $\max(0, x_{G'} - x_{U\&Me})$ in $G' \setminus G_{U\&Me}$, and $\max(0, t-1)$ in $(G \setminus G_{U\&Me}) \setminus \{u\}$, where $t = \max(0, x_G - x_{U\&Me})$. However, both expressions are equal to each other, as operation u in G was replaced, i.e., $x_{G'} = x_G - 1$.

If $u_0 = a_{j_{U\&Me}}$: Suppose operation $a_{j_{U\&Me}}$ appears d times in the common prefix sequence C_{com}. Then, according to Claim 9.17, $a_{j_{U\&Me}}$ does not appear in the multiset of uncommon operation $uncommon(G)$. Thus, $a_{j_{U\&Me}}$ appears exactly $x_G = d$ times in G. Since G' is the multiset G where we replaced the operation u by $a_{j_{U\&Me}}$, then $a_{j_{U\&Me}}$ appears $x_{G'} = d + 1$ times in G'. The U&Me multiset contains all common operations in C_{com}. In addition, the first uncommon prefix sequence in $U\&Me$ is $a_{j_{U\&Me}}$. Thus the U&Me multiset $G_{U\&Me}$ contains at least $x_{U\&Me} \geq d + 1$ times the element $a_{j_{U\&Me}}$.

The operation $a_{j_{U\&Me}}$ appears in $G' \setminus G_{U\&Me}$ and in $G \setminus G_{U\&Me} \setminus \{u\}$ exactly $\max(0, x_{G'} - x_{U\&Me}) = \max(0, x_G - x_{U\&Me}) = 0$ times. As required.

Proof of (2): Let $C^G_{com} = (o^1_G, o^2_G, \ldots, o^n_G)$ be the common prefix sequence of C_G with $C_{U\&Me}$ (i.e., $o^i_G = o^{U\&Me}_G$). Let $com(G)$ and $com(G')$ denote the unary multiset of **common** operations in G and G', respectively. The second potential function of both G and G' are equal, respectively, to the number of operations in G and G' that are not common, i.e., $\pi_2(G) = |uncommon(G)| = |G| - |com(G)|$ and $\pi_2(G') = |uncommon(G')| = |G'| - |com(G')|$. Since the number of operations in G' and G' are equal to each other, i.e., $|G| = |G'|$, it is sufficient to prove that $|com(G')| > |com(G)|$ to show the correctness of the claim. We will show that $com(G)$ is a proper subset of $com(G')$.

First, we will show that $com(G)$ is a subset of $com(G')$ (i.e., $com(G) \subset com(G')$). Since u is an uncommon operation (i.e., $u \notin com(G)$) then $com(G)$ remains a subset of $G' = (G \setminus \{u\}) \uplus \{a_{j_{U\&Me}}\}$. Let $C_{G'}$ be the U&Me-like sequence corresponding to G'. We will show by a simple induction that for every $1 \leq i \leq n$ the $(i-1)^{th}$ redeposition placement of $C_{G'}$ is $R(i-1, C_{G'}) = R(i-1, C_{U\&Me})$ and the i^{th} operation of C_G is a common operation, i.e., $o^i_{G'} = o^i_{G'}$:

For $i = 1$: – We have $R(0, C_{G'}) = L_{init} = R(0, C_{U\&Me})$. The multiset G' contains the common operation $o^1_G = o^1_{U\&Me}$, which has the highest profitability over L_{init}. Thus, the first operation in $C_{G'}$ is $o^1_{G'} = o^1_{U\&Me}$.

$i - 1 \to i$: Since the first $i - 1$ operations are common operations, then we derive that $R(i-1, C_{G'}) = R(i-1, C_{U\&Me})$. The i^{th} U&Me operation $o^i_{U\&Me} = o^i_G$ is a common operation in G' (as G' contains the common multiset $com(G)$) which has the highest profitability over $R(i-1, C_{G'}) = R(i-1, C_{U\&Me})$. Thus, we derive that the i^{th} operation in $C_{G'}$ is then $o^i_G = o^i_{U\&Me} = o^i_{G'}$.

Thus, we have shown that the unary multiset of common operations of G are a subset of the unary multiset common operations in G', i.e., $com(G) \subset com(G')$. Next we will

show that $com(G) \neq com(G')$. This implies that $com(G)$ is a proper subset of $com(G')$ and the proof is completed.

We will show that the $n+1$ operation of G' (n is the length of C_{com}), is either a move or an addition operation that adds to region $j_{U\&Me}$. Let ${G'}_n = G' \setminus com(G)$ denote the unary multiset of operations of $o_{G'}^{n+1}, o_{G'}^{n+2}, \ldots$. When considering the $n+1$ operation in $C_{G'}$, $o_{G'}^{n+1}$, it is, by definition, either the unary or move operation with the highest profitability in ${G'}_n$ over the repositioned placement $R(n, C_{G'})$. The first n operations of $C_{G'}$ are common with $C_{U\&Me}$, and therefore, applying these operations over the initial placement L_{init} derives the same placement, i.e., $R(n, C_{G'}) = R(n, C_{U\&Me})$.

Suppose that the operation $a_{j_{U\&Me}}$ appears x times in G. Then it appears $x+1$ times in G'. According to Claim 9.17, since there is no uncommon operation that adds to region $j_{U\&Me}$, then the common operations $o_G^1 = o_{G'}^1, o_G^2 = o_{G'}^2, \ldots, o_G^n = o_{G'}^n$ add to region $j_{U\&Me}$ exactly x times. Thus, ${G'}_n = G' \setminus com(G)$ contains exactly once the operation $a_{j_{U\&Me}}$.

By its definition, the first uncommon operation of the U&Me sequence with C_G, $o_{U\&Me}^{n+1}$, adds to region $a_{j_{U\&Me}}$. That means, $o_{U\&Me}^{n+1}$ is either the unary operation or the move operation with the highest profitability over the repositioned placement $R(n, C_{G'}) = R(n, C_{U\&Me})$. We show that in the following cases that $o_{G'}^{n+1}$ cannot be a subtraction operation:

$o_{G'}^{n+1}$ is a subtraction operation, and $o_{U\&Me}^{n+1}$ is a unary operation. In this case, $o_{U\&Me}^{n+1}$ is the addition operation $a_{j_{U\&Me}}$. According to the greediness of the U&Me algorithm, the profitability of $a_{j_{U\&Me}}$ over L_{common} is larger than the profitability of o_G^{n+1} over $R(n, C_{G'}) = R(n, C_{U\&Me})$. Thus, $o_{G'}^{n+1}$ is not the most profitable unary operation of $R(n, C_{G'}) = R(n, C_{U\&Me})$ – a contradiction, as $o_{G'}^{n+1}$ is, by definition, the most profitable unary operation in ${G'}_{n-1}$ over $R(n, C_{G'})$.

$o_{G'}^{n+1}$ is a subtraction operation, and $o_{U\&Me}^{n+1}$ is a move operation. According to Corollary 9.14, if the U&Me algorithm selects a move operation over $R(n, C_{U\&Me})$, then there is no profitable unary operation over $R(n, C_{U\&Me})$. Thus, the subtraction operation $o_{G'}^{n+1}$ is not profitable over $R(n, C_{U\&Me}) = R(n, C_{G'})$. Since $C_{G'}$ is a U&Me-like sequence, then $o_{G'}^{n+1}, o_{G'}^{n+2}, \ldots$ are either subtraction or addition operations. However, this is not possible, as $o_{G'}^{n+1}$ is a subtraction operation and the unary multiset of $o_{G'}^{n+1}, o_{G'}^{n+2}, \ldots$, is ${G'}_n$, which contains the addition operation $a_{j_{U\&Me}}$.

Thus $o_{G'}^{n+1}$ is either an addition or a move operation. Since $a_{j_{U\&Me}} \in {G'}_n$ is the addition operation with the highest profitability over $R(n, C_{G'}) = R(n, C_{U\&Me})$, then $o_{G'}^{n+1}$ adds to region $a_{j_{U\&Me}}$. Since the $(n+1)^{th}$ operation in both $C_{G'}$ and $C_{U\&Me}$ add to region

$j_{U\&Me}$, then $a_{j_{U\&Me}}$ is not in the unary multiset of uncommon operations in $C_{G'}$, i.e., $uncommon(G')$ (See Remark 9.2).

As we observed, if the operation $a_{j_{U\&Me}}$ appears x times in G, then it appears x times in $com(G)$. Since $com(G') = G' \setminus uncommon(G')$ contains $a_{j_{U\&Me}}$ exactly $x + 1$ times, then, in particular, $com(G) \neq com(G')$. Thus we have shown that $com(G)$ is a proper subset of G'.

□

Appendix h

The Proof of Lemma 9.4

Lemma 9.4. *Let L be a placement, and let $C_{U\&Me}$ be the U&Me sequence over L. Then $C_{U\&Me}$ is non-canceling.*

Proof of Lemma 9.4. We denote by $C_{U\&Me} = (o^1, o^2, \ldots)$ the U&Me sequence. For the sake of contradiction, assume that U&Me is canceling, i.e., the U&Me sequence adds to some region j_{org} and subtracts from region j_{org}. We call region j_{org} the **original** region. Suppose that operations $o^{i_{add}}$ and $o^{i_{sub}}$ are respectively adding to and removing from the original region j_{org} a single resource.

Without loss of generality we can make the following assumptions in our proof: 1) The addition operation appears in the U&Me sequence before the subtraction operation, i.e., $i_{add} \leq i_{sub}$. The proof is similar to the case where the subtraction operation appears before the addition operation. 2) The indices i_{add} and i_{sub} are of the shortest-gap canceling operations. That means that if o^{t_1} and o^{t_2} are canceling operations in the U&Me sequence, then $|t_1 - t_2| \geq |i_{add} - i_{sub}|$. In particular, these assumptions imply the following observations:

Observatiton 1: the operations $(o^{i_{add}+1}, o^{i_{add}+2}, \ldots, o^{i_{sub}})$ do not cancel each other out.

Observatiton 2: the operations $(o^{i_{add}+1}, o^{i_{add}+1}, \ldots, o^{i_{sub}-1})$ neither add nor subtract a resource from the original region j_{org}.

We denote $R(i_{add} - 1)$ as the repositioned placement after conducting the first $i_{add} - 1$ operations. We will see how to use Claim h.2 to derive an alternative region j_{alt} such that $\Delta(a_{j_{alt}}, R(i_{add}-1)) > \Delta(a_{j_{org}}, R(i_{add}-1))$, i.e., adding to the alternative region j_{alt} is more profitable than adding to the original region j_{org}. This, of course, contradicts

U&Me greediness, where in each iteration it selects the most profitable operation[1]; in particular, U&Me cannot add to the original region j_{org} in iteration i_{add}.

To show the contradiction, we use the following two claims that are essential building blocks to prove Lemma 9.4. The proof of the claims are quite straightforward, and all are based on three facts: 1) Eqs. (9.5)- (9.7). 2) The marginal-differential functions Δg are monotonically non-increasing. 3) U&Me selects only profitable operations. We will later see how these claims imply the contraction.

Claim h.1. *Let $C_{U\&Me} = (o^1, \ldots, o^n)$ be the U&Me sequence. Suppose that for $1 \leq i_1 < i_2 \leq n$ operations $o^{i_1}, o^{i_1+1}, \ldots, o^{i_2}$ are all either addition or subtraction operations. Then o^{i_2} is profitable over the repositioned placement $R(i_1-1)$ corresponding to the first $i_1 - 1$ operations.*

Note that while Observation 9.6 states that o^{i_2} is profitable over the repositioned placement $R(i_2 - 1)$, it is not obvious that o^{i_2} is profitable over a previous repositioned placement $R(i_1 - 1)$.

Proof of Claim h.1. Without loss of generality assume that $i_1 = 1$, operations $o^1, o^2, \ldots, o^{i_2}$ are all addition operations, and o^{i_2} adds to region j, i.e., $o^{i_2} = a_j$. We need to show that the profitability of o^{i_2} over $R(0) = L$, which according to Eq. (9.5) equals $\Delta(a_j, L) = \Delta g_j^{local}(L_j) + \Delta g^{global}(|L|)$, is positive. Suppose that $o^1, o^2, \ldots, o^{i_2-1}$ are adding $\delta_j \geq 0$ resources to region j and $\delta \geq 0$ resources in all regions. That means, the repositioned placement after applying these operations, $R(i_2-1)$, contains $\delta_j + L_j$ resources in region j and $|L| + \delta$ in all regions. The profitability of o^{i_2} over $R(i_2 - 1)$ is positive (Observation 9.6). Therefore, $\Delta(R(i_2-1), a_j) = \Delta g_j^{local}(L_j + \delta_j) + \Delta g^{global}(|L| + \delta) > 0$. Finally, the marginal-differential functions Δg are monotonically non-increasing and thus

$$\Delta(a_j, L) = \Delta g_j^{local}(L_j) + \Delta g^{global}(|L|) \geq \Delta g_j^{local}(L_j + \delta_j) + \Delta g^{global}(|L| + \delta) > 0, \quad \text{(h.1)}$$

as required. □

Claim h.2. *Let $C_{U\&Me} = (o^1, \ldots, o^n)$ be the U&Me sequence. Suppose that there are regions j_1, j_2 that satisfy the following conditions: 1) Operation o^{i_1} adds a resource to region j_1, and does not subtract from region j_2. 2) Operations $o^{i_1+1}, o^{i_1+2}, \ldots, o^{i_2-1}$ neither add to nor subtract from either region j_1 or region j_2. 3) The move operation $b_{j_1 \to j_2} = s_{j_1} \circ a_{j_2}$ is profitable over a repositioned placement $R(t-1)$, where $i_1 < t \leq i_2$. Then, $\Delta(a_{j_2}, R(i_1 - 1)) > \Delta(a_{j_1}, R(i_1 - 1))$.*

[1] By Observation 9.1, if U&Me selects a move operation then it is composed of the addition and subtraction operations with the highest profitability.

Proof of Claim h.2. For the sake of clarity, we denote $L = R(i_1 - 1)$ and $L' = R(t-1)$. By definition of repositioned placement, conducting the composition of operations $o^{i_1}, o^{i_1+1}, \ldots, o^{t-1}$ over $L = R(i_1 - 1)$ derives the placement $L' = R(t)$ (i.e., $(o^{i_1} \circ o^{i_1+1} \circ \ldots \circ o^{t-1})(L) = L'$). Since operations $o^{i_1}, o^{i_1+1}, \ldots o^{i_2}$ add a single resource to region j_1 and they do not subtract resources from region j_2, we derive that

$$\begin{aligned} L'_{j_1} &= L_{j_1} + 1, \\ L'_{j_2} &\geq L_{j_2}. \end{aligned} \tag{h.2}$$

The profitability of $b_{j_1 \to j_2}$ over $R(t-1) = L'$, which according to Eq. (9.7) is $g^{local}_{j_2}(L'_{j_2} + 1) - g^{local}_{j_2}(L'_{j_1})$, is positive, i.e.,

$$g^{local}_{j_2}(L'_{j_2} + 1) > g^{local}_{j_1}(L'_{j_1}) \tag{h.3}$$

Thus, we derive that,

$$g^{local}_{j_2}(L_{j_2} + 1) \underbrace{\geq}_{\text{Eq. (h.2) and } \Delta g \text{ is monotonically non-increasing}} g^{local}_{j_2}(L'_{j_2} + 1) \underbrace{>}_{\text{Eq. (h.3)}} g^{local}_{j_1}(L'_{j_1}) \underbrace{=}_{\text{Eq. (h.2)}} g^{local}_{j_1}(L_{j_1} + 1),$$

i.e., the local value of adding a resource to region j_2 is strictly larger than the local value of adding a resource to region j_1. By our notation, $R(i_1 - 1) = L$, and therefore

$$\Delta(a_{j_2}, R(i_1-1)) = g^{local}_{j_2}(L_{j_2}+1) + g^{global}_{j_2}(|L|) > g^{local}_{j_1}(L_{j_1}+1) + g^{global}_{j_1}(|L|) = \Delta(a_{j_1}, R(i_1-1)),$$

as required. □

To show the contraction, suppose that operations $o^{i_{add}}$ and $o^{i_{sub}}$ are both unary operations. The case where $o^{i_{sub}}$ is a move operation, and $o^{i_{add}}$ can be either a move or a unary operation, is given in the next paragraph. Let $o^t = a_{j_{alt}}$ be the last addition operation in $(o^{i_{add}}, o^{i_{add}+1}, \ldots, o^{i_{sub}-1})$ (i.e., $(o^{t+1}, o^{t+2}, \ldots, o^{i_{sub}-1})$ are all subtraction operations) that adds to region j_{alt}. Then, the operations $o^{t+1}, o^{t+2}, \ldots, o^{i_{sub}}$ are all subtraction operations. Therefore, according to Claim h.1, the operation $o^{i_{sub}} = s_{j_{org}}$ is profitable over the repositioned placement $R(t)$. The operation $o^t = a_{j_{alt}}$ is profitable over $R(t-1)$ (Observation 9.6) and therefore the composite of profitable operations $a_{j_{alt}} \circ s(j) = b_{j_{alt} \to j_{org}}$ is a profitable operation over $R(t-1)$ (Observation 9.7). Finally, we use Claim h.2 to derive that $\Delta(a_{j_{alt}}, R(i_{add} - 1)) > \Delta(a_{j_{org}}, R(i_{add} - 1))$. adding

to the alternative region j_{alt} is more profitable than adding to original region j_{org}, i.e., $\Delta(a_{j_{alt}}, R(i_{add}-1)) > \Delta(a_{j_{org}}, i_{add}-1)$.

Now, suppose that $o^{i_{sub}}$ is a move operation, and $o^{i_{add}}$ can be either a move or a unary operation. Operation $o^{i_{sub}}$ is a move operation, and therefore there exists region j_{alt} such that $o^{i_{sub}}$ moves a resource from the original region j_{org} to the alternative region j_{alt}, i.e., $o^{i_{sub}} = b_{j_{org}\to j_{alt}} = a_{j_{alt}} \circ s_{j_{org}}$.

Suppose that $o^{i_{add}} = b_{j_{alt}\to j_{org}} = a_{j_{org}} \circ s_{j_{alt}}$ is a move operation that subtracts from region j_{alt}, i.e., $o^{i_{add}} = b_{j_{alt}\to j_{org}} = a_{j_{org}} \circ s_{j_{alt}}$. By the following claim, we derive that either $o^{i_{add}} = b_{j_{alt}\to j_{org}}$ or its revere operation $o^{i_{sub}} = b_{j_{org}\to j_{alt}}$ is not profitable.

Claim h.3. *Suppose that $o^{i_{add}}$ is a move operation that moves a resource from region j_{alt} to region j_{org} (i.e., $o^{i_1} = b_{j_1\to j_2}$), and o^{i_2} is the reverse operation that moves a resource from region j_2 to region j_1 (i.e., $o^{i_2} = b_{j_2\to j_1}$). Suppose that operations $o^{i_1+1}, o^{i_1+2}, \ldots, o^{i_2-1}$ neither add to nor subtract from either region j_1 or region j_2. Then, the relative profitability of o^{i_1} is equal to minus the relative profitability of o^{i_2}, i.e.,*

$$Rel(i_1, C) = -Rel(i_2, C). \tag{h.4}$$

Proof. Let $L = R(i_1 - 1, C)$ and $L' = R(i_2 - 1, C)$ be the repositioned placement of the first i_1-1 and i_2-1 operations of C over L_{init}, respectively. By the definition of the repositioned placement, conducting the operations $o^{i_1}, o^{i_1+1}, o^{i_1+2}, \ldots, o^{i_2-1}$ over L is equal to L', i.e., $(o^{i_1} \circ o^{i_1+1} \circ o^{i_1+2} \circ \ldots o^{i_2-1})(L) = L'$. The operations $o^{i_1+1}, o^{i_1+2}, \ldots, o^{i_2-1}$ neither add to nor subtract from either region j_1 or region j_2 and operation o^{i_1} moves a resource from region j_1 to region j_2. Therefore, the quantities of regions j_1 and j_2 in L and L' are equal to:

$$\begin{aligned} L'_{j_1} &= L_{j_1} - 1 \\ L'_{j_2} &= L_{j_2} + 1. \end{aligned} \tag{h.5}$$

By its definition, the relative profitability of o^{i_1} and o^{i_2} is equal, respectively, to the profitability of operations $o^{i_1} = b_{j_1\to j_2}$ and $o^{i_2} = b_{j_2\to j_1}$ over the repositioned placements $L = R(i_1 - 1, C)$ and $L' = R(i_2 - 1, C)$, respectively. According to Eq (9.7), the relative profitability of o^{i_1} is $Rel(i_1, C) = g^{local}_{j_1}(L_{j_2} + 1) - g^{local}_{j_2}(L_{j_1})$ and the relative profitability of o^{i_2} is $Rel(i_2, C) = g^{local}_{j_2}(L'_{j_1} + 1) - g^{local}_{j_1}(L'_{j_2})$. By Eq. (h.5) we derive that $Rel(i_1, C) = -Rel(i_2, C)$, i.e., the desired result. □

Thus, we show that $o^{i_{add}}$ cannot be a move operation that subtracts from region j_{alt}. The operations $o^{i_{add}}, o^{i_{add}+1}, \ldots o^{i_{sub}-1}$ satisfy the first two conditions of Claim h.2. In

addition, $o^{i_{sub}}$ is profitable over $R(i_{sub}-1)$, and satisfies the third condition of Claim h.2. Therefore, we derive that $\Delta(a_{j_{alt}}, R(i_{add}-1)) > \Delta(a_{j_{org}}, R(i_{add}-1))$, as required.

□

Appendix i

The Hardness of the Fair Christmas Game Problem

In this chapter, we prove Theorem 10.4 from Section 10.2, using the Exact Cycle Sum problem.

Definition i.1. Let $G = (V, E)$ be a directed graph. A set of cycles $T = \{C_1, \ldots, C_n\}$ in G is called a set of **disjoint-vertex cycles** if no vertex is shared by different cycles.

The Exact Cycle Sum problem is discussed in [19] and defined as followed:

Exact Cycle Sum	
Input:	A parameter k, a directed graph G.
Problem:	Is there a set $T = \{C_1, C_2, \ldots C_n\}$ of disjoint-vertex cycles in G, which contains exactly k edges?

We show that the Fair Christmas Game Problems is as hard as the Exact Cycle Sum (in the sense that if one solves the former, one solves the latter as well), i.e., we will prove Theorem 10.4 from Section 10.2.

Theorem 10.4. *There is a polynomial reduction from the Cycle Sum Problem to the Fair Christmas Game Problem.*

Proof of Theorem 10.4. Let k, $G = (V, E)$ be an instance of the Exact Cycle Sum problem. We construct a corresponding Christmas Game as follow: For every vertex $v \in V$ we add two players v_{in}, v_{out} to our game. The friend list of player v_{in} is only v_{out}, and the friend list of player v_{out} are players u_{in} where $(v, u) \in E$. We will prove that there is

a set T of disjoint-vertex cycles containing exactly k edges, iff there is a Fair Christmas Game with $2k$ gifts sent.

Suppose G has a disjoint-vertex cycle set $T = \{C_1, C_2, \dots C_n\}$. We denote the union of these cycles by $G' = \bigcup_i C_i = (V', E')$. We define a Fair Christmas Game where player v_{in} gives a gift to v_{out} iff $v \in V'$ and player v_{out} gives a gift to player u_{in} iff $(v, u) \in E'$. The game is fair, as every player v_{in}, v_{out} sends and receives one gift if $v \in V'$; otherwise, v_{in}, v_{out} sends and receives zero gifts.

We define a **game cycle** $C^{game} = (u^1, u^2, \dots u^m, u^1)$ if player u^i sends a gift to player u^{i+1} and player u^m sends a gift to player u^1. Note that a cycle $C_i = (v^1, v^2, \dots, v^m, v^1)$ with m edges corresponds to a game cycle with $2m$ gifts of the form $C^{game} = (v^1_{in}, v^1_{out}, v^2_{in}, v^2_{out}, \dots, v^m_{in}, v$ Thus, if S contains k edges, then its corresponding Fair Christmas Game sends exactly $2k$ gifts.

Suppose there is a Fair Christmas Game with $2k$ gifts sent. We define V_{game} as the set of players sending or receiving one or more gifts. For every player $v_{in} \in V_{game}$, we can construct, by induction, a unique game cycle of the form $C^{game} = (v^1_{in}, v^1_{out}, v^2_{in}, v^2_{out}, \dots,$ $v^m_{in}, v^m_{out}, v^1_{in})$ where $2m$ gifts are sent and it contains player v^{in}. This game cycle corresponds to a cycle $C = (v^1, v^2, \dots)$ in G, with m edges. Thus, $C^{game}_1, C^{game}_2, \dots C^{game}_n$ are the game cycles of the fair game, where $2k$ gifts are sent. Their corresponding cycles are $C_1, C_2, \dots C_n$ with a total number of k edges. Note that the cycles $C_1, C_2, \dots C_n$ have disjoint vertices since, by the construction, the game cycles are unique to every player. □

Appendix j

The SCC algorithm – The Minimum-Length Negative Cycle and The Bipartite-like Graph G_f^B

j.1 Minimum-Length Negative Cycle (MLnC) Algorithm

j.1.1 Preliminaries: the Bellman-Ford algorithm

Given a non-negative integer l, the Bellman-Ford algorithm can be use to find the shortest paths that use l edges at most. This is formulated by the following theorem:

Theorem j.1. *Let $G = (V, E)$ be a weighted graph, $v_1, v_2 \in V$ two vertices, and l a number. Then, by running l iterations of Bellman-Ford, one can retrieve the shortest path between v_1 and v_2 of l edges at most. Moreover, this can be done in $O(l|V||E|)$ time.*

The correctness of Theorem j.1 is proven similarly to Lemma 24.2 in [72].

j.1.2 The Minimum-Length negative Cycle (MLnC) algorithm

To find the minimum-length negative cycle, we present the MLnC algorithm. Let $G = (V, E)$ be a general directed graph with a weight function w defined over the graph edges E. MLnC uses a matrix $A(l) = (a_i^j(j))$, where $a_i^j(l)$ is the cost of the shortest

path between $i \in V$ and $j \in V$, among all shortest paths with l edges. One can compute the matrix $A(l)$, using dynamic programming, as presented by following equations:

$$\begin{aligned} a_i^j(1) &= \begin{cases} w(i,j) & \text{when } (i,j) \in E. \\ \infty & \text{otherwise.} \end{cases} \\ a_i^j(l) &= \min_{k|(k,j)\in E}(a_i^k(l-1) + w(k,j)) \text{ for } l \geq 2. \end{aligned} \tag{j.1}$$

Given a vertex v and a number l, the element $a_i^i(v)$ represents the weight of the shortest cycle of length l that contains v. Thus, using the definition of the minimum-length negative cycle, we derive that the following properties hold:

Observation j.2. *A graph $G = (V, E)$ does not contain negative cycles iff for every number $1 \leq l \leq |V|$ the diagonal of $A(l)$ is non-negative, i.e., $a(l)_v^v \geq 0$ for every vertex v.*

Observation j.3. *Let $G = (V, E)$ be a graph that contains at least one negative cycle. Suppose that the minimum-length negative cycle contains l_{min} edges. Then, the following properties hold:*

- *For every integer $l < l_{min}$, the diagonal of $A(l)$ is non-negative, i.e., $a(l)_v^v \geq 0$ for every vertex v.*
- *Suppose $v_{min} = \arg\min_v a_v^v(l_{min})$ is the minimal element in the diagonal of $A(l_{min})$. Then, v_{min} is a vertex in the minimum-length negative cycle and $a_{v_{min}}^i(v_{min})(l_{min})$, which is the weight of the minimum-length negative cycle, is negative.*

Given a graph $G = (V, E)$, the MLnC algorithm runs as follows: (1) Compute iteratively the elements of $A(l)$, according to Eq. (j.1), until either the diagonal of $A(l)$ contains negative elements. If the algorithm runs for more than $|V|$ iterations, then the graph G does not contain any negative cycle (according to Observation j.2), and the algorithm terminates. Suppose that the algorithm runs for $l_{min} \leq |V|$ iterations. (2) Select the minimal element in the diagonal, i.e., $v_{min} = \arg\min_v a_v^v(l_{min})$. According to Observation j.3, the minimum-length negative cycle is of length l_{min} and contains the vertex v_{min}. (3) Use the Bellman-Ford algorithm as described in Theorem j.1, and find the shortest path form v_{min} to itself that uses l_{min} edges, i.e., the minimum-length negative cycle.

Give a matrix $A(l)$, computing the element $a_i^j(l+1)$ in the matrix $A(l+1)$ takes $O(d_j)$ operations, where d_j is the outgoing degree of vertex j. Thus, every iteration of Step (1) takes $O(\sum_{i\in V}\sum_{j\in V} d_j) = O(|V| \cdot |E|))$. Therefore, Step (1) takes $O(|V|^2 \cdot |E|)$ in case the graph G contains a negative cycle, and $O(l_{min} \cdot |V| \cdot |E|)$ otherwise. Step (2) can be

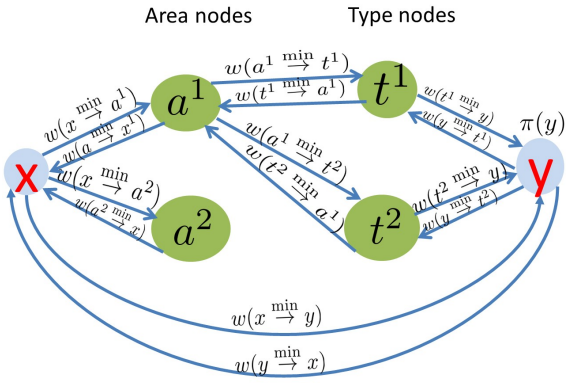


FIGURE J.1: The bipartite-like graph G_f^B.

implemented in $O(|V|)$ and Step (3) takes $O(l_{min} \cdot |V| \cdot |E|)$ according to Theorem j.1. Thus, we draw the following conclusion:

Corollary j.4. *MLnC determines whether a graph G contains a negative cycle. If such cycle exists – it computes the minimal-length negative cycle in $O(l_{min}|V||E|)$, where l_{min} is the number of edges in the negative cycle. Otherwise - the running time is $O(|V|^2|E|)$.*

Remark j.1. A related problem to the minimum-length negative cycle problem is that of finding a simple negative cycle of minimum cost, where the cost of a cycle is the sum of its edge costs. This problem was proved to be NP-complete (See [70]).

j.2 The Bipartite-Like Graph

The **bipartite-like graph**, denoted as G_f^B, is a directed graph constructed by removing the edges in the four-layer residual graph G_f^4. The graph is composed of the following four layers: the first layer comprises the source node x, the second layer comprises area nodes a^j, the third layer comprises resource type nodes t^i, and the last layer comprises the sink node y. As opposed to the bipartite-like graphs depicted in Sections 7.8, 6.3, we connect the source x to the sink y and the sink y to the source x. We do not use the node potentials as described in Chapter 7.8, which cannot be used in Cycle-Canceling algorithms (which includes SCC).

Between every two nodes v and u of successive layers in G_f^4, there exists multiple edges. Let $e_{min}(u, v)$ be the edge with minimum weight in the four-layer residual graph G_f^4 between u and v, and $w_f(e_{min}(u, v))$ the weight of this edge in G_f^4. Then, in the bipartite-like graph we connect a single edge between v and u, and the weight of that edge is equal to $w_f(v \to u) = w_f(e_{min}(u, v)))$.

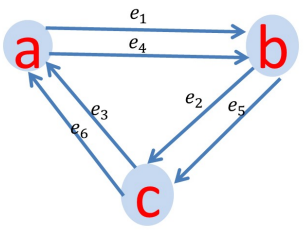


FIGURE J.2: A multigraph that contains two different cycles spanning the same vertices.

j.3 Path Composition

Remark j.1. In multigraphs, we denote cycles and paths by their edges and not by their vertices. For example, the cycle $C = (e_1, e_2, e_3)$ and $C' = (e_4, e_5, e_6)$ are two different cycles in Figure j.2, connecting vertices a, b and c.

We next show that the weight of the minimum-length negative cycle in both the four-layer residual graph G_f^4 and bipartite-like graph G_f^B are equal to each other:

Claim j.5. *Let f be a flow that the SCC algorithm calculates in its j^{th} iteration over the four-layer residual graph G_f^4. The following properties hold:*

(a) If the minimum-length negative cycle in the four-layer residual graph G_f^8 connects vertices $v_1, v_2, \ldots, v_n = v_0$, then the cycle composed of the edges with minimum weight between v_i and v_{i+1}, i.e., the cycle $C = (e_{min}(v_0, v_1), e_{min}(v_1, v_2), \ldots, e_{min}(v_{n-1}, v_n))$ is also the minimum-length negative cycle.

(b) A cycle $C = (v_0, v_1, \ldots, v_n = v_0)$ is the minimum-length negative cycle in the bipartite residual graph G_f^B if and only if $\hat{C} = (e_{min}(v_0, v_1), e_{min}(v_1, v_2), \ldots, e_{min}(v_{n-1}, v_n))$ is a minimum-length negative cycle in G_f^4.

Proof of Claim e.2.

Proof of (a): Let C' be the minimum-length negative cycle that connects vertices $v_1, v_2, \ldots, v_n = v_0$. The weight of every edge in C cannot be larger than the corresponding edge in C', and thus the weight of C cannot be larger than the weight of C'. Since C' is the minimum-length negative cycle, then C is also a negative cycle. Let C'' be a negative cycle. By the definition of a minimum-length negative cycle, there can be two cases: 1) The length of the cycle C'' is strictly larger then the length of C', and thus the length of the cycle C'' is strictly larger then the length of C (which is equal to the

length of C''). 2) The length of C'' is equal to the length of C' (and the length of C), and the weight of C'' is smaller than or equal to the weight of C'. Since the weight of C' is smaller than or equal to the weight of C, then the weight of C'' is smaller than or equal to the weight of C. Thus, we have proved that C is the minimum-length negative cycle.

Proof of (b):
$\Rightarrow$: Suppose otherwise. According to (a), there exists a negative cycle $\hat{C}' = (e_{min}(v'_0, v'_1), e(v'_1, v'_2), \ldots, e_{min}(v'_{n'-1}, v'_{n'}))$ which is "better" than $\hat{C}$, i.e., $n > n'$, or the length of $\hat{C}$ and $\hat{C}'$ are equal to each other but the weight of $\hat{C}'$ is smaller than the weight of $\hat{C}$. The length and weight of C and C' are equal, respectively, to the length and weight of $\hat{C}$ and $\hat{C}'$. Thus, we derive that C cannot be the minimum-length negative cycle in G_f^B, as the cycle C' is a negative cycle that either is shorter (in length) than C, or the length of C and C' are equal to each other, but the weight of C' is smaller than the weight of C – a contradiction.

$\Leftarrow$: Suppose otherwise. There exists a negative cycle $C' = (v'_0, v'_1, \ldots, v'_{n'})$ which is better than C, i.e., $n > n'$, or the length of C and C' are equal to each other but the weight of C' is smaller than the weight of C. The length and weight of C and C' are equal, respectively, to the length and weight of $\hat{C}$ and $\hat{C}'$. Thus, we derive that $\hat{C}$ cannot be the minimum-length negative cycle in G_f^4, as the cycle $\hat{C}'$ is a negative cycle that either is shorter (in length) than $\hat{C}$, or the length of $\hat{C}$ and $\hat{C}'$ are equal to each other, but the weight of $\hat{C}'$ is smaller than the weight of $\hat{C}$ – a contradiction. □

j.4 Computing the Edge Weights in the Bipartite-like Graph

Given a flow f_L corresponding to a placement L, the weights of the bipartite-like graph edges $w_f(u \overset{\min}{\to} v)$ can be computed given the marginal-differential functions Δg and the placement quantities L_i^j, as we show in the next lemma:

Lemma j.6. *Let f_L be a flow corresponding to a placement L that SCC calculates in some iteration. The weights of the bipartite-like graph edges in G_f^B can be computed as follows:*

(a) $w_f(x \overset{\min}{\to} y) = 0$ *if* $|L| < s$ *and otherwise* ∞.

(b) $w_f(y \overset{\min}{\to} x) = 0$ *if* $|L| > 0$ *and otherwise* ∞.

(c) $w_f(x \overset{\min}{\to} a^j) = -\Delta g^j(L^j + 1)$ *if* $L^j < s^j$ *and otherwise* ∞.

(d) $w_f(a^j \overset{\min}{\to} x) = \Delta g^j(L^j)$ *if* $L^j > 0$ *and otherwise* ∞.

(e) $w_f(a^j \overset{\min}{\to} t^i) = -\Delta g_i^j(L_i^j + 1)$ *if* $L_i^j < s$ *and otherwise* ∞.

(f) $w_f(t^j \overset{\min}{\to} a^i) = \Delta g_i^j(L_i^j)$ *if* $L_i^j > 0$ *and otherwise* ∞.

(g) $w_f(t^i \overset{\min}{\to} y) = -\Delta g_i(L_i + 1)$ *if* $L_i < s$ *and otherwise* ∞.

(h) $w_f(y \overset{\min}{\to} t^i) = \Delta g_i(L_i)$ *if* $L_i > 0$ *and otherwise* ∞.

The correctness of the lemma is based on the monotonicity of the marginal-differential functions Δg, and is similar to Lemmas 6.10, e.3.

Proof of Lemma j.6. We will show parts (c) and (d) of the lemma. Other parts can be similarly proven.

To show part (c), let a^j and t^i respectively denote area and resource type nodes, and let e be an edge of minimum weight between a^j and t^i. If e has a cost of $-\Delta g_i^j(r)$ for $1 \leq r \leq L_i^j$, then the flow f sends flow through e. Thus, the residual capacity of e is $c_f(e) = c(e) - f(e) = 1 - 1 = 0$, and therefore its weight in G_f is equal to infinity ($w_f(e) = \infty$).

If e has a cost of $-\Delta g_i^j(r)$ for $L_i^j + 1 \leq r \leq s$ then f does not send flow through e. Thus the residual capacity of e is $c_f(e) = c(e) - f(e) = 1 - 0 = 1$, and therefore its weight in the residual graph G_f is equal to the edge weight in G^4, i.e., $w_f(e) = w(e)$. Thus, $w_f(e)$ can be either equal to $-\Delta g_i^j(L_i^j + 1), -\Delta g_i^j(L_i^j + 2), \ldots$. Among these edges, the one with the minimum cost must be $-\Delta g_i^j(L_i^j + 1)$ (as Δg_i^j is monotonically decreasing and $-\Delta g_i^j$ is monotonically increasing). Thus, the weight of the minimal area-to-type path $w_{f_L}(a^j \overset{\min}{\to} t^i)$ is $-\Delta g_i^j(L_i^j + 1)$. In the special case where $L_i^j = s$, all edges between a^j and t_i have zero residual capacity and infinite weight in the residual graph G_f. Thus, in such case, $w_{f_L}(a^j \overset{\min}{\to} t^i) = \infty$.

To show part (d), let a^j and t^i respectively denote area and resource type nodes. All edges between t^i and a^j in the residual graph G_f^8 are reverse edges. If f does not send flow through the edge e between a^j and t_i, then its reverse edge $\hat{e}$ in the residual graph has zero residual capacity, and thus it has an infinite weight in G_f. If f sends flow through e, then its reverse edge $\hat{e}$ has non-zero residual capacity, and the weight of the reverse edge in the residual graph is equal to (-1) multiplied by the weight of the edge e in the four-layer graph G^4.

The flow f sends flow through edges with costs $-\Delta g_i^j(1), -\Delta g_i^j(2), \ldots, -\Delta g_i^j(L_i^j)$. The cost of their respective reverse edges in the residual graph is equal to $\Delta g_i^j(1), \Delta g_i^j(2), \ldots, \Delta g_i^j(L_i^j)$. Among these edges, the edge with the minimum weight is equal to $\Delta g_i^j(L_i^j)$ (as Δg_i^j is monotonically decreasing). Thus, the weight of the minimal type-to-area path $w_{f_L}(a^j \overset{\min}{\to} t^i)$ equals $\Delta g_i^j(L_i^j)$. In the special case where $L_i^j = 0$, all reverse edges between a^j and t_i

have zero residual capacity and infinite weight in the residual graph G_f. Thus, in such case, $w_{f_L}(a^j \overset{\min}{\to} t^i) = \infty$. □

j.5 SCC Update of the Placement Quantities

We can characterize how SCC updates the placement quantities L_i^j, similarly to the one described in Lemma 6.11, as follows:

Lemma j.7. *Let f be a flow corresponding to L that SCC computes in some iteration. Suppose that SCC augments the flow by the cycle $C = (e_{min}(v_0, v_1), e_{min}(v_1, v_2), \ldots, e_{min}(v_{n-1}, v_n))$ in G_f^4. Then SCC updates the flow f to another flow f' that is corresponding to a placement L', where L' is defined as follows: 1) For every edge (a^j, t^i) in C, SCC updates increases L_i^j by one. 2) For every reverse edge (t^i, a^j) in C, SCC updates decreases L_i^j by one.*

The proof is similar to that of Lemmas 6.11, e.4.

Proof of Lemma j.7. Since the capacity of each edge between a^j and t^i is equal to 1, then we augment by 1 unit of flow in each iteration of SCC. The edges (a^j, t^i) are original edges of G^4, and therefore augmenting through C increases the flow of node (a^j, t^i) by 1, i.e., $f'^j_i \leftarrow f_i^j + 1$. In contrast, the edges (t^i, a^j) are reverse edges of G^4, and therefore augmenting through C decreases the flow of node (a^j, t^i) by 1, i.e., $f'^j_i \leftarrow f_i^j - 1$. As we removed only the edge with minimum weight e_{min} between two vertices, the new updated flow f' is corresponding to L', i.e, $L'^j_i = f'^j_i$. Therefore, the lemma is followed. □

Appendix k

The SCO algorithm - the Full Details

k.1 Full Version of Lemma 10.11, and its Proof

Lemma k.1 (Extended version of Lemma 10.11). *Let o be an extended chain operation $O(j_1 \to j_2 \to \ldots \to j_n)$. Suppose that between j_k to j_{k+1} it moves a resource of type i_k ($k = 1, 2 \ldots n-1$). Then the profitability of o over L can be one of the following formats, according to the class of the operation o:*

1. ***Class 1:*** *If o neither adds a resource to j_1 nor removes one from j_n then*

$$\Delta(o, L) = \Delta^+(j_1) + \sum_{k=1}^{n-1} \Delta_{i_k}(j_k \to j_{k+1}) + \Delta^-(j_n)$$

2. ***Class 2:*** *If o adds a resource of type i_0 to j_1 but does not remove from j_n then*

$$\Delta(o, L) = \Delta^+_{i_0}(j_1) + \sum_{k=1}^{n-1} \Delta_{i_k}(j_k \to j_{k+1}) + \Delta^-(j_n).$$

3. ***Class 3:*** *If o does not add a resource to j_1 but removes a resource of type i_n from j_n then*

$$\Delta(o, L) = \Delta^+(j_1) + \sum_{k=1}^{n-1} \Delta_{i_k}(j_k \to j_{k+1}) + \Delta^-_{i_n}(j_n).$$

4. ***Class 4:*** *If o adds a resource of type i_0 to region j_1 and removes a resource of type $i_n \neq i_0$ from j_n then*

$$\Delta(o, L) = \Delta^+_{i_0}(j_1) + \sum_{k=1}^{n-1} \Delta_{i_k}(j_k \to j_{k+1}) + \Delta^-_{i_n}(j_n).$$

5. ***Class 5:*** *If o adds a resource of type i_0 to region j_1 and removes a resource of type $i_n = i_0$ from j_n then*

$$\Delta(o, L) = \sum_{k=1}^{n-1} \Delta_{i_k}(j_k \to j_{k+1}) + \Delta_{i_0}(j_n \to j_1).$$

where,

$$\Delta_i(j_1 \to j_2) = \Delta g_i^{j_2}(L_i^{j_2}) - \Delta g_i^{j_1}(L_i^{j_1} - 1) \tag{k.1}$$

$$\Delta^+(j) = -\Delta g^j(L^j - 1) \tag{k.2}$$

$$\Delta^+_i(j) = \Delta g_i^j(L_i^j) + \Delta g_i(L_i) \tag{k.3}$$

$$\Delta^-(j) = \Delta g^j(L^j) \tag{k.4}$$

$$\Delta^-_i(j) = -\Delta g_i^j(L_i^j - 1) - \Delta g_i(L_i - 1). \tag{k.5}$$

Proof. Let $\hat{L} = o(L)$. Then, by the definition of profitability and by Eq. (10.7), the profitability of o over L is equal to the sum of $\underbrace{g_i^j(\hat{L}_i^j) - g_i^j(L_i^j)}_{\text{area \& type}}$, $\underbrace{g_i(\hat{L}_i) - g_i(L_i)}_{type}$ and $\underbrace{g^j(\hat{L}^j) - g^j(L^j)}_{area}$ over all regions j and resource types i.

The proof is similar to the one pointed out in the original lemma, (Lemma 10.11). If the operation adds a type i resource in region j, i.e., $\hat{L}_i^j = L_i^j + 1$, then it contributes to the area & type functions $g_i^j(\hat{L}_i^j) - g_i^j(L_i^j) = g_i^j(L_i^j + 1) - g_i^j(L_i^j) \underbrace{=}_{\text{Definition of } \Delta g_i^j} \Delta g_i^j(L_i^j)$. If the operation removes a type i resource from region j, i.e., $\hat{L}_i^j = L_i^j - 1$, then it contributes to the area & type functions $g_i^j(\hat{L}_i^j) - g_i^j(L_i^j) = g_i^j(L_i^j - 1) - g_i^j(L_i^j) \underbrace{=}_{\text{Definition of } \Delta g_i^j} -\Delta g_i^j(L_i^j - 1)$. If the operation does not change the number of type i resources in region j – then, of course, $g_i^j(\hat{L}_i^j) - g_i^j(L_i^j) = 0$ and it does not contribute to the profitability.

In a similar way, if the operation adds a resource to or removes a resource from region j, it contributes, respectively, to the area functions $\Delta g^j(L^j), -\Delta g^j(L^j - 1)$, and if the operation adds or removes a resource of type i, it contributes, respectively, to the area functions $\Delta g^j(L^j), -\Delta g^j(L^j - 1)$.

When the operation o moves a resource of type i from region j_1 to region j_2, then operation o removes a type i resource from region j_1 and adds a resource to region j_2. Thus, each move operation contributes to the area & type functions $\Delta g_i^{j_2}(L_i^{j_2}) - \Delta g_i^{j_1}(L_i^{j_1} - 1) = \Delta_i(j_1 \to j_2)$. Now, the operation of every class moves a type i_k resource from region j_k to region j_{k+1}, and thus contributes to the area & type functions $\sum_{k=1}^{n} \Delta_{i_k}(j_k \to j_{k+1})$.

Note that if o is a fifth-class operation, it adds a resource of type i_0 to region j_1 and removes a resource of type i_0 from j_n. This is equivalent to moving a type i_0 resource from region j_n to region j_1. Thus, in this case, the operation contributes $\Delta_{i_0}(j_n \to j_1)$.

If o is either a first- or a third-class operation, then it does not add a resource to j_1 and thus the number of resources in j_1 is reduced, i.e., the operation o contributes to the profitability $\Delta^+(j_1) = -\Delta g^{j_1}(L^{j_1} - 1)$. If o is either a second-, a fourth-, or a fifth-class operation, the number of resources in region j_1 does not change. The operations in these classes increase the number of type i_0 resources in region j_1, i.e., they contribute to the area & type profitability function $\Delta g_{i_0}^{j_1}(L_{i_0}^{j_1})$ (note that in fifth-class operations this term appears as part of $\Delta_{i_0}(j_n \to j_1) = \Delta g_{i_0}^{j_1}(L_{i_0}^{j_1}) - \Delta g_{i_0}^{j_n}(L_{i_0}^{j_n} - 1)$). If o is either a second- or a fourth-class operation, then the number of type i_0 resources increases by one, and therefore contributes to the profitability the sum $\Delta g_{i_1}(L_{i_1})$. For operations of the fifth class, the number of type $i_0 = i_n$ resources remains the same and does not contribute to the profitability.

In the first- and the second-class operations, o does not remove a resource from j_n and thus the number of resources in j_1 is increased, i.e., the operation o contributes to the profitability $\Delta^-(j_n) = \Delta g^{j_1}(L^{j_n})$. In operations of other classes, the number of resources in region j_n does not change. These operations decrease the number of type i_n resources in region j_n, i.e., they contribute to the area & type profitability function $-\Delta g_{i_n}^{j_n}(L_{i_n}^{j_n} - 1)$. Note that in fifth-class operations, this term appears as part of $\Delta_{i_0}(j_n \to j_1) = \Delta g_{i_0}^{j_1}(L_{i_0}^{j_1}) - \Delta g_{i_0}^{j_n}(L_{i_0}^{j_n} - 1)$, where $i_n = i_0$). In the third- and fourth-class operations, the number of type i_n resources is reduced by one (for the fifth-class operation the number of type $i_0 = i_n$ resources remains the same), and therefore contributes to the sum $-\Delta g_{i_n}(L_{i_n} - 1)$.

Finally, the operation o does not change the number of type i resources in region j, where $(i, j) \neq (i_k, j_k), (i_k, j_{k-1})$. Thus, the region&type marginal profit associated with the number of type i resources in region j is not changed. In addition, the number of type $i \neq i_0, i_n$ resources and the number of resources in region $j \neq j_1, j_2$ do not change, and so does not the marginal profit associated with those numbers. □

k.2 The Full Algorithm to find the Shortest Profitable Operation

The algorithm finds the shortest profitable operation for every class of extended chain operation, as seen in Lemma 10.11. Then, the algorithm will compare the different shortest profitable operations across the different classes and will choose the shortest profitable operation among them. The implementation of the algorithm in each class is similar, with slight differences. We will define algorithm A_i as the algorithm that finds the shortest profitable operation in class i.

All these algorithms are two-stage algorithms: in the first stage they create a graph G, and in the second stage they run a variation of the shortest path algorithm to find the shortest profitable operation.

Given the placement L, in the first stage of these algorithms a complete digraph of region vertices $1, 2 \ldots, k$ is constructed. Our goal is to create a graph G that represents extended chain operation $E(j_1 \to j_2 \to \ldots \to j_n)$ such that between regions j_k and j_{k+1} the operation moves the best resource of type $i(j_k \to j_{k+1})$ between them. More formally, all algorithms define over the edges (j_1, j_2) weights $w(j_1 \to j_2) = (-1) \cdot \max_i \Delta_i(j_1 \to j_2)$, i.e., the best move between region j_1 and j_2 and save the resource type that maximizes this value, i.e., $i(j_1 \to j_2) = \arg\max_i \Delta_i(j_1 \to j_2)$ (i.e., the resource type maximizes Eq. (10.9)).

In addition, algorithms A_1, A_2, A_3, A_4 add two nodes: a source node x and a sink node y, and connect, for every region j, the edges (x, j) and (j, y). The weight of edges (x, j) in A_1 and A_3 (i.e., for the first- and third-class operations) are $w^+(j) = (-1) \cdot \Delta^+(j)$. Algorithms A_2 and A_4 find the best resource of type i_0 to add to region j_1, and therefore they set $w^+(j) = (-1) \cdot \max_i \Delta_i^+(j)$ and save $i_0(j) = \arg\max \Delta_i^+(j)$. The weight of edges (j, y) in A_1 and A_2 are $, w^-(j) = (-1) \cdot \Delta^-(j)$. Algorithms A_3 and A_4 find the best resource of type i_n to remove from region j_n, and therefore they set $w^-(j) = (-1) \cdot \max_i \Delta_i^-(j)$, and save $i_n(j) = \arg\max \Delta_i^-(j)$.

In the second stage, these algorithms use the Bellman-Ford algorithm [72], which, in the i^{th} iteration, computes the shortest path between two vertices s and t of i edges at most. For algorithms $A_1, A_2 \ldots A_n$ it finds the shortest path between $s = x$ and $t = y$. Algorithm A_5 runs simultaneously over all region nodes j and, in the i^{th} iteration, finds the shortest path between $s = j$ to itself ($t = j$) that uses i edges at most. These algorithms stop in the first iteration i_{min} where the shortest path with i_{min} edges has negative weight. When an algorithm finds that the shortest path is $(x, j_1, j_2, \ldots, j_{i_{min}-2}, y)$ (or $(j_1, j_2, \ldots, j_{i_{min}}, j_1)$ in the case of A_5), then it returns the

chain operation $O(j_1 \to j_2, \ldots \to j_{i_{min}-2})$, which moves from region j_k to j_{k+1} a resource of type $i(j_k \to j_{k+1})$. According to the algorithm class, the operation can add a resource of type $i_0(j_1)$ (Classes $2, 4$) to region j_1 and/or remove resource of type $i_n(j_n)$ (Classes $3, 4$) from region j_n.

According to the next claim, these algorithms must find the shortest profitable operation in their respected classes:

Claim k.2. *Algorithm A_i find a shortest profitable operation over all extended chain operations of Class i ($i = 1, 2, 3, 4, 5$).*

Proof. The proof is identical to the proof in Claim 10.12.

□

The complexity of $A_1 \ldots, A_4$ in the first stage is $O(k^2 \cdot m)$ where k is the number of regions and m is the number of resource types. This is because computing $w(j_1 \to j_2)$ takes $O(m)$ steps. Using Bellman-Ford on a graph $G = (V, E)$ takes $O(VE)$. Since our graph contains $O(k)$ vertices and $O(k^2)$ edges – then the running time of the second stage is $O(k^3)$. Thus the complexity of $A_1, A_2, \ldots A_4$ is $O(k^2(k + m))$ steps. Algorithm A_5 in the second stage runs the Bellman-Ford algorithm simultaneously for every region node j i.e., its time complexity in the second stage is $O(k^4)$ and in both stages is $O(k^2(k^2+m))$, which is the time complexity of our full algorithm.

k.3 Description of Residual Graphs and the Augmenting Cycle Theorem for Multigraphs

In Theorem 10.10, we use properties over the well-known **residual graph**. The residual graph is defined next:

Definition k.1. Let $G = (V, E)$ be a (multi)-graph, where a weight function w and an integer capacity function c are defined over the graph edges E, and let f be a flow. We denote the **reverse edges** in G by $\hat{E} = \{(u, v)|(v, u) \in E\}$. The **residual (multi)-graph** $G_f = (V, E_f)$ is a graph constructed similarly to the residual graph in the max-flow problem. On the residual graph edges one defines weight w_f and capacity c_f, and is constructed from a graph G and from flow f by the following steps: 1) Add edges from G to G_f, such that edge $e \in E$ will have a weight of $w_f(e) = w(e)$ and a capacity of $c_f(e) = c(e) - f(e)$. 2) Add the reverse edges of G, $\hat{E}$. That means, if $(v, v') \in E$, then add edge $(v', v) \in \hat{E}$ to G_f with a weight of $w_f(v', v) = -w(v, v')$ and a capacity of

$c_f(v', v) = f(v, v')$. Note that for every edge e in G_f, we have $c(e) \geq 0$. 3) Every edge e in G_f with zero residual capacity $c_f(e) = 0$ is removed from the graph[1].

Most of the min-cost flow algorithms are based on augmenting flow through a path or a negative cycle in a residual graph G_f. When a min-cost flow algorithm augment Δ units of flow through a subset of edges E' from G_f, it affects the flow f in the following way:

- If $e \in E'$ is an edge in the original graph G, then we set $f(e) \leftarrow f(e) + \Delta$.
- If $e = (u, v) \in E'$ is a reverse edge in G, then we set $f(e) \leftarrow f(e) - \Delta$.

According to the new updated flow, one can update the graph G_f with a new weight function w_f and new capacity c_f. We define the weight w_f of cycle C in the residual graph G_f as the sum of the cycle edge weights, i.e., $w_f(C) = \sum_{e \in C} w_f(e)$.

The usefulness of augmenting flow through cycles is described in the Augmenting Cycle Theorem, a well-known theorem, which is presented below.

Theorem k.3 (Augmenting Cycle Theorem for graphs). *Let f and f' be two flows defined over a graph G with edge weights w and integer capacity function c. Suppose that f and f' have the same flow value $|f| = |f'|$. Then, in the residual graph G_f there exist cycles $C_1, C_2, \ldots C_m$ such that 1) f' is equal to f, plus augmenting a positive integer Δ_i of flow units through all C_i, $i = 1, 2, \ldots, m$. 2) The weight of f' is equal to the weight of f plus the weight of the cycles C_i in the residual graph G_f i.e.,*

$$W(f', w) = W(f, w) + \sum_{i=1}^{m} w_f(C_i)\Delta_i \tag{k.6}$$

3) The union of all these cycles $\bigcup_{i=1}^{m} C_i$ is unique.

Proof of the Augmenting Cycle Theorem for graphs. A proof can be found in many places, such as [70][2]. □

Therefore, many min-cost flow algorithms are based on augmenting flow iteratively through negative cycles in the residual graph G_f, until a min-cost flow is reached.

Note that the theorem was proven assuming G is a graph but not a multigraph. Thus, it cannot be applied over the four-layer multigraph G^4. However, we extend that theorem and show that the theorem applies to a multigraph as well.

[1] Alternatively, one can set the weight of these edges to infinity

[2] Note that [70] uses a slightly different version than ours.

Theorem k.4 (Augmenting Cycle Theorem for multigraphs). *Let f and f' be two flows defined over a multigraph G with edge weights w and capacity function c. Suppose that f and f' have the same flow value $|f| = |f'|$. Then, in the residual graph G_f there exists cycles $C_1, C_2, \ldots C_m$ such that 1) f' is equal to f plus augmenting a positive integer Δ_i of flow units through all C_i, $i = 1, 2, \ldots, m$. 2) The weight of f' is equal to the weight of f plus the weight of the flowing cycles C_i in the residual graph G_f, i.e.,*

$$W(f', w) = W(f, w) + \sum_{i=1}^{m} w_f(C_i)c_f(C_i). \tag{k.7}$$

3) The union of all these cycles $\bigcup_{i=1}^{m} C_i$ is unique.

Proof of the Augmenting Cycle Theorem for multigraphs. Given a multigraph G we expand G into $\hat{G}$, such that for every edge e in G that connects vertex v_1 to v_2, we add a new vertex v_e. Then, we connect v_1 to v_e and v_e to v_2. We set the weight of the edge (v_1, v_e) to be equal to the weight of the edge e in G. In addition, the weight of the edge (v_e, v_2) is set to zero. The capacity of both edges $(v_1, v_e), (v_e, v_1)$ in $\hat{G}$ is equal to the capacity of the edge e.

Every flow in the multigraph G has a corresponding flow in G, and vice versa. That means, if a flow f in G sends 1 unit of flow through edge e, then the appropriate flow $\hat{f}$ in $\hat{G}$ sends 1 unit of flow through edges (v_1, v_e) and (v_e, v_2) in $\hat{G}$. The weights of both flows are equal to each other (i.e., $W(f, w) = W(\hat{f}, \hat{w})$). Similarly, for every cycle C in G there is a cycle $\hat{C}$ in $\hat{G}$ and vice versa, with the same weight and residual capacity. Now let f and f' be two flows, with corresponding flows $\hat{f}$ and $\hat{f}'$ in G'. Then, according to the Augmenting Cycle Theorem for graphs, there exist cycles $\hat{C}_i$ in $\hat{G}_{\hat{f}}$ corresponding to cycles C_i in G, where the above three conditions are satisfied with respect to flow $\hat{f}'$ and $\hat{f}$. One can check that the cycles C_i satisfy all three conditions for the flows f and f'. □

k.4 Proof of Lemma 10.13 from Section 10.3

Lemma 10.13. *For every operation o and placement L there exists a sub-operation of o, to be denoted by $o^*(o, L) \subseteq o$ that possesses the following two properties: 1) There is a lowest-weight cycle C such that augmenting flow through C corresponds to $o^*(o, L)$, i.e., it changes the flow f_L to $f_{o^*(o,L)(L)}$, and 2) operation $o^*(o, L)$ is an extended chain operation or a composition of two operation-disjoint extended chain operations.*

Proof of Lemma 10.13. To prove Lemma 10.13, we use Observation k.5 (given below), which characterizes the structure of the residual operation multigraph $G(o, L)$. The observation will imply that the definition of operation $o^*(o, L)$ is well-defined. We then show that the operation $o^*(o, L)$ has a lowest-weight cycle representation (Claim k.6), and then prove that $o^*(o, L)$ is an extended chain operation (Claim k.7).

To define $o^*(o, L)$ we use the following observation that characterizes the edges of the residual operation multigraph $G(o, L)$.

Observation k.5. *The residual operation multigraph $G(o, L)$ should contain only the following edges from the four-layer residual multigraph $G^4_{f_L}$:*

- *For every region node j, if $L^j < o(L)^j$ then $G(o, L)$ contains original edges $(x, j)^{L^j+1}$, $(x, j)^{L^j+2}, \ldots, (x, j)^{o(L)^j}$ of the residual graph $G^4_{f_L}$ i.e., with respective weights $-\Delta\zeta^j(L^j), -\Delta\zeta^j(L^j+1), \ldots, -\Delta\zeta^j(o(L)^j-1)$. If $L^j > o(L)^j$ then $G(o, L)$ contains reverse edges $(j, x)^{L^j}, (j, x)^{L^j-1}, \ldots, (j, x)^{o(L)^j+1}$ of $G^4_{f_L}$ with respective weights $\Delta\zeta^j(L^j - 1), \Delta\zeta^j(L^j - 2), \ldots, \Delta\zeta^j(o(L)^j)$. If $L^j = o(L)^j$ then $G(o, L)$ does not contain any original edge or reverse edge between x and j.*

- *For every region node j and resource type node i, if $L^j_i < o(L)^j_i$ then $G(o, L)$ contains original edges $(j, i)^{L^j_i+1}, (j, i)^{L^j_i+2}, \ldots, (j, i)^{o(L)^j_i}$ of the residual graph $G^4_{f_L}$ i.e., with respective weights $-\Delta\zeta^j_i(L^j_i, D), -\Delta\zeta^j_i(L^j_i+1, D), \ldots, -\Delta^j_i(o(L)^j_i-1, D)$. If $L^j_i > o(L)^j_i$ then $G(o, L)$ contains reverse edges $(i, j)^{L^j_i}, (i, j)^{L^j_i-1}, \ldots, (i, j)^{o(L)^j_i+1}$ of $G^4_{f_L}$ with respective weights $\Delta\zeta^j_i(L^j_i - 1, D), \Delta\zeta^j_i(L^j_i - 2, D), \ldots, \Delta\zeta^j_i(o(L)^j_i, D)$. If $L^j_i = o(L)^j_i$ then $G(o, L)$ does not contain any original edge or reverse edge between i and j.*

- *For every resource type node i, if $L_i < o(L)_i$ then $G(o, L)$ contains original edges $(i, y)^{L_i+1}, (i, y)^{L_i+2}, \ldots, (i, y)^{o(L)_i}$ of the residual graph $G^4_{f_L}$. i.e., with respective weights $-\Delta\zeta_i(L^j_i, D), -\Delta\zeta_i(L_i + 1, D), \ldots, -\Delta_i(o(L)_i, D)$ If $L_i > o(L)_i$ then $G(o, L)$ contains reverse edges $(y, i)^{L_i}, (y, i)^{L_i-1}, \ldots, (y, i)^{o(L)_i+1}$ of $G^4_{f_L}$ with respective weights $\Delta\zeta_i(L_i - 1, D), \Delta\zeta_i(L^j_i - 2, D), \ldots, \Delta\zeta_i(o(L)_i, D)$. If $L_i = o(L)_i$ then $G(o, L)$ does not contain any original edge or reverse edge between i and y.*

- *We denote $|L|$ as the number of resources in a placement L. If $s - |L| < s - |o(L)|$ (i.e., $|o(L)| \geq |L|$) then $G(o, L)$ contains edges $(x, y)^{s-|L|+1}, (x, y)^{s-|L|+2}$, $\ldots, (x, y)^{s-|o(L)|}$ of the residual graph $G^4_{f_L}$. If $s - |L| > s - |o(L)|$ then $G(o, L)$ contains reverse edges $(y, x)^{L^j_i}, (y, x)^{L^j_i-1}, \ldots, (y, x)^{o(L)^j_i+1}$ of graph $G^4_{f_L}$. All reverse and original edges between x and y have zero weight. If $s - |L| = s - |o(L)|$ then $G(o, L)$ does not contain any edge or reverse edge between x and y.*

Proof of Observation k.5. Suppose that we send 1 unit of flow through the edges of $G(o, L)$. Then, the corresponding flow of L, f_L, was changed to the corresponding flow of $o(L)$, $f_{o(L)}$. We define the edges described in the observation by G'. We will show that by augmenting the flow through G' we change the flow f_L to $f_{o(L)}$. By the definition of the residual operation graph $G(o, L)$, augmenting the flow through $G(o, L)$ changes the flow f_L to $f_{o(L)}$. Since there is a single subgraph that can change the flow f_L to $f_{o(L)}$ (see the Augmenting Cycle Theorem k.4) we get that $G' = G(o, L)$.

Let a be a region node. According to its definition, the corresponding flow of L augments 1 unit of flow through edges $(x, a)^1, (x, a)^2, \ldots, (x, a)^{L^a}$. By the following cases, we show that after augmenting the flow through G', the corresponding flow of L changes so that the flow in edges between source x and region node a are similar to the flow of these edges in the corresponding flow of $o(L)$:

1. If $L^a < o(L)^a$, then augmenting 1 unit of flow through G', i.e., through original edges $(x, a)^{L^a+1}, (x, a)^{L^a+2}, \ldots, (x, a)^{o(L)^a}$, will change the flow f_L so that edges $(x, a)^1$, $(x, a)^2$, …, $(x, a)^{o(L)^a}$ will contain 1 unit of flow, similar to the corresponding flow of $o(L)$.

2. If $L^a > o(L)^a$, then augmenting 1 unit of flow through G', i.e., through reverse edges $(a, x)^{L^a}, (a, x)^{L^a-1}, \ldots, (x, a)^{o(L)^a-1}$, will cancel the flow in edges $(x, a)^{L^a}$, $(x, a)^{L^a-1}, \ldots, (x, j)^{o(L)^a+1}$. This means the flow changes so it contains 1 unit of flow through edges $(x, a)^1, (x, a)^2, \ldots, (x, a)^{o(L)^a}$, similar to the corresponding flow of $o(L)$.

3. If $L^a = o(L)^a$, then augmenting 1 unit of flow through G' does not change the flow of edges between source x and a, and the flow still contains 1 unit of flow through edges $(x, a)^1, (x, a)^2, \ldots, (x, a)^{o(L)^a}$, similar to the corresponding flow of $o(L)$.

We can similarly prove that augmenting the flow through other edges in G' will change the corresponding flow of L to the flow of $o(L)$. □

Now, according to Observation k.5, for every region node j and resource type node i, the residual operation multigraph $G(o, L)$ (and therefore every lowest-weight cycle C) cannot contain both original edges from j to i and reverse edges from j to i. This implies a well-defined definition to the operation $o^*(o, L)$.

Suppose that C is a lowest-weight cycle in the residual operation multigraph $G(o, L)$. We define operation $o^*(o, L)$ such that for every region node j and resource type node i it does the following:

- If C contains original edges from j to i, then $o^*(o, L)$ adds a type i resource to region j.

- If C contains reverse edges from i to j, then $o^*(o,L)$ removes a type i resource from region j.
- If C does not contain any edges between i and j, then $o^*(o,L)$ will not change the number of type i resources from region j.

By the following claim, we will show that there is a lowest-weight cycle C' that represents $o^*(o,L)$:

Claim k.6. *There is a lowest-weight cycle C' (that is a simple cycle with the lowest weight in $G(o,L)$) which represents its corresponding operation $o^*(o,L)$, i.e., augmenting flow through C' changes the placement L to $o^*(o,L)(L)$.*

Proof of Claim k.6. Let C be the lowest-weight cycle that we used to create $o^*(o,L)$. We observe that if C connects v_1 to v_2, then it must use a single edge with the lowest weight to connect between the vertices (it can use only one edge as C is simple). Suppose C contains an original edge connecting vertices j to i, denoted by $j \to i$. According to Observation k.5, the residual operation graph $G(o,L)$ (and thus C) should contains edges $(j,i)^{L_i^j+1}, (j,i)^{L_i^j+2}, \ldots, (j,i)^{o(L)_i^j}$ of edge weights $-\Delta\zeta_i^j(L_i^j, D), -\Delta\zeta_i^j(L_i^j + 1, D), \ldots, -\Delta_i^j(o(L)_i^j - 1, D)$. Since the marginal profit functions are concave (i.e., $\Delta\zeta(n,D) \geq \Delta\zeta(n+1,D)$ for every n) then we can take a lowest-weight cycle C' that contains edge $(j,i)^{L_i^j+1} = (j,i)^{L_i^j+1}$. In a similar way, we can assume that if C' contains either $x \to y$, $x \to j$, $i \to y$, then it can be assumed, respectively, that it uses edges $(x,y)^{s-|L|+1}, (x,j)^{L^j+1}, (i,y)^{L_i+1}$. These are exactly the same edges of the residual operation multigraph of $o^*(o,L)$, $G(o^*(o,L),L)$, according to Observation k.5.

If C contains a reverse edge $i \to j$, then according to Observation k.5, the residual operation graph $G(o,L)$ (and thus C) contains edges $(i,j)^{L_i^j}, (i,j)^{L_i^j-1}, \ldots, (j,i)^{o(L)_i^j+1}$ of edge weights $\Delta\zeta_i^j(L_i^j - 1, D), \Delta\zeta_i^j(L_i^j - 2, D), \ldots, \Delta_i^j(o(L)_i^j, D)$. Since the marginal profit functions are concave, then we can take a lowest-weight cycle C' that contains edge $(i,j)^{L_i^j}$, and we can assume the cycle uses that edge. In a similar way, we show that if C' uses either $y \to x$, $j \to x$, $y \to i$, then it can be assumed, respectively, that it uses edges $(y,x)^{s-|L|}, (j,x)^{L^j}, (y,i)^{L_i}$. These are exactly the edges of the residual operation multigraph of $o^*(o,L)$, $G(o^*(o,L),L)$ according to Observation k.5 for the reverse edges. Thus, we proved that $G(o^*(o,L),L) = C'$, and that augmenting the flow through C' changes the placement L to $o^*(o,L)(L)$.

□

To complete Lemma 10.13 we show the following claim:

Claim k.7. *The operation $o^*(o,L)$ is an extended chain operation, or a composition of two disjoint extended chains.*

Proof of Claim k.7. Let C' be a lowest-weight cycle C' (among the simple cycles in $G(o, L)$) corresponding to $o^*(o, L)$. We prove this claim based on the structure of $G(o, L)$ and based on the fact that C' is a simple cycle.

First, C' should contain at least one region node j. Otherwise, C' must contain only the sink x, the source y and type nodes i. The cycle C' cannot be a cycle of length 2, as it is a subgraph of residual operation graph $G(o, L)$, which cannot contain such a cycle (according to Observation k.5), and if C'' is a cycle of length 3 or more, it must contain a vertex twice and thus it is not a simple cycle – a contradiction. We can prove, similarly, that C' should contain at least one resource node i.

Now, if C' contains an original edge from a region node j to a resource type node i, then the residual operation multigraph $G(o, L)$ contains only original edges connecting j to i. Then, by its definition, $o^*(o, L)$ adds a type i resource to region j. If C' contains a reverse edge from a resource type node i to a region node j, then $o^*(o, L)$ removes a type i resource from region j. Thus, if C' contains a path $j_k \to i_k \to j_{k+1}$ that means the operation moves resource of type i_k from j_{k+1} to j_k.

We will now show that $o^*(o, L)$ can be one of six classes of operations, according to the format of the cycle C'. For these classes we define $j_1, j_2 \ldots, j_n$ to be region nodes and $i_0, i_1, j_2 \ldots, i_n$ the resource type nodes, and $v_1 \to v_2$ is an edge connecting v_1 to v_2.

1. **First-class cycle:** A cycle that only includes the source x but not the sink y must be of the format $j_1 \to i_1 \to j_2 \to i_2, \ldots, \to j_n \to x \to j_1$. Since C contains x but not y, there exists a region j_1 such that $x \to j_1$ is included in the cycle, and there exists j_n such that $j_1 \to x$ is included in the cycle. Then, C should contain a path between j_1 and j_n that does not use x (as C is simple) and does not use y. Therefore, the format of the cycle is $j_1 \to i_1 \to j_2 \to i_2, \ldots, \to j_n \to x \to j_1$.

   If C' is a first-class cycle, then $o^*(o, L)$ is an extended chain operation that moves resource of type i_k from region j_{k+1} to region j_k (i.e., $E(j_n \to j_{n-1} \to \ldots j_1)$), and neither adds a resource to j_n nor removes one from j_1.

2. **Second-class cycle:** A cycle that includes the source x, the sink y and the reverse edge (y, x) is of the format $j_1 \to i_1 \to j_2 \to i_2, \ldots, \to j_n \to i_n \to y \to x \to j_1$. In such case, we define j_1 as the region such that $x \to j_1$ is included in the cycle, and i_n as the resource type node that $i_n \to y$ is in the graph. There must be a path between j_1 and i_n that does not use the sink y or the source x, and thus the format of the cycle is $j_1 \to i_1 \to j_2 \to i_2, \ldots, \to j_n \to i_n \to y \to x \to j_1$.

If C' is a second-class cycle, then $o^*(o, L)$ is an extended chain operation that moves a resource of type i_k from region j_{k+1} to region j_k (i.e., $E(j_n \to j_{n-1} \to \dots j_1)$), adds a resource of type i_n to j_n, and removes a resource from j_1.

3. **Third-class cycle:** A cycle that includes the source x, the sink y, and the original edge (x, y) is of the format $j_1 \to i_1 \to j_2 \to i_2, \dots, \to j_n \to x \to y \to i_0 \to j_1$. This is similar to the proof of second-class cycles.

   If C' is a third-class cycle, then $o^*(o, L)$ is an extended chain operation that moves a resource of type i_k from region j_{k+1} to region j_k (i.e., $E(j_n \to j_{n-1} \to \dots j_1)$), does not add a resource to j_n, but removes a resource of type i_0 from j_1.

4. **Fourth-class cycle:** A cycle that does not include x, but includes the sink y is of the format $j_1 \to i_1 \to j_2 \to i_2, \dots, \to j_n \to i_n \to y \to i_0 \to j_1$. This proof is similar to the proof of first-class cycle.

   If C' is a fourth-class cycle, then $o^*(o, L)$ is an extended chain operation that moves a resource of type i_k from region j_{k+1} to region j_k (i.e., $E(j_n \to j_{n-1} \to \dots j_1)$), adds a resource of type i_n to j_n, and removes a resource of type $i_0 \neq i_n$ from j_1.

5. **Fifth-class cycle:** A cycle that neither includes x nor y is of the format $j_1 \to i_1 \to j_2 \to i_2, \dots, \to j_n \to i_n \to j_1$. The proof can be shown similar to the previous classes.

   If C' is a fifth-class cycle, then $o^*(o, L)$ is an extended chain operation that moves a resource of type i_k from region j_{k+1} to region j_k (i.e., $E(j_n \to j_{n-1} \to \dots j_1)$), adds a resource of type i_n to j_n, and removes a resource of type $i_0 = i_n$ from j_1.

6. **Sixth-class cycle:** A cycle that includes the source x and sink y but does not include the edge (x, y) or its reverse edge (y, x) is composed of two disjoint paths: one path from x to y i.e., $P_1 = x \to j_1 \to i_1 \to j_2 \to i_2, \dots, \to j_n \to i_n \to y$ and the other path from y to x, i.e., $P_2 = y \to i'_0 \to j'_1 \dots, \to j'_{n'} \to x$. The resource type nodes i'_0 and i_n are defined such that $i_n \to y$ and $y \to i'_0$ are included in the cycle, and the region nodes j_1, $j'_{n'}$ are defined such that $j'_n \to x$ and $x \to i_1$ are included in the cycle. There must be a path between j_1 and i_n, and a path between i'_0 and $j_{n'}$ that do not use the sink y or the source x, and thus are of the above format.

   If C' is a sixth-class cycle, then $o^*(o, L)$ is composed of two disjoint extended chain operations: The first extended chain operation moves resource of type i_k from region j_{k+1} to region j_{k-1} (i.e., $E(j_n \to j_{n-1} \to \dots j_1)$), and adds a resource of type i_n to region j_n. The second extended chain operation moves resource of type i'_k from region j'_{k+1} to region j'_k (i.e., $E(j'_n \to j'_{n-1} \to \dots j'_1)$), and removes a

resource of type i'_0 from region j'_0. Note that the operations are disjointed, as the cycle C' is a simple cycle

□

Thus the proof of Lemma 10.13 is completed.

□

הרחבות של המידול [1]

בפרק 11 אנו מציגים הרחבות של מידול, כגון בעיית ההשמה שבו עלות השינוי מחושבת כחלק מפונקציית הרווח שלנו, בעיית ההשמה הפרמטרית ואחרות. אנו מראים שבעיות אלו ניתנות לפתרון על פי האלגוריתמים שהצענו בעבודה זאת. תוצאות אלו פורסמו ב[1].

רשימת מאמרים

בעבודה זאת, אנו מציגים שני מאמרים שפורסמו בכנסי proceeding שנבדקו עם ביקורת עמיתים [12,13], מאמר שפורסם בכתב עת [1] שנבדק אף הוא בביקורת עמיתים, ומאמר אחד שהוגש [18]. בנוסף, עבודה זאת מכילה תוצאות שעדיין לא פורסמו [14,22,23] ושאנו מתכוונים להגיש לכתבי עת עם ביקורת עמיתים.

לנוחיות קריאת העבודה, רשימה של סימונים מתאימים ניתן למצוא בעמוד ix, ורשימה של בעיות השמה ניתן למצוא בעמוד xi.

בעיית ההשמה במערכת עם סוגים מרובים (ConRePP-2) [1]

בפרק 10 אנו דנים בבעיית ההשמה עם הגבלה על מספר השינויים, במערכת עם סוגים מרובים (ConRePP-2). תוצאות אלו פורסמו ב[1]. אנו מוכיחים שבעיה זאת קשה. כלומר, אנו מראים שאם קיים אלגוריתם פולינומיאלי לבעיה זו[9], אז ישנו פתרון אלגוריתם לבעיית השידוך המושלם המדויק (Exact Perfect Matching Problem)[10]. השאלה האם לבעיית השידוך המושלם המדויק יש פתרון פולינומיאלי נחשבת לבעיה פתוחה במדעי המחשב, שעדיין לא פתרו קרוב ל30 שנה ([19-21]). לפיכך, ניתן לשער שלבעיה זו אין פתרון פולינומיאלי.

לפיכך, פיתחנו שלושה אלגוריתמים היוריסטיים הפותרים את הבעיה בזמן פולינומיאלי: הראשון נקרא SCC, והוא מבוססת על אלגוריתם הזרימה שפותר את בעיית ההשמה הסטטית ללא הגבלה על קיבולת המשאבים (SPP-4). האלגוריתם הינו גרסה של האלגוריתם הידוע Cycle-Canceling, שניתן להשתמש בו בשביל פתרון בעיית השmin-cost flow. האלגוריתם השני נקרא Hybrid אשר משתמש בSCC. האלגוריתם מבצע שינויים קטנים כאשר הדרישה משתנה מעט, ושינויים גדולים כשהדרישה משתנה הרבה. לבסוף, אנו מציגים אלגוריתם בשם SCO אשר מוצא את הפתרון האופטימלי במקרים ספציפיים. אנו מראים שניתן לממש את SCO בזמן פולינומיאלי, על ידי שימוש בשיטות בתורת הגרפים ובmin-cost flow.

אנו נעזרים באנליזה נומרית על מנת להעריך את ביצועי האלגוריתמים שלנו, בהינתן סביבת עבודה מעשית, כמו העלויות של השכרת שרתים מאמזון EC2, ועומסי רשת מעשיים. התוצאות שלנו מראות שאלגוריתם Hybrid משיג תוצאות הקרובות להשמה האופטימלית (הרווח הינו לפחות 98.3% מהרווח של הפתרון האופטימלי אשר מחושב ללא מגבלה על כמות השינויים), עם כמות שינויים נמוכה. בפרט, אלגוריתם Hybrid מבצע כמות שינויים נמוכה יותר מאשר כמות השינויים בפתרון האופטימלי (יותר מ65%), ויותר נמוכה מהקצאה שהינה פרופורציונלית לממוצע הדרישה [11].

---

[9] פולינומיאלי בהגבלה על כמות השינויים r, בכמות סוגי המשאבים m, ומספר האזורים k.

[10] בהינתן גרף שהקשתות צבועות באדום או בכחול ופרמטר k, בעיית השידוך המשולם המדויק הוא למצוא שידוך מושלם עם k קשתות אדומות; שידוך מושלם הינו קבוצה של קשתות שמכסות את כל הצמתים בגרף, כך שאין שתי קשתות בשידוך שמשתפות את אותה צומת.

בעיות אלו מאתגרות; לא רק שעליהן להתייחס לדרישה השרירותית שמשתנה בכל זמן, אלא הצלחנו להראות שחלק מבעיות אלו הן קשות[8]. זאת בניגוד לבעיות השמה סטטיות, הנפתרות בזמן פולינומיאלי.

לאחר קבלת הפתרון האופטימלי L_{opt}, על מפעיל המערכת לשנות את המשאבים באזורים השונים. אנו מציגים אלגוריתמים יעילים לבעיות אלו שהינם פולינומיאליים בכמות השינויים שמפעיל המערכת עושה, במספר סוגי המשאבים ובמספר האזורים השונים.

הערה 1.3 שימו לב שמשום שמפעיל המערכת חייב לשנות פיזית את המשאבים, סיבוכיות הזמן של כל אלגוריתם הפותר בעיית השמה עם הגבלה על כמות השינויים (ConRePP) חייבת להיות לפחות פולינומיאלית במספר המשאבים שמפעיל המערכת משנה. בשל כך, הפתרונות שאנו מציעים הינם יעילים.

בעיית ההשמה במערכת עם סוג בודד (ConRePP-1)

בפרק 9 אנו פותרים את בעיית ההשמה עם הגבלה על מספר השינויים, בהינתן מערכת עם סוג בודד. הפרק מבוסס על מאמר שהוגש [18]. אנו מציעים אלגוריתם חמדני יעיל, הנקרא **U&Me**, אשר פותר את בעיית ההשמה. סיבוכיות זמן הריצה של האלגוריתם הינה כמעט לינארית בחסם על כמות השינויים r. האלגוריתם משתמש בגישה חמדנית, כך שתחילה הוא מבצע *פעולות אונריות* (כלומר, או הוספה או הורדה של משאב בודד), ומסיים ב*פעולות הזזה (בינאריות)* (כלומר, הזזה של משאב מאזור אחד לאזור אחר). אלגוריתם U&Me מגדיר סדרה של פעולות אונריות ופעולות הזזה כך שביצוע פעולות אלו מוליך לפתרון האופטימלי. סדרה זאת נקראת **סדרת U&Me.**

על אף פשטות האלגוריתם, הוכחת נכונות האלגוריתם אינה טריוויאלית והיא מורכבת. ההוכחה מתבססת על שלושה חלקים עיקריים: 1) סדרת U&Me לא מכילה פעולות שמבטלות זו את זו (כלומר, U&Me לא מוסיף ולא מוריד משאבים מאותו אזור). 2) לכל שינוי חוקי של משאבים קיימת סדרה של פעולות המתאימה לשינוי זה. סדרה זו מקיימת תכונות שסדרת הU&Me מקיימת גם, ולכן היא נקראת **סדרת U&Me-דמה**. 3) אנו מוכיחים שביצוע הפעולות של סדרת הU&Me הינו רווחי יותר מכל שינוי של המשאבים, בהתבסס על התכונות של סדרות U&Me-דמה.

אנו מבצעים אנליזה נומרית על מנת לחקור את השימושיות של האלגוריתם. אנו מראים שניתן להציג פתרון שהינו כמעט אופטימלי גם אם ישנה הגבלה נוקשה בכמות השינויים; כלומר, אפילו אם הפתרון משנה מעט את כמות המשאבים (60% מכמות המקסימלית של שינויים) אזי הפתרון המתקבל הוא כמעט אופטימלי (הרווחיות תגיע ל90% מהפתרון האופטימלי). על פי תוצאות אלו, מפעילי המערכת יכולים לשנות מעט את ההשמה, ולהגיע לפתרון שהוא כמעט אופטימלי.

[8] בעבודה שלנו, בעיה נחשבת ל"קשה" בעבודה שלנו אם פתרון פולינומיאלי לבעיה גורר פתרון לבעיה ידועה ופתוחה, כמו שאלת P=NP.

ניתוח רגישות של השמות אופטימליות [1]

בפרק 8 אנו דנים בדרישה דינמית המשתנה בצורה סטוכסטית. אסטרטגיה נאיבית לתגובה במקרה של דרישה משתנה היא לחלק את המשאבים בצורה אופטימלית לכל שינוי בדרישה. מדיניות זו עלולה להיות בזבזנית בכמות השינויים, ועלולה להיות בלתי אפשרית למימוש בגלל הגבלות על כמות השרתים שניתן להפעיל או לסגור. יותר מכך, בפרק 8 (שמתבסס על התוצאות שפורסמו ב[1]), אנו חוקרים את הרגישות של השמות אופטימליות בתגובה לשינוי בדרישה. התוצאות מראות שמציאת השמה אופטימלית בכל תקופה נתונה עלולה לגרום לשינוי משמעתי של המשאבים, גם במקרה של שינוי קטן בדרישה. בניגוד לכך, שינוי קטן זה גורם לשינוי קטן ברווח של השמה האופטימלית המתאימה. לכן, פתרון מעשי של בעיית ההשמה הוא להתחשב בעלות השינוי[5] ולכן יש הכרח להגביל את כמות השינויים בהשמה.

האנליזה שמוצגת בפרק 8 מבוססת על מיפוי הבעיה לבעיית זרימה מסוג min-cost flow, שבה השתמשנו על מנת לפתור את בעיית ההשמה הסטטית ללא הגבלת קיבולת המשאבים (SPP-4).

בעיות השמה עם הגבלה על מספר השינויים

כדי לענות על הבעיות שנוצרו כתוצאה משינויים רבים, אנו מגדירים **בעיות השמה עם הגבלה על מספר השינויים (ConRePP)**. בעיות אלו מנסות למצוא את ההשמה האופטימלית ביחס לדרישה החדשה, תחת מספר מוגבל של שינויים בכמות המשאבים (הוספות או הורדות של המשאבים). הצורך להקצאה מחודשת עולה עקב שינויים במערכת כמו שינוי הדרישה, שינויי בתמחור המחירים (עקב תנודות במחירי המשאבים[6]), או ע"י שינוי בטופולוגית הרשת כתוצאה מנפילת מרכז נתונים. בעיות אלו מנוסחות באופן הבא: בהינתן השמה $L(t)$ בתקופה t, מה היא ההשמה האופטימלית ביחס לדרישה החדשה $D(t+1)$, תחת האילוץ שניתן לבצע r פעולות אונריות[7]? הניסוח המדויק של כל בעיה כזאת מוצגת לפניכם:

בעיית השמה עם הגבלה על מספר השינויים (ConRePP)

קלט: חיזוי הסתברותי של הדרישה $D(t+1)$ בתקופה $t+1$, ההשמה $L(t)$ בתקופה t, חסם על מספר השינויים r.

פלט: מצא את ההשמה $L(t+1)$ המגדילה ככל האפשר את הרווח $P(L(t+1), D(t+1))$, תחת הגבלה על מספר השינויים $\sum_{i=1}^{m}\sum_{j=1}^{k} |L_i^j(t) - L_i^j(t+1)| \leq r$, כאשר $L_i^j(t)$ מייצג את כמות המשאבים מסוג i באזור j בתקופה t.

---

[5] במחשוב ענן, עלות השינויי עשויה לכלול את התקורה בהעלאת שרתים חדשים ובהורדת שרתים קיימים.

[6] בענן של אמזון, המחירים של השכרת שרתים מסוג spot instance משתנים בצורה דינמית לפי הדרישה והמלאי של שרתים אלו [2].

[7] פעולה אונרית הינה פעולת הוספה או פעולת הורדה של משאב בודד.

באמצעות שימוש בטכניקות אלו, נקבל אלגוריתם **שמהיר יותר** מהאלגוריתם שהוצע בפתרון של הבעיה הקודמת (SPP-3).

אזור בודד או למערכת עם אזורים הומוגניים אינן חזקות דיין לפתור את בעיית השמה במערכת כללית א-סימטרית.

אנו מראים שניתן לפתור את בעיית ההשמה ע"י המרה של הבעיה לבעיית זרימה מסוג min-cost flow על גרף ענק עם שמונה שכבות. בעיית min-cost flow לרוב נפתרת ע"י אלגוריתמים מתאימים, כמו SSP (Successive Shortest Paths). מכיוון שהגרף ענק, אלגוריתם SSP רץ בזמן ארוך. על מנת לפתור בעיה זאת, אנחנו מציגים אלגוריתם שמייעל את SSP בשם **BG**. האלגוריתם רץ בזמן קצר יותר, ולכן מיועד למערכת רבי משתתפים. ניתן להשתמש בBG גם לקבלת חסמים עליונים עבור היוריסטיות וגם לניתוח רגישות. אנחנו משתמשים באנליזה נומרית בשביל לאמת את התוצאות שלנו, ובפרט להראות שהקצאה פרופורציונלית לפי ממוצע הדרישה בעלת ביצועים פחות טובים מהאלגוריתם האופטימלי.

מרבית ההוכחות של פרק זה נמצאים בנספח b.

בעיית השמה הכללית: מערכת ללא הגבלה על קיבולת המשאבים, עם פונקציה רווח כללית (SPP-4)

בפרק 7 אנו מרחיבים את האנליזה שהוצגה בפרק 6 ע"י העשרת המודל ופונקציית הרווח שלנו באמצעות שימוש בפרמטרים החשובים לאפליקציות של מחשוב ענן. אותם פרמטרים כוללים עלויות השכרה, קיבולת של השרת ותלות בסוגי המשאבים; פונקציית הרווח המוצגת בבעיות הקודמות (SPP-1, SPP-2, SPP-3) אינה מכלילה פרמטרים אלה. לפיכך, הבעיה הנוכחית (SPP-4) הינה קשה יותר לפתרון מאשר הבעיות הקודמות. על מנת לפתור בעיה זאת אנו מציעים שיטות פתרון חדשות, כמו אלגוריתם בשם Successive-Shortest-Path-Negative-Edges , בשילוב עם שיטת פוטנציאל (node potentials). ניתן לראות תוצאות אלו ב[14].

בנוסף, האלגוריתם המוצג לבעיה זאת פותר קושי שהוצג באלגוריתם לפתרון הבעיה הקודמת (SPP-3): סיבוכיות הזמן של האלגוריתם הפותר את SPP-3 תלויה במספר המקסימלי של שרתים שהמערכת יכולה להכיל. באפליקציות ענן מספר זה הוא גבוה; למשל, מרכזי נתונים של מיקרוסופט עשויי להכיל עד עשרות אלפי שרתים [15]. לכן, אנו מציגים ופותרים את בעיית ההשמה ללא אילוץ על קיבולת המשאבים, בהנחה שקיים פתרון אופטימלי כזה. סיבוכיות הזמן של האלגוריתם המתאים תלויה רק בגודל ההשמה, ולא במספר המקסימלי של משאבים שניתן למקם במערכת. בנוסף, הבעיה הינה שימושית בפתרונות של תכנון הקיבולת – הבעיה מאפשרת למצוא את מספר המשאבים בהשמה (ואת מיקומם) בשביל לייעל את הרווח כאשר הרווח השולי של השמת משאב שואף לאפס.

כדי למצוא את הפתרון, אנו מציגים תוצאה תאורטית חדשה ומוכיחים את משפט הקמירות של SSP (**SSP convexity theorem**). תוצאה זו יכולה להיות שימושית בבעיות אופטימיזציה אחרות (כמו [13,16,17]) שבו משתמשים באלגוריתם SSP על מנת לפתור את בעיית הזרימה מסוג min-cost flow.

בכל אזור מכל סוג, ובהנחה שישנו אורקל שיכול לתת את התפלגות המערכת, האלגוריתמים שאנו מציעים רצים בזמן פסבדו-פולינומאלי[4] בגודלו של הקלט הצר.

סיכום של בעיות אלו מוצג בהמשך, וניסוחם המתמטי המדויק מובא בראשית כל פרק, על פי המודל שניתן בפרק 3. בשל כך, הפתרונות שאנו מציעים הינם יעילים.

בעיית השמה במערכת עם אזור בודד (SPP-1) ובמערכת עם אזורים הומוגניים רבים (SPP-2) [12]

בפרקים 4 ו5 אנחנו חוקרים את בעיות ההשמה שפורסמו ב[12]. בבעיות אלו, אנחנו מניחים שתי הנחות עיקריות: 1) המערכת מורכבת מאזורים הומוגניים, כלומר, הדרישה מתפלגת בצורה זהה בין האזורים השונים, ולכל האזורים יש אותה קיבולת של כמות המשאבים. 2) פונקציית הרווח P הינה פונקציית תועלת, ללא עלויות, ופשוטה. הגדרה פורמלית של פונקציית הרווח הפשוטה ניתן למצוא בפסקה 3.4. בפרק 4 אנחנו פותרים את בעיית ההשמה לאזור בודד (SPP-1); מטרת הפרק היא לחשוף את הקורא לניסוח המתמטי של הבעיה, ולעקרונות האנליזה שלנו. בפרק 5 אנחנו פותרים את הבעיה למערכת עם אזורים רבים שהינם הומוגניים (SPP-2). התוצאה המרכזית שלנו שהפתרון האופטימלי במערכות אלו באופן המאופיין כ*max percentile*, כלומר מגדילות ככל הניתן את פונקציית ההצטברות (c.d.f) של הדרישה. התוצאה היא בניגוד לפתרונות שהוצעו בעבר, שהראו שהפתרונות הינן פרופורציונליות לממוצע הבקשות (אנחנו דנים ביעילות של פתרונות אלו בפסקה 5.6). אנו מראים שהפתרון מקיים תכונה של *איזון* שמצמצם את מרחב הפתרונות לאלו של השמות מאוזנות וסימטריות-מלאות.

אנו מציגים אלגוריתמים קומבינטורים לבעיות אלו, ואנו מראים שהאלגוריתמים רצים בזמן קצר ובשל כך הם שימושיים. האופטימליות של הפתרון יכולה לשמש עבור יישום מדיניות אופטימלית בהקצאת משאבים, לשם השוואה עם אלגוריתמים אחרים, או לשם מתן הנחיות תפעול לבנייה מערכות. אנו מספיקים אנליזה נומרית וסימולציות כדי לאמת את התוצאות שלנו ואת התנהגות המערכת. אנחנו משווים את הביצועים של פתרונות אלטרנטיביים עם הפתרון האופטימלי שלנו.

מרבית ההוכחות של פרק 5 מופיעים בנספח a.

בעיית השמה במערכת עם אזורים הטרוגניים (SPP-3) [13]

בפרק 6 אנחנו מרחיבים את בעיות ההשמה של פרקים 4 ו5, כך שייפתרו בעיית השמה של מערכת עם אזורים הטרוגניים (SPP-3) . תוצאות אלו פורסמו ב[13]. טכניקות הפתרון שהוצגו למערכת עם

[4] **אלגוריתם פסבדו-פולינומאלי** הינו אלגוריתם שזמן הריצה שלו פולינומאלי בערכו הנומרי של הקלט, אבל מערכי בגודלו הבינארי של הקלט.

על אף שהבעיות המובאות בתזה זו מנוסחות בתחום של מחשוב ענן, כלליות המידול שלנו מאפשרת לפתור בעיות רבות בתחום השמת משאבים, החל מבעיות מלאי מבוזרות ועד להקצאת כוח אדם במרכזי שירות שונים.

בעיות השמה סטטיות

החלק הראשון בעבודה שלנו מתייחס לבעיות השמה שבו על ספק השירות לייעל את המערכת בתקופת זמן נתונה. בעיות אלו נקראות **בעיות השמה סטטיות** (באנגלית, SPPs), בניגוד לבעיות השמה דינמיות שבו מיקום המשאבים משתנה לאורך זמן. בעיות אלו מנוסחות בצורה דומה: בהינתן הדרישה $D = D(t)$ בתקופה t, מצא את ההשמה L המגדילה ככל האפשר את הרווח $P(L, D)$, ומקם משאבים אלו. הניסוח המתמטי של כל בעיית השמה סטטית מוצגת לפניכם:

בעיית השמה סטטית (SPP)	
קלט:	חיזוי הסתברותי של הדרישה $D = D(t)$ בתקופה t.
פלט:	מצא את ההשמה L המגדילה ככל האפשר את הרווח $P(L, D)$.

בעבודה זו, אנו פותרים מגוון רחב של בעיות השמה סטטיות. בעיות אלו מיועדות למערכות ולאפליקציות עם תכונות שונות, ועם אלגוריתמים מתאימים הרצים בזמן משתנה. למשל, פתרנו בעיית השמה סטטית עבור אזורים הומוגניים, שבו הדרישה היא סימטרית ברחבי האזורים. כמו כן, פתרנו בעיית השמה סטטית עבור אזורים הטרוגניים. הפתרון האופטימלי למערכות הומוגניות הוא מהיר ופשוט יותר מזה של מערכות הטרוגניות; עם זאת, מכיוון שמרבית המערכות במציאות הינן הטרוגניות, הפתרון למערכות הטרוגניות הינו יותר שימושי.

לאחר קבלת הפתרון האופטימלי L_{opt}, על מפעיל המערכת למקם אם המשאבים באזורים השונים. אנו מציגים אלגוריתמים יעילים לבעיות אלו שהינם פולינומיאליים במספר המשאבים שמפעיל המערכת מציב (כלומר, במספר המשאבים בפתרון האופטימלי), במספר סוגי המשאבים ובמספר האזורים השונים.

הערה 1.2 שימו לב שמכיוון שמפעיל המערכת חייב למקם פיזית את המשאבים, סיבוכיות הזמן של כל אלגוריתם הפותר בעיית השמה סטטית (SPP) חייבת להיות לפחות פולינומיאלית במספר המשאבים שמפעיל המערכת מציב. זו הסיבה שהאלגוריתמים המוצעים הינם יעילים. יתר על כן, במקרה ויש צורך לעבד (או לקרוא) את התפלגות הדרישה השרירותית, יהיה צורך להשקיע לפחות $O(s)$ רק בעיבוד האינפורמציה הדרושה, כשs הינו חסם עליון על כמות המשאבים שההשמה יכולה להכיל ($s \geq |L| = \sum_i \sum_j L_i^j$). בהנחה שמטרתו של מפעיל המערכת היא לחשב את כמות המשאבים

פופולאריים, ישנם סרטים שהם בעלי ביקוש נמוך. מחקרים הראו שדפוסי השימוש באפליקציות VoD הינן מגוונות, למשל בשירותים כמו נטפליקס, IPTV ויוטיוב [6,7].

ב. הדרישה הינה בלתי צפויה עם שונות גבוהה. דרישה עם שונות גבוהה יכולה להיווצר, למשל, ע"י גל של משתמשים במערכת, או ע"י מאורע ברשת (למשל, נפילה בלתי צפויה של מרכז נתונים, הגורם לכל הבקשות להיות מסופקות ע"י מרכז נתונים אחר). מחקרים הראו שהדרישה במערכות אלו יכולה להיות תנודתית מאוד ולכן בלתי צפויה [8]. במערכות בהן שונות הדרישה היא נמוכה ניתן להניח שהדרישה הינה (כמעט) דטרמיניסטית, וניתן להשתמש בפתרונות המניחים זאת (כמו [9,10]), בנוסף לפתרונות המוצעים בתזה.

ג. הדרישה הסטוכסטית משתנה מאוד לאורך זמן בין היום לבין הלילה באזורים שונים, למשל, בשל שינויי במספר הלקוחות האפשריים [5]. על מנת להפעיל בצורה יעילה את המערכת, על הספק להגיב לשינויי הדרישה, ומעת לעת למקם מחדש את המשאבים באזורים השונים.

המודל שלנו, המבוסס על התפלגות הדרישה, שונה מפתרונות אחרים המבוססים על ממוצע הדרישה. פתרון פשוט כמו הקצאה פרופורציונלית לממוצע הדרישה אינה יעילה[3] כפי שמודגם בדוגמה הבאה: נניח שבמערכת ישנו אזור בודד, המסוגל להכיל עד n משאבים. ניתן למקם במערכת שני סוגים שונים של משאבים, וכל משאב עונה על בקשה אחת בדיוק. הדרישה למשאבים מהסוג הראשון היא דטרמיניסטית, ומגיעים בדיוק לn בקשות. הדרישה למשאבים מהסוג השני היא סטוכסטית, כך שבהסתברות של $\frac{1}{k}$ מגיעים nk^2 בקשות ובהסתברות של $1-\frac{1}{k}$ מגיעים לאפס בקשות. על ספק המערכת לענות על כמה שיותר בקשות בממוצע. האלגוריתם האופטימלי ימקם n משאבים מהסוג הראשון, וכל הבקשות למשאבים אלו ייענו. הקצאה פרופורציונלית תמקם $\frac{n}{k+1}$ משאבים מהסוג הראשון ו $\frac{nk}{k+1}$ משאבים מהסוג השני. כמות הבקשות שיסופקו במקרה זה שווה ל $\frac{2n}{k+1}$. כאשר k שואף לאינסוף, נקבל שההקצאה הפרופורציונלית עונה על אפס בקשות, ובפרט, יחסי הביצועים בין הגישות השונות שואף לאינסוף. דוגמה יותר שימושית שמסבירה את ההבדלים בין הגישות השונות נדונה בפרק 6.4.

הערה 1.1: ניתן ליעל את ביצועי ההקצאה הפרופורציונלית ע"י קיצוץ בזנב ההתפלגות של הדרישה. כלומר, להניח שהדרישה לא יכולה להיות יותר מאשר כמות המשאבים במערכת. אפילו בהתפלגויות אלו, יתכן שהקצאה פרופורציונלית אינה אופטימלית: ניקח מערכת של אזור בודד עם $\frac{n}{2}$ סוגי משאבים, המסומנים ב $t_1, t_2, .., t_{\frac{n}{2}}$, עם דרישה דטרמיניסטית של בקשה אחת. בנוסף, המערכת מכילה $\frac{n}{2}$ סוגי משאבים, המסומנים ב $t_{\frac{n}{2}+1}, , .., t_n$, שבה בהסתברות של $\frac{1}{\sqrt{n}}$ מגיעות n בקשות, ובהסתברות של $1-\frac{1}{\sqrt{n}}$ מגיעות אפס בקשות. נניח והמערכת יכולה להקצות n משאבים. בהקצאה האופטימלית יוקצה משאב בודד לסוגים $t_1, t_2, .., t_{\frac{n}{2}}$, ויוקצו $\frac{n}{2}$ משאבים מסוגים אחרים. הרווח של ההקצאה האופטימלית שווה ל $\frac{n}{2}+\frac{\sqrt{n}}{2}=\theta\,(n)$. לעומת זאת, הקצאה פרופורציונלית תקצה $\frac{1}{\frac{n}{2}+\frac{n}{2}\sqrt{n}}\cdot n=\frac{2}{1+\sqrt{n}}$ משאבים לסוגים $t_1, t_2, .., t_{\frac{n}{2}}$, ותקצה $\frac{\sqrt{n}}{\frac{n}{2}+\frac{n}{2}\sqrt{n}}\cdot n=\frac{2\sqrt{n}}{1+\sqrt{n}}$ לשאר המשאבים. הרווח של הקצאה פרופורציונלית שווה ל $\frac{2n}{\sqrt{n}+1}=\theta(\sqrt{n})$. כלומר, הקצאה פרופורציונלית $\frac{2}{1+\sqrt{n}}\cdot\frac{n}{2}+\frac{2\sqrt{n}}{1+\sqrt{n}}\cdot\frac{n}{2}\cdot\frac{1}{\sqrt{n}}=$ פחות טובה מההקצאה האופטימלית בפקטור של $\theta(\sqrt{n})$.

[3] ישנם מחקרים, כמו [11], שהראו שהקצאה פרופורציונלית לדרישה היא אופטימלית במקרים מסוימים.

דוגמא נוספת לאפליקציה שמתאימה לפרדיגמה זו היא אפליקציית וידאו על פי דרישה (VoD) המשתמשת בטכנולוגיית עמית-אל-עמית (Peer-to-Peer). מטרת הספק באפליקציה זו היא לספק תכני וידאו ללקוחות הנמצאים באזורים שונים ומצוידים בממירים שונים. לענות על דרישת הלקוחות משרת מרכזי לגרום לצוואר בקבוק, ומגביל את הסילומיות (scalability) של המערכת. ניתן לפתור בעיה זאת ע"י השמת מראש של הסרטים בממירים של הלקוחות (העמיתים), ובאמצעות העלאת הסרטים ללקוחות אחרים. לשרת בקשה לסרט ע"י עמית באותו אזור של הבקשה הינו רווחי יותר משירות הבקשה ע"י עמית באזור אחר. מטרת ספק השירות היא למצוא היכן למקם את הסרטים השונים על מנת להקטין את העלות הממוצעת (או לחילופין, להגדיל את הרווח) של שירות הבקשות. ניתן למצוא אפליקציות אחרות שנופלות תחת המידול שלנו בפסקה 3.3 .

בתזה, אנחנו מתמודדים עם הבעיה הכללית של מיקום המשאבים השונים באזורים מבוזרים שונים תחת דרישה סטוכסטית. למטרה זו, אנחנו מנסחים בעיות ומודלים שונים, ומציעים פתרונות שונים לבעיות ההשמה מתאימות.

המידול שלנו

עבודה זאת כוללת אוסף של מודלים מגוונים המשלבים עלויות שונות ומתאימות למערכות שונות. לנוחיות הצגת העבודה אנחנו מציגים בפרק 3 מסגרת כללית של כל המודלים השונים באמצעותו ניתן לנסח את הבעיות השונות בעבודה זו.

התשתית של המערכות השונות מציוה באזורים שונים, ומכילה סוגים שונים של משאבים. מספר המשאבים בכל אזור יכול להיות מוגבל. ספק השירות מקבל חיזוי הסתברותי של הדרישה הסטוכסטית $D = \{D_i^j\}$, כאשר D_i^j הוא משתנה מקרי המייצג את מספר הבקשות למשאב מסוג i באזור j. אנחנו מניחים שהדרישה D הינה כללית, כלומר, ההתפלגויות של כל משנה מקרי D_i^j היא שרירותית, ואין אנו מניחים שהמשתנים המקרים הינם בלתי-תלויים או מתפלגים בצורה זהה. אנו יודעים כי חיזוי הסתברותי מייצר תוצאות יותר מדויקות מחיזוי דטרמיניסטי. ניתן לחלץ חיזוי הסתברותי ע"י תיעוד של הדרישה, כפי שמתבצע בתעשייה [4].

בהינתן הדרישה, מטרת הספק היא למקם באופן אופטימלי (תחת אילוצים) את המשאבים באזורים השונים כדי למקסם את הרווח. הרווח שווה לתועלת של סיפוק הדרישה פחות העלויות של הפעלת המשאבים. התועלת לוקחת בחשבון מרחקים גאוגרפים, ככה שלענות על בקשה ממשאב הנמצא באותו אזור הינה רווחית יותר מלענות על הבקשה ע"י משאב באזור מרוחק; לא לענות על בקשה תייצר רווח שלילי או אפסי. העלויות הינן פונקציות קמורות שרירותיות, ויכולות לייצג עלויות אזוריות, עלויות אנרגיה, עלויות השכרת משאבים ועלויות אחרות.

השימוש של הכלליות של הדרישה ושל עלויות ההפעלה מאפשרות לנו לטפל במגוון תרחישים שונים:

א. הדרישה ועלויות הפעלה יכולות להשתנות בין אזורים וסוגים של משאבים שונים. למשל, בספקי ענן כמו אמזון EC2 ומיקרוסופט Azure, הדרישה ומחירי ההשכרה שונים מפלטפורמה לפלטפורמה ובין אזורים שונים [2,3,5]. בנוסף, באפליקציות VoD, הדרישה יכולה להשתנות בין סרטים שונים (ובין השרתים שמכילים סרטים אלו). כלומר, בעוד שישנם סרטים

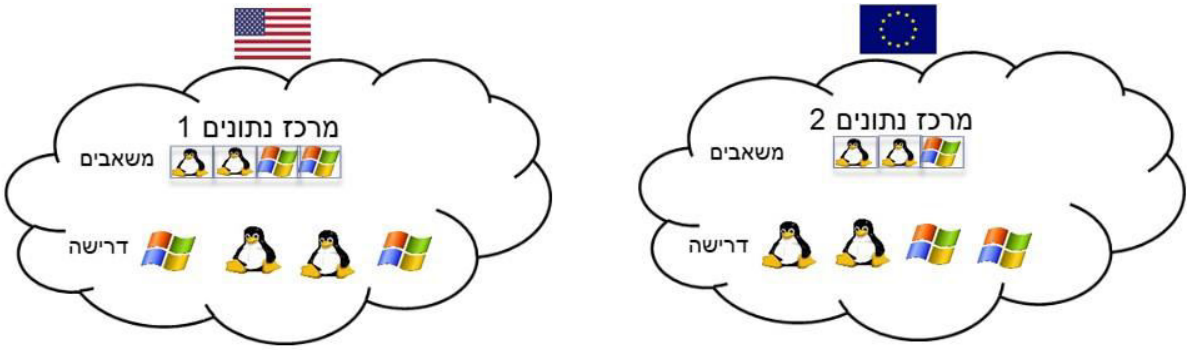
תקציר

בשנים האחרונות קיימת עליה בשימוש של שירותי האינטרנט, הכוללים מידע רב העובר בשירותים אלו. ספק השירות משכפל מידע וממקם משאבים (כמו שרתים) באזורים השונים. כתוצאה מחלוקה זו – זמינות של שירותי המידע ויעילות המערכת עולה. לדוגמא, פלטפורמות פופולריות של מחשוב ענן, כמו אמזון EC2 [2] ומיקרוסופט Azure [3], מחזיקות חוות שרתים וירטואליים[2] במרכזי נתונים מבוזרים, ומשכירים שרתים אלו לכל דורש. בשונה מתכנון מרכזי, ספק השירות מציב את המשאבים קרוב למשתמשים, על מנת להעלות את טיב השירות המאופיין בזמני תגובה נמוכים ועמידות ביחס לתכנון מרכזי.

במערכות אלו, על ספק השירות לאזן שני פרמטרים עיקריים:

א. התועלת ממענה על הדרישה למשאבים. הרווח ממענה על הבקשה ע"י משאב שנמצא באותו אזור הוא יותר גדול מאשר ממענה על הבקשה ע"י משאב הנמצא באזור אחר.
ב. העלויות של הצבת והפעלת המשאבים. למשל, באמזון EC2 התשלום מתבצע בהתאם למאפיינים הפיזיים ולמיקום של השרתים שהושכרו.

בשל כך, בהינתן הדרישה המבוזרת מאזורים שונים, מטרת ספק השירות היא למצוא השמה של המשאבים שתגדיל ככל האפשר את הרווח של המערכת.



תמונה 1: דוגמא בסיסית של מחשוב ענן, שבו משתמשים מבקשים שרתים עם מערכות הפעלה שונות (וינדוס ולינוקס).

[2] שרתים וירטואליים נקראים גם בשם "מכונות וירטואליות" (Virtual Machines) בעגה המקצועית.

תמצית

התזה עוסקת בבעיית השמת המשאבים במערכות גיאוגרפיות מבוזרות ובפרט במחשוב ענן. אנחנו מניחים שהמערכת עם מספר רב של משתמשים, ובכל אזור גיאוגרפי ישנה דרישה למשאבים מסוגים שונים. מטרת מפעיל המערכת היא להציב את המשאבים באזורים אלו על מנת לענות על הדרישה ובכך להגדיל את רווח המערכת. הדרישה מיוצגת ע"י משתנה מקרי רב-ממדי, ועל מפעיל המערכת להתייחס להתפלגות השרירותית המלאה של הדרישה. בעיית השמה השרתים במחשוב ענן ומרכזי נתונים נופלים תחת קטגוריה זאת.

הבעיות שלנו פותחו במסגרת מודל נרחב, הכוללת עלויות שונות ומתאימות למערכות רבות. בעיות אלו מיוצגות בצורות שונות בהתאם לאפליקציות ולמערכות שאותן הן מייצגות. האלגוריתמים שלנו רצים בזמן פולינומיאלי[1] מהיר.

ניתן להשתמש באלגוריתמים ובפתרונות שלנו עבור המטרות הבאות:

1) מציאת פתרון אופטימלי.

2) מציאת חסמים עבור אלגוריתמים היוריסטיים.

3) ניתוח רגישות.

הפתרונות שלנו מבוססים על תוצאות אנליטיות, המשלבות מתודולוגיות של אנליזה סטוכסטית וגם מתורת הגרפים, וניתן להשתמש בשיטות אלו עבור בעיות אופטימיזציה אחרות.

[1] פולינומיאלי במספר המשאבים שמפעיל המערכת מציב או משנה.

אוניברסיטת תל אביב

הפקולטה למדעים מדויקים ע"ש ריימונד ובברלי סאקלר

בית הספר למדעי המחשב ע"ש בלווטניק

השמת משאבים במחשוב ענן ואפליקציות רשת

מאת

יובל רוכמן

חיבור לשם קבלת תואר "דוקטור לפילוסופיה"

עבודת המחקר בוצעה בהדרכתו של

פרופסור חנוך לוי

הוגש לסנאט של אוניברסיטת תל אביב

תשרי תשע"ח