Dynamic placement of resources in Cloud Computing and Network Applications

Yuval Rochman[a,*], Hanoch Levy[a], Eli Brosh[b]

[a]*School of Computer Science, Tel Aviv University, Ramat Aviv, Tel Aviv 69978 Tel Aviv, Israel*

[b]*Nexar, Ltd.*

Abstract

We address the problem of dynamic resource placement in general networking and cloud computing applications. We consider a large-scale system faced by time varying and regionally distributed demands for various resources. The system operator aims at placing the resources across regions to maximize revenues, and thus needs to address the problem of how to dynamically reposition the resources in reaction to the time varying demand.

The challenge posed by this setting is to deal with arbitrary multi-dimensional stochastic demands which vary over time. Under such settings one should provide a tradeoff between optimizing the resource placement as to meet its demand, and minimizing the number of added and removed resources to the placement. Our analysis and simulations reveal that optimizing the resource placement may inflict huge resource repositioning costs, even if the demand has small fluctuations. We therefore propose an algorithmic framework that overcomes this difficulty and yields very efficient dynamic placements with bounded repositioning costs. Our solution is developed under a very wide cost model, and thus allows accommodation of many systems. Our solutions are based on new analytic techniques utilizing graph theory methodologies that can be applied to other optimization/combinatorial problems.

Keywords: Resource-placement, stochastic, distributed-cloud, graph algorithms.

1. Introduction

Cloud computing has emerged as an attractive solution for building large scale services and geographically distributed applications over the Internet. Popular cloud computing platforms like Amazon EC2 [1] and Microsoft Azure [2] organize a shared pool of (virtual) servers [1] in geographically distributed data-centers to enable on-demand delivery of computer resources at scale. By using a distributed cloud platform, the service provider can place server resources at geographical areas close to its users to provide adequate level of service quality, e.g., low response times, and for better resiliency.

To engineer such a system, 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 cost of placing a resource, which reflects the cost of renting a

*Corresponding author

Email addresses: yuvalroc@gmail.com (Yuval Rochman), hanoch@cs.tau.ac.il (Hanoch Levy)

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

server in the cloud. 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 data-center location. The service provider thus deals with regionally distributed demands for various resources by aiming to find a low cost placement of resources at the regions that also provides high service quality.

It has been observed that the demand in such settings can exhibit large variations. It can vary over time, changing significantly (by an order of magnitude) over the course of a day due to, for example, changes in the amount of available users [3]; and can also sustain large spikes due to, for example, unpredicted datacenter failures [4]. To obtain a cost effective operation of the system, the provider needs to dynamically adapt to demand changes, and periodically reposition allocated resources.

In this paper, we address the *general* problem of how to dynamically place resources of multiple types over geographically distributed regions in response to changing stochastic demands. To this end, we develop a general model which incorporates the major cost parameters that affect the design of resource placement systems. The stochastic demand is captured as an arbitrary multi dimensional distribution which *varies over time*.

We analyze the sensitivity of optimal placements to changes in the (stochastic) demand. Our results reveal that changing the optimal placement in reaction to demand deviations can inflict huge repositioning costs, and, that in contrast, these small demand deviations result in minor effect on the profit. Therefore, a practical treatment of the placement problem should account for repositioning costs [2] and possibly should limit the number of resources that can be repositioned.

For this analysis we rely on mapping the problem to a min-cost flow problem (in graph theory), a technique that is usually intended to solve combinatorial problems efficiently. However, we demonstrate that such a mapping, in addition, yields profit sensitivity analysis in placement problems in general.

Addressing the difficulties introduced by large repositions we formulate the *constrained (resource) reposition problem* in reaction to changing demands as one that finds an optimal placement under a constraint of allowing up to a fixed number of additions or removals of resources. This placement problem is challenging; Not only that it needs to deal with arbitrary stochastic demands which can vary over time, but we also establish theoretical strong evidence that under the wide setting of this work the problem at large is hard (that is, it is expected that a polynomial time algorithm cannot solve the problem). We therefore develop a heuristic algorithm called SCO that solves the reposition problem, and is guaranteed 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.

For a multi-period online setting, whereby in every period (opportunity to reposition the resources) only the (stochastic) demand of the current period is known, we propose Hybrid, an algorithm that preforms small resource repositioning when the demand changes are small and conducts SCO when they are large.

We use 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 near optimal placement (its cost is higher up to 1.4% from the optimal placement cost), while maintaining a low reposition cost. In particular, the reposition cost of the proposed Hybrid algorithm is significantly lower than that of an optimal placement (more than 65%) as well as that of a proportional mean-based placement, a simple and widely-used scheme [5] wherein the number of servers is proportional to the average demand.

For the sake of completeness we also examine the *unconstrained reposition problem with reposition costs*; in this problem the number of repositions are not constrained but each of them incurs cost. Our analysis can be extended to provide an exact solution for this problem.

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

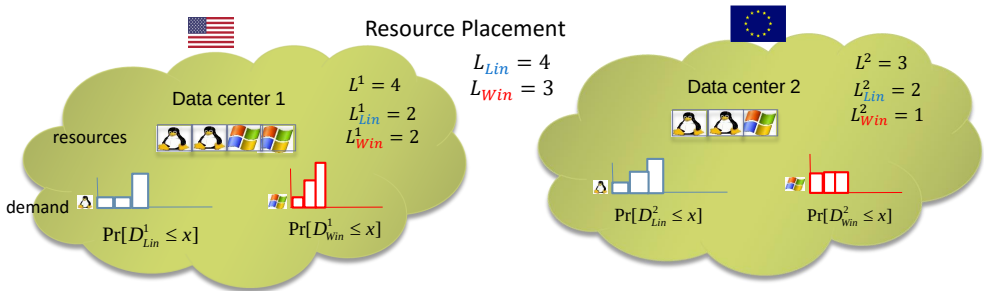

Figure 1: An example of the system. Note that the placement is $L^1_{Lin} = L^2_{Win} = 2$, $L^2_{Lin} = 2, L^2_{Win} = 1$ etc.

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 from geo-distributed inventory problems to personnel allocation in multi contact center settings.

The rest of the paper is organized as follows. Section 2 presents our model and the problem. Section 3 establishes fundamental sensitivity properties of resource repositioning. Section 4 proves the hardness of the constrained reposition problem. Section 5 presents the polynomial time SCO algorithm for solving the problem. Section 6 presents the Hybrid algorithm. Section 7 presents extensions of the model and the solutions, including to the unconstrained reposition problem with reposition costs and including accounting for the costs of unsatisfied requests. Section 8 provides performance evaluation of the system through numerical results. Section 9 describes previous work. Finally, Section 10 concludes this paper.

2. The model and the problem

For the sake of exposition, we start by formulating the model used for the dynamic and static placement problems, and then describe these problems. Although we focus our examples on a SaaS (Software as a Service) service provider that rents servers from a cloud provider, the generality of our model allows solving a variety of resource placement problems spanning from server allocation in server farms to personnel placement in call centers.

2.1. The model

We consider a provider that aims to maximize the profit from servicing requests by placing resources in k **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 arrive stochastically, and can be granted locally by a resource in the same area or granted remotely from a different area. We do not assume any correlation of these arrivals, neither make any assumptions regarding their distributions.

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 two variables in our model: the arbitrary demand $D = (D^j_i)$ and the provider's resource placement $L = (L^j_i)$ for $i = 1, 2, \ldots m$ and $j = 1, 2, \ldots k$. The notations L^j_i, D^i_j denote respectively the number of type i requests made and type i resources placed in area j. An example of the system topology and placement (of Windows and Linux resources) in response to user demand is depicted in Figure 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 areas. 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 rest our analysis on variable transformation that makes the analysis feasible through a semi-separable function. The function form is

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

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 areas, and $L^j = \sum_{i=1}^{n} L_i^j$ is the number of area-j resources of all types. The functions ζ_i, ζ^j and ζ_i^j are called the marginal profit functions, or, for short, **the marginal functions**.

The function ζ_i represents the basic profit composed of the revenue of serving a type i request, regardless of serving it locally or by remote resource, minus the cost of operating or manufacturing the type i-resources. We assume that the profit gained by satisfying a type i request is a constant $R_i > 0$. 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 cost function $C_i : \mathbb{N} \to \mathbb{R}^+$:

$$\zeta_i(L_i, d_i) = R_i \cdot \min(L_i \cdot B_i, d_i) - C_i(L_i). \tag{2}$$

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. (2):

$$\zeta_i^j(L_i^j, d_i^j) = R_i^j \cdot \min(L_i^j \cdot B_i, d_i^j) - C_i^j(L_i^j), \tag{3}$$

where $R_i^j > 0$ is the profit of granting a type i request in region j.

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 in an area and regional taxes. The function form is expressed as follows:

$$\zeta^j(L^j) = -C^j(L^j). \tag{4}$$

As stated above, the combination of Equations (1),(2),(3) and (4) allows modeling 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 when defining a profit function for this setting.

We must capture the fact that the revenue 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. (1)) where the expectation is over a demand distribution D. Substituting this into in Eqs.(2),(3),(4) the profit function in Eq. (1) gets the following form:

$$P(L,D) = \sum_{i=1}^{m}\sum_{j=1}^{k}\zeta_i^j(L_i^j, D_i^j) + \sum_{i=1}^{m}\zeta_i(L_i, D_i) + \sum_{j=1}^{k}\zeta^j(L^j), \tag{5}$$

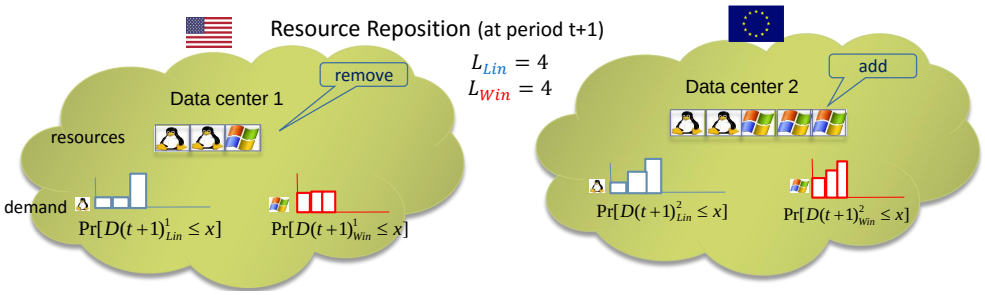

Figure 2: Reposition of resources as response to demand change: Remove Windows resources from region 1 and add in region 2. The initial placement $L(t)$ is given in Figure 1.

where,

$$\zeta_i^j(L_i^j, D_i^j) = R_i^j \cdot E_{D_i^j}[\min(L_i^j \cdot B_i, D_i^j)] - C_i^j(L_i^j), \tag{6}$$

$$\zeta_i(L_i, D_i) = R_i \cdot E_{D_i}[\min(L_i \cdot B_i, D_i)] - C_i(L_i^j), \tag{7}$$

$$\zeta^j(L^j) = -C^j(L^j), \tag{8}$$

are the marginal functions.

Remark 1 (Costs of un-served demands). For simplicity of the model exposition, the model description given above used throughout the paper assumes that demands are not served incur no specific cost. In Appendix A we extend the model to account for costs that are incurred by the system due to the un-served demands.

2.2. Problem formulation

The problem we deal with in this paper is addressed under a dynamic setting challenge. That means, not only that the profit is a complex general function and the demand is arbitrary stochastic, the (stochastic) demand random variable also varies over time. This can model the difference between the stochastic demands during day and night, the changes occurring between busy hours (rush hours) to non-busy hours, change of setting (e.g. due to area failure) and many other changes.

A naive strategy to react to the dynamically changing conditions (demand) is to allocate an optimal placement in response to any change. However, it may be infeasible as for sparing with the efforts for temporal improvement, or due to limitations of operating new or turn off resources. Specifically, the task of resource reposition may require technical, administrative and management activities that do require time and human resources, especially when dealing with large scale geo-distributed systems; further the time duration allowed for the repositions may be limited, and thus the combination of these effects implies that the number of repositions is limited. Since such tasks are normally performed in a central operation site we assume that the restrictions on number of repositions is global. Thus, the optimal placement problem becomes a **Constrained Reposition Problem**: 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 (each operation is either a single *addition* or a single *subtraction* of a resource). An example of resource reposition in response to demand change is given in Figure 2.

This problem can be formulated mathematically as follows:

$$\max_{L(t+1)} E_{D(t+1)}[P(L(t+1), D(t+1))] \tag{9}$$

$$\text{such that } \sum_{i=1}^{m}\sum_{j=1}^{k} |L_i^j(t) - L_i^j(t+1)| \leq r. \tag{10}$$

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) [3]. This problem is called the **Unconstrained Placement Problem**. To simplify the technicalities of the analysis we do not allow placements to contain infinite number of resources, and bound the size of a placement L by a large enough **storage constant** s. That means, every placement should include up to $\sum_{j=0}^{k} L^j \leq s$ resources.

The focus of this paper is on the Constrained Reposition Problem, which we show to be hard to solve, and for which we provide an Heuristic Solution. The Unconstrained Placement Problem, in contrast, can be solved by a reduction to a min-cost flow problem (see [6]).

2.3. Assumptions of the marginal profit functions

Apart from the above general profit function structure we assume that given a demand D the 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. That means, a discrete function $f : \mathbf{N} \to R$ is considered to be concave if its differential $\Delta g(n) = g(n+1) - g(n)$ is monotonically non-increasing. This is a reasonable assumption as the profit may express diseconomy of scales; this results mainly from the fact that the expected value of the minimum (see left terms in the right hand side of Eq. (6), (7)) can be proven to be concave, and also since communications and control may become more difficult to handle as the number of resources grows.

The concavity property is essential for deriving a solution for the Constrained Reposition Problem (as described in Section 5).

Remark 2. The marginal cost functions $C()$ in our model allows to express dynamic physical limitations and bounds over the number of resources allocated. For example, if the system bounds at some point the number of resources in area j by s^j, then we set the area-j marginal cost function to be $C^j(x) = \infty$ for $x \geq s^j + 1$. Using such setting does not contradict the concavity of the marginal profit function.

3. Fundamental Sensitivity Properties of Resource Reposition

We start this paper with establishing two properties which are fundamental to the problem of resource placements and repositions. The properties deal with how sensitive are optimal placements to changes in the (stochastic) demand.

To establish these properties we use the following definitions:
Definition 1. *We use the following definitions:*

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-**CDF distance** is defined as the L_1-distance between the CDF vectors of N_1 and N_2 i.e.,*

$$d(N_1, N_2) = \sum_{n=0}^{\infty} |\Pr(N_1 \leq n) - \Pr(N_2 \leq n)|. \tag{11}$$

[3] Note that there might be multiple optimal placements.

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

2. *The demand sets $D = \{D_i^j\}$, $D' = \{D'^j_i\}$ 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 sets 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. (5)-(8) from Section 2 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 done here for simplicity of presentation and is done only in this section.*

3.1. Properties of L_1-CDF distance

First we compare the ϵ-near distance to the well-known Klomogorov-Smirnov (KS) test[7]. 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 are also obey the KS test.

Claim 3.1. *Let N_1 and N_2 be two discrete distributions. If they are strongly-ϵ near then they obey the KS test. .*

Proof. The KS test for 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-CDF distance that relates the CDF-distance to the difference of expected values in some particular cases.

Claim 3.2. *Let D_1 and D_2 be two discrete non-negative distributions. Then if D_1 dominates over D_2 then $d(D_1, D_2) = E(D_1) - E(D_2)$.*

Remark 3. Formally, distribution D_1 dominates D_2 if $\Pr(D_1 \geq n) \geq \Pr(D_2 \geq n)$ for every value $n \geq 1$.

Proof of Claim 3.2. In our proof we use the fact that the expectation of a non-negative positive distribution equals to:

$$E(X) = \sum_{k=1}^{\infty} \Pr(X \geq k) \tag{12}$$

If D_1 dominates D_2 then

$$d(D_1, D_2) = \sum_{n=0}^{\infty} |\Pr(D_1 \leq n) - \Pr(D_2 \leq n)| \underbrace{=}_{1-\Pr(D_i \leq n) = \Pr(D_i \geq n+1)} \sum_{n=1}^{\infty} |\Pr(D_1 \geq n) - \Pr(D_2 \geq n)| =$$
$$\underbrace{=}_{\text{Definition of dominance}} \sum_{n=1}^{\infty} \Pr(D_1 \geq n) - \sum_{n=1}^{\infty} \Pr(D_2 \geq n) \underbrace{=}_{\text{Eq. (12)}} E(D_1) - E(D_2). \tag{13}$$

□

The following claim states some particular cases where Claim 3.2 holds:

Claim 3.3. *The following holds: 1) If $D_1 \sim Poiss(\lambda_1), D_2 \sim Poiss(\lambda_2)$, where $\lambda_1 > \lambda_2$ then D_1 dominates D_2. 2) If $D_1 \sim Bin(n, p_1), D_2 \sim Bin(n, p_2)$, where $p_1 > p_2$ then D_1 dominates D_2.*

Proof. A proof can be seen in [8]. □

3.2. Sensitivity of repositions to demand fluctuations

The first property, stated in Theorem 3.4, establishes that optimal unconstrained revenue placements can be quite sensitive (in fact, infinitely sensitive) to changes in the demand. Thus, an algorithmic framework that will insist on finding optimal placement in reaction to any demand change may result with infinitely many resource repositioning and efforts.

Theorem 3.4 (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. That means that for every $\epsilon > 0$ there exist two demand sets 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' equals to the storage constant s (See Section 2.2), which can be infinitely large.*

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

The demand D' is defined as follows: 1) The probability that the demand of type-1 in area 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-CDF distance between D_1^1 and D'^1_1 is

$$d(D'^1_1, D_1^1) \underbrace{=}_{Eq.\ (11)} \sum_{n=0}^{\infty} |\Pr(D'^1_1 \geq n) - \Pr(D_1^1 \geq n)| = \frac{\epsilon}{2^{n+2}} \underbrace{=}_{\frac{1}{4}+\frac{1}{16}+\frac{1}{32}+\ldots=\frac{1}{2}} \frac{\epsilon}{2} < \epsilon.$$

2) The number of requests for type i resources in area j, where $(i,j) \neq (1,1)$ is 0 (i.e., $\Pr(D'^j_i = 0) = 1$). The L_1-CDF distance between D_i^j 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' equals to the storage constant s. □

The theorem implies that, similarly to the revenue placements, optimal unconstrained **profit** placements are sensitive to the demand distribution if the cost function C is small enough, in particular if it equals to zero:

Corollary 3.5. *For every system with revenue constants R_i, R_i^j there exists a cost function $C()$ where the optimal unconstrained **profit** placements (which maximize the profit) are sensitive to the demand D. That means that for every $\epsilon > 0$ there exist two demand sets D, D' with optimal unconstrained 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' equals to the storage constant s (See Section 2.2), which can be infinitely large.*

3.3. Sensitivity of the 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 reaction 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. (6) and (7), regardless of the cost. We use the following definitions to formulate the theorem:

Definition 2. *Let $D = \{D_i^j\}$, $D' = \{{D'}_i^j\}$ be two demand sets.*

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

$$d(D, D') = \sum_{i=1}^{m} \sum_{j=1}^{k} d(D_i^j, {D'}_i^j) R_i^j + \sum_{i=1}^{m} d(D_i, {D'}_i) R_i, \tag{14}$$

   where R_i^j and R_i are the revenue constants in Eqs. (2) and (3).
2. *Demands sets D and D' are called **weakly ϵ-near** if the distance between them is less than ϵ, i.e., $d(D, D') < \epsilon$.*
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 of D minus ϵ. That is $P(L, D) \geq \max_{L'} P(L', D) - \epsilon$.*

The second sensitivity theorem is presented next:

Theorem 3.6 (Sensitivity in placement profit)**.** *Let ϵ be the threshold parameter, and let $D(t)$ and $D(t+1)$ be two demand sets 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 4. Using simple inequalities on the profits of the optimal allocations on $D(t)$ and on $D(t+1)$ it is possible to provide a short proof that $L(t)$ is an 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 in Section 5.

Proof. The proof is quite involved and is supported by Claims 3.7, 3.8, 3.10 and Theorem 3.9 which are presented (and proved) below, during the development of the proof. The proof uses a mapping of the Unconstrained Reposition Problem to the min-cost flow problem over a 4-layer multigraph (graph with parallel edges between vertices) G^4 (see Figure 3) 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 (See Section 2.2), which is a large as needed constant. In every edge we can transfer a single unit of flow or not transfer flow at all, namely, the edge capacity is 1. In this multigraph every placement L is represented by a certain flow f_L, so that the quantities of a placement ($L = (L_i^j)$) are the flow values between region nodes j and resource type nodes i, i.e., the number of edges with a single unit of flow from j to i. Similarly the flow between x and an area node j represents the number of resources allocated in area j, and the flow between resource type node i and sink y represents the number of resources allocated of resource type i. In addition, the graph contains s edges from x to y, where the flow on these edges represent resources (out of s) that are not utilized. The full details of the reduction are given in the appendix (Appendix B).

Example 3.1. 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 respectively resource types α, β are $L_\alpha^a = L_\beta^a = 1$, while the number of resources in b are $L_\alpha^b = 0, L_\beta^b = 1$. The flow of the associated placement f_L is presented in Figure 3, where the red edges are edges with a single unit of flow, and the blue edges are with 0 unit of flow. Between every two nodes if there are n red edges, then there are $s - n$ blue edges. In the diagram the total flow is $s = 4$ (for example, there are three blue edges between x and y and one red edges).

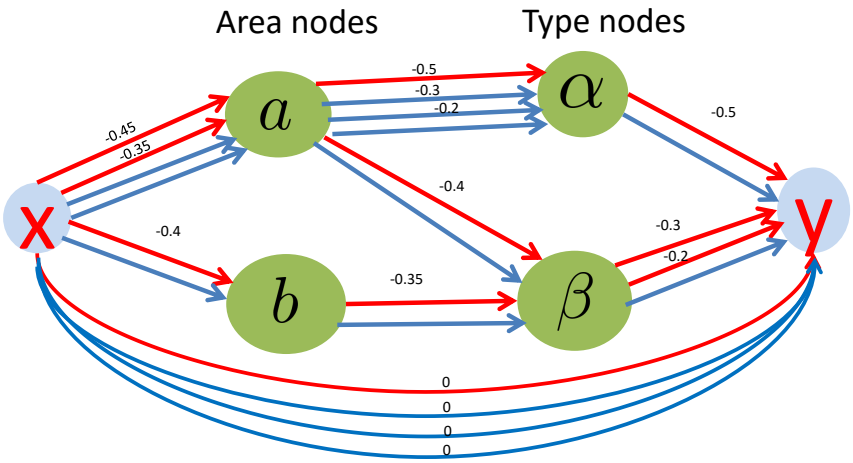

Figure 3: The flow f_L in the multigraph G^4 where $L^a_\alpha = L^a_\beta = 1$ and $L^b_\alpha = 0, L^b_\beta = 1$ (Example 3.1). Red edges are edges with a single unit of flow, and blue edges with 0 unit of flow. Between every two nodes there are totally $s(= 4)$ edges. We omit some blue edges from the graph for the sake of presentation.

In the min-cost flow problem one places weights on the edges and defines the *flow weight* (or *flow cost*) as the sum of costs on the edges with non-zero flow. For example the flow weight of the graph in Figure 3 is -3.45. Then one considers all flows whose *flow value* (or *required flow*), which is the total flow from source node x to its neighbors, is fixed (say s) and seeks over all these flows the one that minimizes the flow weight.

We construct the mapping by defining the weights $w(e)$, such that, roughly speaking, the weight of the i^{th} edge between a region node (say a) and 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 equals to the profit differential $-\Delta\zeta^a_\beta(n, D) = -(1) \cdot (\zeta^a_\beta(n, D) - \zeta^a_\beta(n-1, D))$. The edge weights are multiplied by minus one as we convert a maximum profit placement problem into a minimal cost flow problem. Similarly edges from the source node x to an area 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.

The following claim establishes how the profit of a placement relates to the cost of its corresponding flow:

Claim 3.7. *Let L be a placement. Then its profit equals to a constant c minus the weight of its corresponding flow f_L i.e.,*

$$P(D, L) = c - W(f_L, w(D)), \tag{15}$$

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. A detailed rigorous proof can be found in Appendix C. It is based in the fact that the sum of

differential weights

$$\begin{aligned}(-1)\cdot\Delta\zeta(0,D)&=\zeta(0,D)-\zeta(1,D),\\(-1)\cdot\Delta\zeta(1,D)&=\zeta(1,D)-\zeta(2,D),\\&\vdots\\(-1)\cdot\Delta\zeta(L_i^j-1,D)&=\zeta(L_i^j-1,D)-\zeta(L_i^j,D),\end{aligned}$$

forms a telescoping series. We denote $W(v_1,v_2,f_L)$ the total flow weight of f_L between v_1 to v_2. So for example, if a is a region node and β is a resource type node we show that $W(a,\beta,f_L)$ is a sum of terms in the above equations (i.e. $W(a,\beta,f_L)=\sum_{n=0}^{L_\beta^a-1}(-1)\cdot\Delta\zeta_\beta^a(n,D)$), and therefore, breaking ζ to its components we get:

$$\zeta^a(L^a)=\zeta^a(0)-W(x,a,f_L), \tag{16}$$
$$\zeta_\beta^a(L_\beta^a,D)=\zeta_\beta^a(0,D)-W(a,\beta,f_L), \tag{17}$$
$$\zeta_\beta(L_\beta,D)=\zeta_\beta(0,D)-W(\beta,y,f_L). \tag{18}$$

If we sum these equations for every resource type and every region, we will get the desired result. Note that according to Eq. (6)-(8) in Section 2, $\zeta^a(0),\zeta_\beta^a(0,D),\zeta_\beta(0,D)$ neither depend on the demand D, nor on the placement L. Thus the constant c, which is the sum of these terms, neither depend on the demand D, nor on the placement L.

□

Having established how the profit relates to the corresponding flow we next observe that if one aims at finding the minimal weighted flow (among all flows of the same flow value) one would pick in the graph the edges whose weights are the smallest - that is those resources which profit the placement the most. Note that the weight of the direct edge from x to y is zero - leaving no incentive to direct the flow through that edge. Thus the solution of the unconstrained optimal placement L_{opt} is corresponding to the flow f_{min} which is of minimal weight (i.e., the flow that minimizes the sum of red edges weights), as seen in the following claim:

Claim 3.8. *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 unconstrained placement for demand D.*

Proof. The correctness of the claim stems from the following three facts: 1) the marginal profit functions ζ are concave functions (See Section 2.3). I.e., $\Delta\zeta$ are monotonically non-increasing functions (where $\Delta\zeta(n,D)=\zeta(n+1)-\zeta(n)$). 2) if the capacities of a min-cost flow network are integers, then its min-cost flow f has an integer value flow. The latter is a well-known theorem [a proof can be seen in [9], Theorem 9.10 (Integrality Property)]. 3) By Claim 3.7. The detailed proof is given in Appendix D. □

To continue proving Theorem 3.6, 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. According to Eq. (15) the change in the flow weight between $f_{opt}(t)$ and $f_{opt}(t+1)$ with respect to weight $w(t+1)$ equals 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 3.6 we only need to show the following equation:

$$W(f_{opt}(t), w(t+1)) - W(f_{opt}(t+1), w(t+1)) \leq d(D(t), D(t+1)). \tag{19}$$

To bound the difference $W(f_{opt}(t), w(t+1)) - W(f_{opt}(t+1), w(t+1))$ we use the following graph-theoretic result proven in [10]:

Theorem 3.9. *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} be respectively their integer min-cost flows of the same flow value ($|f_{min}| = |f'_{min}|$)[4]. Then the flow weight of f_{min} with respect to the new weight w' is bounded by the flow weight of f'_{min} respect to w', plus the sum over all edges in G of the edge-weight deviations $|w'(e) - w(e)|$. That means*

$$W(f_{min}, w') - W(f'_{min}, w') \leq \sum_{e \in E} |w'(e) - w'(e)|. \tag{20}$$

Proof. In [10]. □

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 3.10. *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.,*

$$\sum_{e \in E} |w'(e) - w'(e)| \leq d(D(t), D(t+1)). \tag{21}$$

Proof of Claim 3.10. Below we provide the main arguments of the proof. The full detailed proof is given in Appendix E.

We first express the edge weights $w(e)$ by the model parameters. To compute the differential of the marginal functions ζ_i^j, ζ_i, we use the following formula to compute the partial expectation (i.e., $E_X(\min(n, X))$) of a discrete non-negative distribution by its CDF values $\Pr(X \geq n)$:

$$E_X(\min(n, X)) = \sum_{k=1}^{n} \Pr(X \geq k) = \sum_{k=0}^{n-1} (1 - \Pr(X \leq k)). \tag{22}$$

The formula was Proven in [11] Claim 6.4. The exact expression of the edge weights $w(e)$, expressed by the demand CDF, and utilizing Eq. (22), can be found in Appendix E.

[4] As stated above the min-cost flow can be derived with respect to any flow value s. Thus we can set the flow values of both min-cost problems to be equal to the same s, that is $|f_{min}| = |f'_{min}| = s$.

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

$$\sum_{e \text{ connects } a \text{ to } \beta} |w(e) - w'(e)| \leq d(D^a_\beta(t), D^a_\beta(t+1))R^a_\beta, \tag{23}$$

$$\sum_{e \text{ connects } \beta \text{ to } y} |w(e) - w'(e)| \leq d(D_\beta(t), D_\beta(t+1))R_\beta, \tag{24}$$

$$\sum_{e \text{ connects } x \text{ to } a} |w(e) - w'(e)| = \sum_{e \text{ connects } x \text{ to } y} |w(e) - w'(e)| = 0. \tag{25}$$

These inequalities holds since as the L_1-CDF 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 CDF 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 between x and a.

Therefore we conclude that

$$\sum_{e \in E} |w(e) - w'(e)| =\leq \underbrace{\sum_{i=1}^{m}\sum_{j=1}^{k} d(D^j_i(t), D^j_i(t+1))R^j_i}_{\text{Eq. (23)}} + \underbrace{\sum_{i=1}^{m} d(D_i(t), D_i(t+1))R_i}_{\text{Eq. (24)}} = d(D(t), D(t+1)). \tag{26}$$

as required. □

Finally, from Claim 3.10 and Theorem 3.9 we imply Eq. (19) and consequently Theorem 3.6 is proven.

□

From Theorem 3.6 the following is implied:
Corollary 3.11. *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. One can create D' from D by adding to the pdf of D an $\epsilon/2$ mass at some position k and subtracting it at position $k+1$. Following Theorem 3.6 the corollary holds. □

successive

4. Hardness of the Constrained Reposition problem

In this section we prove the hardness of the Constrained Reposition Problem; specifically, we show that if there is an optimal polynomial solution for the Constrained Reposition Problem - then there is a polynomial solution to the Exact Perfect Matching Problem [5]. The question whether the Exact Perfect Matching problem has a polynomial time solution is considered to be an old open problem in graph theory for almost 30 years ([12], [13] [14]). One may therefore conjecture that the Constrained Reposition problem is a hard

[5]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.

problem. We show that the hardness of conjecture regarding the Constrained Reposition problem remains true even if the marginal profit functions ζ_i^j (defined in Eqs. (6)-(8)) are linear functions with limitations (See Remark 5 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 the Constrained Reposition Problem. Then, in Appendix F 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).

4.1. The Fair Christmas Game Problem and its reduction to the Constrained Reposition Problem

The Fair Christmas Game Problem is formulated for the sake of proving that the Constrained Reposition Problem 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 to a friend only a single gift, 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 the Constrained Reposition Problem by showing that every instance of the former can be formulated as an instance of the latter, and thus conclude that the Constrained Reposition Problem is hard at least as the Fair Christmas Game Problem:

Theorem 4.1. *There is a polynomial reduction from the Fair Christmas Game Problem to the Constrained Reposition Problem.*

Proof. We construct the reduction by creating an instance of the Constrained Reposition Problem 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 was given a gift yet). The placement $L(t+1)$ resulting from the reposition is corresponding 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 0 or 1 (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 construction will force the

solution of the optimal constrained-reposition problem to be balanced and $\hat{S}$-valid. The functions are selected as follows:

$$\zeta_i(x, D_i) = \zeta^j(x) = \begin{cases} x \cdot (n^2+1) & \text{if } x \leq n. \\ -\infty & \text{otherwise.} \end{cases} \tag{27}$$

$$\zeta_i^j(L_i^j, D_i^j) = \begin{cases} 0 & \text{if } i = j \text{ or } L_i^j = 0. \\ 1 & \text{if } i \neq j,\ j \in S(i) \text{ and } L_i^j = 1. \\ -\infty & \text{otherwise.} \end{cases} \tag{28}$$

It is easy to see that under these functions the following holds:

1. The functions are concave and are legitimate for using in the constrained-reposition problem (where we set the revenue constants in Eqs (6), (7) to be zero, i.e., $R_i = R_i^j = 0$).
2. The profit of $L(t)$ according to Eq. (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. (28).
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. (27), and Eq. (28) 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 imply 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 (equals 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 equals 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, $\sum_i \sum_j \zeta_i^j(L(t+1)_i^j, D_i^j)$ equals 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 imply the following claims:

Claim 4.2. *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 respected Fair Christmas game is k.*

Claim 4.3. *Given a friend list $\hat{S}$, any fair game corresponding to $\hat{S}$ where k gifts 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 5. The marginal profit $\zeta_i, \zeta^j, \zeta_i^j$ (Eqs. 27, 28) are all linear function with limitations over the number of type i resources, the number of area j resources, and type i area j resources.

4.2. Hardness of the Constrained Reposition Problem

The following theorem, proved in Appendix F, 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 [12].

Theorem 4.4. *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.[12] showed that the Exact Cycle Sum Problem is hard at least as the Exact Matching Problem:

Theorem 4.5 ([12])**.** *There is a polynomial reduction from the Exact Matching Problem to the Cycle Sum Problem.*

Theorems 4.4, 4.5 imply that a polynomial solution for Fair Christmas Game Problem implies a polynomial solution for the Exact Matching Problem[6]. Combining it with Theorem 4.1 implies the hardness of Constrained Reposition Problem as follows:
Corollary 4.6. *The Constrained Reposition Problem is hard as the Exact Matching Problem. I.e., if there is a polynomial solution to the Constrained Reposition Problem - then there is a polynomial solution to the Exact Matching Problem.*

Finally, to imply the hardness of the Constrained Reposition Problem we observe that solving polynomially the Exact Matching Problem is a studied and a well-known problem which is open for almost 30-years ([12], [14], [13]), 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.

5. The Shortest Cycle Operation (SCO) algorithm

As there is no polynomial time solution for the Constrained Reposition Problem (as shown in Section 4) we present a polynomial time heuristic algorithm called **SCO** that solves the problem. We prove that SCO finds an optimal solution in the following cases: 1) If the reposition constraint r is very large; in particular SCO can find the unconstrained optimal placement ($r = \infty$). 2) If the reposition constraint r is very small. 3) In a single region system or a single type system.

5.1. Description of SCO

To describe SCO we use the following definitions:
Definition 3. *An **operation** o is a composition of unit operations (i.e., addition or subtraction operations). The **length** m of an operation o is the number of repositions the operation do.*

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 $m = 3$. Note that an operation that is a composition of m unit operations can be of length smaller than m, if the operations cancel each other.

[6]It can be shown that the problems are equivalent, but for the sake of this work this is not required.

Definition 4. *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 equaling 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 type 1 resource 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 5. *Given a placement L a **shortest profitable operation** of L is a profitable operation o of length m such that: 1) The length of every profitable operation o' is not less than m. 2) If o' is an operation of length m, then o is more profitable than o', i.e., $\Delta(o, L) \geq \Delta(o', L)$.*

Note 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: 1) We set initially set $A(0) \leftarrow L(t)$. 2) In every iteration i we find the shortest profitable operation o_i for placement $A(i)$ and set $A(i+1) \leftarrow o_i(A(i))$. We terminate if $A(i)$ is the optimal unconstrained solution of $D(t+1)$, or if the reposition constraint will be violated. In the next subsections we 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 equal respectively to 5 and 3. The profit function is $P(L, D) = \min(L_1^1, 5) + \min(L_1^2, 3) + m_1 \cdot \min(L_1, 8) - m_2 \cdot L_1^1$, where $m_1 >> m_2 >> 1$ are constants. The initial placement is set to be $A(0) = L(t)$, where $L_1^1(t) = 6, L_1^2(t) = 1$ and the reposition constant 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 equals respectively to $m_1 - m_2 > 0$ and $1 + m_1 - m_2 > 0$. The profitability of removing a single resource from $A(0)$ in regions 1 and 2 is respectively $m_2 - m_1 < 0$ and $m_2 - m_1 - 1 < 0$. 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 not the optimal unconstrained solution.

An addition or a subtraction operation over $A(1)$ is not profitable: adding a resource has a profitability $c - m_2 < 0$ where c is a constant, while removing a resource has a profitability of $m_2 - m_1 + c < 0$. Thus, there is no profitable unit operation (i.e., a profitable operation of length 1).

The SCO thus finds a shortest profitable operation of length $m \geq 2$. One might check that SCO in the second iteration adds to region 2 and removes from region 1. SCO will terminate after two iterations, as $A(2) = (5, 3)$ is the optimal unconstrained solution.

Remark 6. If the reposition constant 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 constant larger than 3 SCO returns $A(2) - (5, 3)$

5.2. Key properties of the SCO algorithm

Next, we establish the key properties of SCO presented in the following theorems:

Theorem 5.1 (Optimality of SCO in single region or single type)**.** *In systems with a single region or with a single resource type SCO will return an optimal solution for the Constrained Reposition Problem.*

Proof. For a single type system, [15] showed that a greedy algorithm that first conducts a sequence of length $m = 1$ (as long as they are profitable) followed by a sequence of operations of length $m = 2$ (as long as they are profitable) yields the optimal solution of the Constrained Reposition Problem. It is easy to see that SCO follows that algorithm and therefore yields the optimal result. Similarly for a single type system. □

Theorem 5.2 (Optimality of SCO for small reposition constants)**.** *For every system there is a constant r_1 such that for every reposition constant $r \leq r_1$ SCO finds an optimal solution for the Constrained Reposition 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 m. Then for every $r \leq m-1$ there is no profitable operation of length r that can improve the profit of $L(t)$. To this end, we set $r_1 = m-1$, and for every $r < r_1$ SCO finds an optimal solution for the Constrained Reposition Problem[7]. □

Theorem 5.3 (Optimality of SCO for large reposition constants)**.** *For every system where an optimal solution for the Unconstrained Placement Problem exists there is a constant r_2 such that for every reposition constant $r \geq r_2$ SCO finds an optimal unconstrained solution. In particular, for $r \geq r_2$, SCO solves the Constrained Reposition Problem.*

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 order all possible placements by their profit (according to demand D). That means, $P(L_1, D) \geq P(L_2, D) \geq P(L_3, D) \ldots$ where L_1 is the optimal unconstrained solution. 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 note 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., $rank(L') \leq rank(L) - 1$).
- 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 imply 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 constant 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 constant 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 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. □

[7] It can be shown that for $r_1 = m$ SCO finds an optimal solution for the Constrained Reposition Problem.

5.3. Implementation of the SCO algorithm - outline

Finding the shortest profitable operation is not trivial. The number of available operations equals 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. In the next subsection we propose a polynomial time algorithm that does address this problem, namely that given a placement L over demand D it finds its shortest profitable operation.

Our strategy is as follows: First (Section 5.4) 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 5.5) we analyze the profit of these extended chains and, roughly speaking, show that the profit consists of the sum of marginal profits of each of the move operations. Lastly (Section 5.6) we use this information to construct a graph (see Fig. 4), whose nodes represent the regions and an edge between node i and j represents the move operation from node 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.

5.4. 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 6 (Extended chain operation)**.** *A **move operation** of format $O(j_1 \to j_2)$ is an operation that moves a resource from region j_1 to j_2*[8]*. A **chain operation** of format $O(j_1 \to j_2 \to \ldots \to j_n)$ 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 $O(j_1 \to j_2 \to \ldots \to j_n)$ 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** $E(j_1 \to j_2 \to \ldots \to j_n)$ is a simple chain operation $O(j_1 \to j_2 \to \ldots \to j_n)$ that in addition, can possibly conduct either (non exclusive) one of the following two unit operations: 1) add a resource to region j_1, 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 $O(a \to b)$ 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 $E(a \to b)$ 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 7. 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 $E(j_1 \to j_2 \to \ldots \to j_n)$ can do is either $2(n-1)$ (if the operation does not adds a resource to j_1 nor removes 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).

[8] 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 to extended chains operations.

Theorem 5.4 (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 5.4, 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 3) is postponed to Section 5.7, 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 Section 3, based on the concavity assumption of the marginal profit functions (Section 2.3). 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 algorithm and less of graph theory results, we will present the correctness of the theorem later in Section 5.7.

Based on Theorem 5.4, 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 graph. In order to compute the edge weights in the graph, we study in the next subsection the structure of the profitability of the extended chain operations as computed by the algorithm.

5.5. Profitability structure of an extended chains

To express the profitability structure we recall that the profit function $P(D,L)$ is composed of the sum of marginal functions $\zeta_i(D_i,L_i), \zeta_i^j(D_i^j,L_i^j), \zeta^j(L^j)$ (Eq. (5)). We denote $g_i(n)=\zeta_i(n,D_i^j), g^j(n)=\zeta^j(n,D_i^j)$ and $g^j(n)=\zeta^j(n)$, so that

$$P(L,D)=\underbrace{\sum_{i=1}^{m}\sum_{j=1}^{k} g_i^j(L_i^j)}_{\text{area \& type}}+\underbrace{\sum_{i=1}^{m} g_i(L_i)}_{\text{type}}+\underbrace{\sum_{j=1}^{k} g^j(L^j)}_{\text{area}}, \tag{29}$$

and denote the **differential** of a discrete function g as $\Delta g(n)=g(n+1)-g(n)$. In the next lemma we will use the differential of area & type functions Δg_i^j and the differential of area 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 $O(j_1 \to j_2 \ldots \to j_n)$ (that neither add resource to j_1 nor removes from j_n). For sake of presentation brevity, we show next the profitability structure for this class of operations. In Appendix G, we show the profitability structure for all five classes.

Lemma 5.5. *Let o be an extended chain operation $O(j_1 \to j_2 \to \ldots \to j_n)$ that neither adds a resource to j_1 nor removes 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*

$$\Delta(o,L)=\Delta^+(j_1)+\sum_{k=1}^{n-1}\Delta_{i_k}(j_k \to j_{k+1})+\Delta^-(j_n). \tag{30}$$

where,

$$\Delta_i(j_1 \to j_2) = \Delta g_i^{j_2}(L_i^{j_2}) - \Delta g_i^{j_1}(L_i^{j_1} - 1) \tag{31}$$

$$\Delta^+(j) = -\Delta g^j(L^j - 1) \tag{32}$$

$$\Delta^-(j) = \Delta g^j(L^j). \tag{33}$$

Intuitively, $\Delta_i(j_1 \to j_2)$ represents the change in the area & type part of Eq. (29) affected by the moving a type i resource from j_1 to j_2. The terms $\Delta^+(j), \Delta^-(j)$ represent the change in the area part of Eq. (29) affected by respectively removing and adding a resource to area j.

Proof. By the definition of profitability and Eq. (29), the profitability of o over L (denoted by $o(L)$) equals to the sum of three terms: 1) sum of area & type marginal function 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) Sum of type marginal function differentials $g_i(o(L)_i) - g_i(L_i)$ for all resource types i and 3) sum of area marginal function differentials $g^j(o(L)^j) - g^j(L^j)$ all regions j.

First, we consider the effect of a move operation over the area & 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 to $g_i^j(L_i^j + 1) - g_i^j(L_i^j) = \Delta g_i^j(L_i^j)$. For $j = j_1$, the term $g_i^j(o(L)_i^j) - g_i^j(L_i^j)$ equals to $g_i^j(L_i^j - 1) - g_i^j(L_i^j) = -\Delta g_i^j(L_i^j - 1)$. Thus the move operation contributes to the area & type terms $\Delta g_i^{j_2}(L_i^{j_2}) - \Delta g_i^{j_1}(L_i^{j_1} - 1) = \Delta_i(j_1 \to j_2)$. 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 area & 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 do not change, and $g_i^j(o(L)_i^j) - g_i^j(L_i^j) = 0$.

Next consider the area function term (sum of $g^j(o(L)^j) - g^j(L^j)$): For these we note that for every region $j \neq j_1, j_n$ the number of resources in region j do not changed, and $g^j(o(L)^j) - g^j(L^j) = 0$. The operation o removes a resource in j_1 and add a resource addition in j_n. Thus, the sum over regions $g^j(o(L)^j) - g^j(L^j)$ over all regions j equals to $\Delta g^{j_n}(L^{j_n}) - \Delta g^{j_1}(L^{j_1} - 1) = \Delta^+(j) + \Delta^-(j)$, as required.

Lastly, consider the type function term (sum of $g^j(o(L)^j) - g^j(L^j)$). For these we note that the total number of resources from each type does not change. Thus it is equal to zero.

□

Remark 8. Lemma 5.5 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 5.5 its profitability depends on the differential functions Δg, which is not necessarily equal zero[9]. However operation o is the identical operation ($o(L) = L$ for every placement L), and thus its profitability equals to zero.

5.6. The algorithm for finding a shortest profitable operation

As the profitability structure of an extended chain operation is 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

[9]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 equals to $(-1) \cdot (L^{j_1} - 1) + 0 + L^{j_1} = 1$

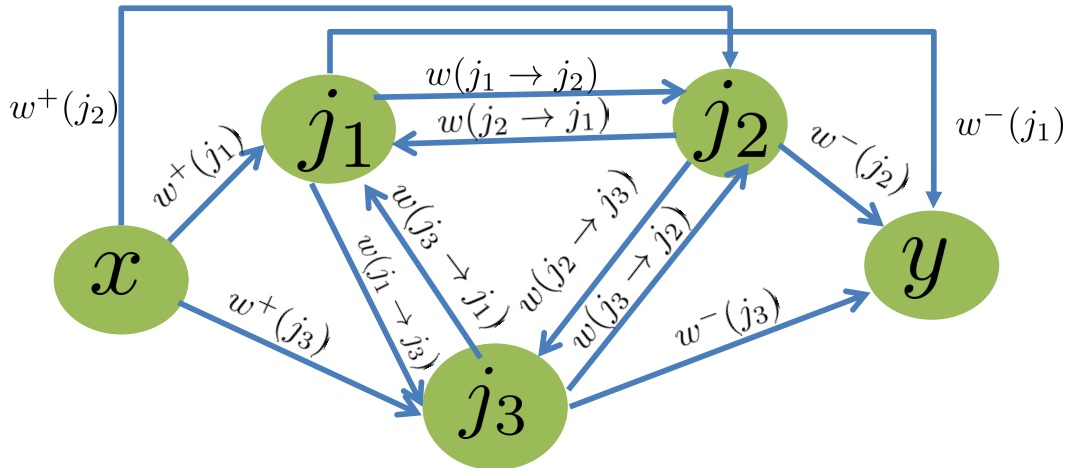

Figure 4: The graph with three regions j_1, j_2, j_3 and with the appropriate weight edges $w(j_1 \to j_2) = (-1) \cdot \max_i \Delta_i(j_1 \to j_2)$, $w^+(j) = (-1) \cdot \Delta^+(j)$, $w^-(j) = (-1) \cdot \Delta^-(j)$.

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 H. For sake of presentation, we will show how the algorithm finds the shortest profitable operation over the class of chain operations $O(j_1 \to j_2 \to \ldots \to j_n)$. Our full algorithm is given in Appendix H.

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 Fig. 3 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. (31)). Note that we multiply $\Delta_i(j_{l_1} \to j_{l_2})$ by (-1) as we convert a max profitable problem into a shortest path problem (that requires finding a minimal 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. (32), (33)). The graph G with its edge weights is depicted in Figure 4.

Note that a path $P = (x, j_1, j_2, \ldots, j_{i_{min}-2}, y)$ in the graph is corresponding to the best chain operation o of format $O(j_1 \to j_2, \ldots \to j_{i_{min}-2})$ and according to Lemma 5.5 the weight of the path P equals to (-1) multiplied by the profitability of o. We will use this observation to find the shortest profitable operation.

After constructing the graph G our algorithm considers the edge weights as edge lengths and uses the Bellman-Ford algorithm [16], which computes in the i^{th} iteration the shortest path between x and y that uses at most i edges [10]. 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 $O(j_1 \to j_2, \ldots \to j_{i_{min}-2})$ that moves from region j_k to j_{k+1} a resource of type $i(j_k \to j_{k+1})$. According to the next claim, the algorithm finds a shortest profitable operation:

Claim 5.6. *The algorithm finds a shortest profitable operation over the class of chain operations* $O(j_1 \to j_2 \to \ldots \to j_n)$.

[10] 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.

Proof. According to Lemma 5.5, the profitability of an operation of format $O(j_1 \to j_2 \to \ldots \to j_n)$ should be less than or equals 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 $i(j_k \to j_{k+1})$ from j_k to j_{k+1} equals exactly 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 $n+2$ $((x, j_1, j_2, \ldots, j_{n-2}, y))$ equals to the profitability of the best extended chain operation of length n $(O(j_1 \to j_2 \to \ldots \to j_n))$ 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 a profitable operation with minimal length, then it is corresponding to a negative weight shortest path with minimal number of edges, which is what the algorithm finds. □

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 $w(j_1 \to j_2)$ takes $O(m)$. Using Bellman-Ford on a graph $G = (V, E)$ takes at most $O(VE)$ 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 remove from j_n not is $O(k^2(k+m))$.

In Appendix H we show that finding the algorithms for shortest profitable operation in three other classes are similar to this algorithm and they run at $O(k^2(k+m))$ steps. Finding the shortest profitable operation among the operations of the fifth class, however, is solved in a more expensive time complexity of $O(k^2(k^2+m))$ instead of $O(k^2(k+m))$. The details of this algorithm can be found in Appendix H. Thus, the full algorithm which finds and compares the shortest profitable operations in each class takes $O(k^2(k^2+m))$ steps.

Remark 9. We can use a similar algorithm that instead of using region nodes $1, 2 \ldots, k$ it uses resource type nodes $1, 2, \ldots, m$. The complexity of such 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.

5.7. Every shortest profitable operation is an extended chain - proof of Theorem 5.4

Proof of Theorem 5.4. We use the reduction to a min-cost flow described in Theorem 3.6 and Figure 3. The methodology of reduction to a graph flow problem allows us to conduct the analysis in the domain of graphs theory. Given a flow f_L corresponding to placement L in the 4-layer graph G^4 we use the **residual multigraph** $G^4_{f_L}$ that will allow to represent all possible operations o (placement changes) on L by augmenting flows through 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 4-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 3 is presented in Figure 5. The formal definition of a residual multigraph G_f for a general (multi) graph G and flow f is given in Appendix I.

When some operation o changes a placement L to placement $o(L)$ the flow corresponding to L, f_L, is changed to the flow corresponding to $o(L)$, $f_{o(L)}$. The flow value of both flows equals 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 can materialize by augmenting the flow through cycles $C_1, C_2 \ldots C_n$ in the residual multigraph G_f [See The Augmenting Cycle Theorem (Theorem Appendix I.2) in Appendix I]. We define the

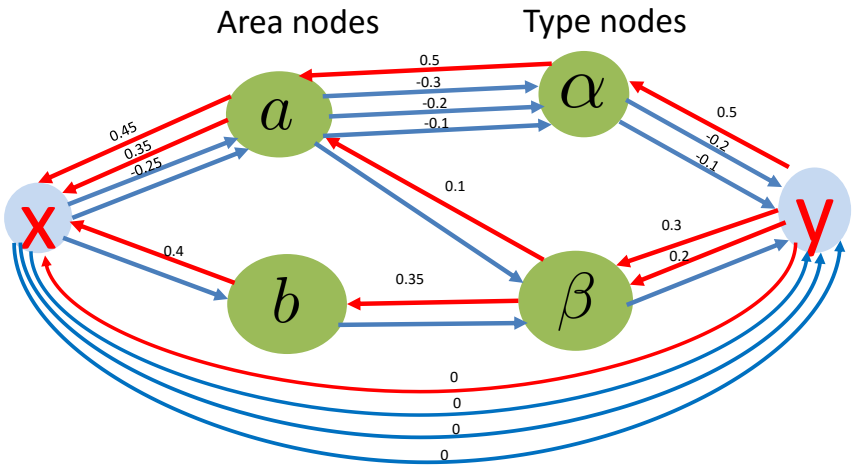

Figure 5: The residual multigraph $G^4_{f_L}$ where $L^a_\alpha = L^a_\beta = 1$ and $L^b_\alpha = 0, L^b_\beta = 1$ (the flow f_L is depicted in Figure 3). Red edges are reverse edges with flow $f_L(e) = 1$, and blue edges are original edges with flow $f_L(e) = 0$. Between every two nodes there are totally $s(= 4)$ edges. We omit some blue edges from the graph for the sake of presentation.

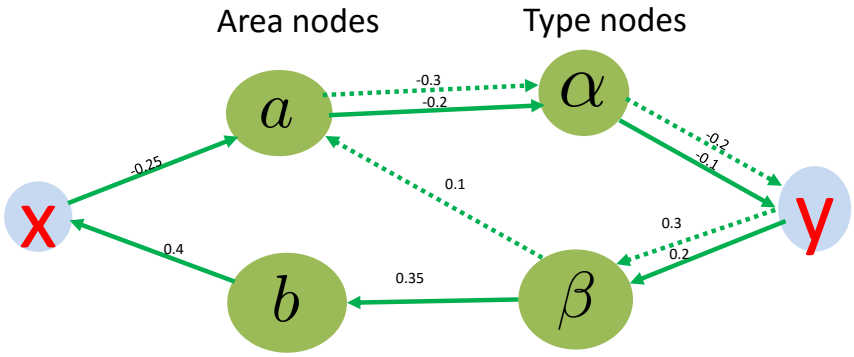

Figure 6: The residual operation multigraph $G(o, L)$, a subgraph of $G^4_{f_L}$ from Figure 5, where o is an operation that adds two α-type resources to region a, and removes two β-type resources, one from region a and one from region b. The dotted edges represent the lowest weighed cycle C in $G(o, L)$. An augmentation through C will add one α resource to a and remove one β resource from a (this operation is $o^*(o, L)$).

residual operation multigraph $G(o, L)$, a sub-graph of $G^4_{f_L}$, as the union of cycles such that augmenting one unit of flow through them changes the flow f_L to $f_{o(L)}$ in $G^4_{f_L}$, i.e.,

$$G(o, L) = \bigcup C_i. \tag{34}$$

A detailed description the structure (and how to build) $G(o, L)$ is given in Appendix J (Observation Appendix J.1) and an example is given in Example 5.1 below. Since the weight of a flow f_L is minus the profit of L plus some constant c (see Claim 3.7), then the sum of edge weights of the residual operation multigraph is the minus the operation profitability, i.e.,

$$\sum_{e \in G(o,L)} w_f(e) = -\Delta(o, L). \tag{35}$$

Next, we define a **lowest weight cycle** C as a simple cycle with minimal weight in $G(o, L)$; note that $G(o, L)$ might have more than one such cycles.

Example 5.1. Let $G^4_{f_L}$ be the residual graph depicted in Figure 5, which is corresponding 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 3. 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 6, where every original edge is corresponding to adding a resource, while every reverse edge is corresponding to removing a resource. Note that the residual multigraph consists of 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 in dotted 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 unit operations (operations that either add or remove a single resource) from o'. For example, consider o in Figure 6 consisting of adding two α-type resources to region a, and removing 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 the operation o' is corresponding to augmenting flow trough the lowest weight cycle C. Also, note o' is an extended chain of zero length (See Remark 7 in Section 5.4).

This leads to our first lemma used to prove Theorem 5.4:

Lemma 5.7. *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 is corresponding 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 and a full formal proof can be seen in Appendix J.

The proof first characterizes the structure of the residual operation multigraph $G(o, L)$ (see Observation Appendix J.1). 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 contains 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 Appendix J.2).

Next, we characterize all cycles in the residual operation multigraph $G(o, L)$ (Claim Appendix J.3) 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 class 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

resources types. If the lowest weight cycle C is a cycle of such 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 5.4 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 5.8. *Let L be a placement and let o_L be its shortest profitable operation. Then the following claims hold:*

1. *The operation o_L equals to the operation $o'(o_L, L)$ chosen in Lemma 5.7.*
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 $w_f(C_i) \geq 0$. Thus the profitability of o_L, which equals 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. (35),(34)), is non-positive - a contradiction that o_L is shortest profitable, which is in particular a profitable operation. Thus, the operation $o'(o_L, L)$, which is corresponding 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 uses strictly less 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 less repositions than o - a contradiction as o is a shortest profitable operation. Therefore, o must be an extended chain operation. □

□

6. 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.

Two challenges must be faced by such 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 desired 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 SCO algorithm developed in Section 5, and the sensitivity results developed in Section 3. When it senses (using the sensitivity results) large deviation in the demand it conducts SCO and repositions as many resources as possible (up to the maximal number possible, r_{max}). When it senses small deviation in the demand it conducts only minimal (or zero) 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 set $D_{ref}(t)$ equaling $D(\tau)$ for some $\tau < t$, where τ is the last time where the algorithm conducted a large reposition. It also holds as reference the placement at time $t - 1$, $L(t - 1)$. Given distributions $D_{ref}(t), D(t)$ and placement $L(t - 1)$ the algorithm computes

the placement at period t, $L(t)$. The algorithm determines whether $D(t)$ and $D_{ref}(t)$ are weakly ϵ-near. If they are - it conducts a "minimal" reposition by running SCO over placement $L(t-1)$ (with its associated flow), with the reposition cost threshold set to r_{min}. Otherwise, it conducts a "maximal possible" reposition by running SCO over placement $L(t-1)$ with the reposition cost threshold set to r_{max} and then resets $\tau = t$.

Note that if we set $\epsilon = 0$, Hybird is equivalent to SCO with r_{max}. If we set $\epsilon = \infty$, Hybird is equivalent to SCO with r_{min}.

7. Model extensions

In this section we study two extensions of the model studied in this paper.

7.1. The Unconstrained Placement Problem with Reposition Costs

A related problem to the Constrained Reposition Problem is whereby repositions are attributed with (linear) costs rather than being considered as a constraint.That is, in that 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 to capture costs associated with reposition 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 equals to $\pi_i^j \geq 0$ while the cost of removing 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 $t+1$ equals to

$$Rep(t)_i^j(L_i^j(t+1)) = \pi_i^j \cdot \underbrace{\max(L(t+1)_i^j - L(t)_i^j, 0)}_{\text{number of type-}i\text{ resources added to region }j} + \theta_i^j \cdot \underbrace{\max(L(t)_i^j - L(t+1)_i^j, 0)}_{\text{number of type-}i\text{ resources removed from region }j}. \tag{36}$$

The Unconstrained Placement Problem with Reposition costs is formulated as follows: Given the placement $L(t)$ of the previous period and the new demand $D(t+1)$, find the placement $L(t+1)$ that maximizing the profit $P(D(t+1), L(t+1))$ minus the reposition costs i.e.,

$$\max_{L(t+1)} E_{D(t+1)}[P(L(t+1), D(t+1))] - \sum_{i=1}^{m}\sum_{j=1}^{k} Rep(t)_i^j(L_i^j(t+1)). \tag{37}$$

Similarly to the Unconstrained Placement Problem without Reposition costs, we assume that the marginal functions of P, i.e., ζ are concave.

As we formulate next that the Unconstrained Placement Problem with Reposition costs can be reduced to the general Unconstrained Placement Problem (without Reposition costs), and can be solved using the transformation to the min-cost flow problem as described in [6].

Theorem 7.1. *The Unconstrained Placement Problem with Reposition costs can be reduced to the Unconstrained Placement Problem without Reposition costs.*

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 profit function should be expressed as we done in our modeling section (Section 2.1). 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 area+type local cost functions, ${C'}_i^j$ includes the repositions costs, i.e., equals to

$$ {C'}_i^j(L_i^j) = C_i^j(L_i^j) + Rep(t)_i^j(L_i^j), \tag{38} $$

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' equals 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 Unconstrained Placement Problem, as described in [6] holds only if the marginal functions are concave (see Section 2.3). That means, we need to show that the marginal profit functions of P', denoted by ζ', are concave. Since only the local costs C_i^j have changed, it is remain to show that only the area+type marginal functions ${\zeta'}_i^j$ are concave. According to Eq. (39), these functions are equal to

$$ {\zeta'}_i^j(L_i^j, D_i^j) = \zeta_i^j(L_i^j, D_i^j) - Rep(t)_i^j(L_i^j). \tag{39} $$

where ζ_i^j are the area+type 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. (36) the differential equals 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 constants 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. (39), the marginal functions of P', ${\zeta'}_i^j(L_i^j, D_i^j)$, are a sum of concave functions, and thus they must be concave.

□

Finally, in [6] we described an algorithm that solves the Unconstrained Placement Problem in $O(|L|km(k+m))$, where $|L|$ is the size of the optimal placement (which is bounded by our storage constant s), k number of regions and m number of resource types m.

7.2. The parameterized Unconstrained problem – Can it be used to solve the constrained reposition problem

The Parameterized Unconstrained Problem is a special case of the Unconstrained Placement Problem with Reposition Costs, whose solutions can possible used to solve the Constrained Reposition Problem for particular reposition constants r. Formally, the Parametrized Unconstrained Problem is defined as the Unconstrained Placement Problem, where every reposition (add or subtract) incurs a cost of $\lambda \geq 0$. Given a reposition cost parameter λ, the Parametrized Unconstrained Problem is to find the optimal placement $L_{par}(\lambda)$ with the larger parametrized profit, i.e., solves the following problem (similar to Eq. (37)):

$$ \max_{L(t+1)} E_{D(t+1)}[P(L(t+1), D(t+1))] - \lambda \cdot \sum_{i=1}^{m}\sum_{j=1}^{k} |L(t+1)_i^j - L(t)_i^j|. \tag{40} $$

The Parametrized Unconstrained Problem, can be solved using the same solution of the Unconstrained Placement Problem with Reposition Costs.

Now suppose that a solution $L_{par}(\lambda)$ has repositioned r_λ resources. Let L be a placement that repositioned at most r_λ resources. Then the profit $P(L, D(t+1))$ of L can not 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. (40)) than $L_{par}(\lambda)$ and $L(\lambda)$ is not an optimal placement for the Parametrized Unconstrained Problem. Thus, we present the following corollary:

Corollary 7.2. *Suppose $L_{par}(\lambda)$ solves the Parametrized Unconstrained Problem with parameter λ, and uses r_λ repositions. Then $L_{par}(\lambda)$ solves the Constrained Reposition Problem under r_λ repositions.*

Note that this corollary does not contradict the hardness of the Constrained Reposition Problem (presented in Section 4); Given a general reposition constant r, it is hard to find a parametrized solution $L_{par}(\lambda)$ which uses exactly $r_\lambda = r$ repositions. However, the corollary may helps us in attempting to find the optimal solution for particular constraint r, by solving the Parameterized Unconstrained 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 can possibly expedite this search by utilizing monotonicity.

This corollary enables to us to measure the effectiveness of every heuristic strategy for solving the Constrained Reposition Problem. Suppose that $L_{par}(\lambda)$ is a parametrized solution that uses r_λ repositions, and let $L_A(\lambda)$ be the solution an heuristic algorithm A finds under r_λ repositions. Using Corollary 7.2 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 problem $L_{par}(\lambda)$. This approach is used in Section 8.4, when we measure the effectiveness of SCO.

7.3. Modeling Extensions of unsatisfied requests

In the appendix (Section Appendix A) we extend our modeling to include in our profit function negative service costs of unsatisfied requests. In our current modeling unsatisfied requests has no impact on the profit function. We show that such generalized profit function incorporating unsatisfied requests can be expressed as a profit function which does not incorporates unsatisfied requests plus a constant depending only on the demand (and not on the placement). Thus, solutions for the Constrained Reposition Problem, where unsatisfied requests has no impacts (such as the SCO solution provided in Section 5) can be used.

8. Performance evaluation of the dynamic algorithms

In this section we evaluate the performance of the dynamic algorithms presented in this article; for the sake of providing a scale of reference we compare them with the Proportional Mean placement - A replication strategy proposed in [5]. We simulate a mobile game app 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([17]).

8.1. Parameter Settings

We refer to our model (Section 2) for a full explanation of our simulation parameters. To achieve good performance the mobile app provider aims at receiving 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 serves offers two different applications ($m = 2$), each requiring different type of platform from the service provider (e.g. either a Windows platform or a Red Hat Enterprise Linux (Linux) platform on the cloud). We assume that both types of cloud platforms can serve up to $B_{Windows} = B_{Linux} = 500$ users.

The revenue constants R_i, R_i^j (see Eq. (6) (7)) are respectively associated with serving locally 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 serving locally a type i user is uniform Europe across regions, and denote this local constant by $R_i^{loc} := R_i^j$, for all platform type $i \in \{Windows, Linux\}$ and region $j \in \{USA, Europe, Asia\}$.

We denote for sake of presentation the other (global) service constant by $R_i^{glo} := R_i$. These revenue constants R_i^{glo}, R_i^{loc} can be defined as the Average Revenue Per User (ARPU) of granting a user request. The ARPU may vary across mobile applications as reported in [18]. The full details assumed for the revenue constants can be found in Table K.1 in Appendix K, where we use values similar to those reported in [18].

Due 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 [17]), i.e., we set the area costs to be $C^{USA}(x) = C^{Eroupe}(x) = C^{Asia}(x) = \infty$ for any $x > 20$.

We set the area & type cost function (See Eq. (6)) to be a linear function $C_i^j = p_i^j \cdot x$ where p_i^j are the prices of renting an on-demand instance of resource i in region j. In Table K.2 in Appendix K we provide the on-demand costs p_i^j. The costs are based on 2014 Amazon EC2 price system on the m3.medium VM instances [17], on Windows and Linux platforms, over three regions: North Carolina (USA), Ireland (Europe) and Singapore (Asia). Note that when using Amazon EC2 servers there are no area or type costs and thus we can set the area and type cost functions to be zero (See Eq. (7), Eq. (8)), i.e., $C^{USA}(x) = C^{Eroupe}(x) = C^{Asia}(x) = 0$ 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 area j on time t, $D_i^j(t)$, to be a time-dependent Poisson distribution with parameter $\lambda_i^j(t)$ (as done in [19] and [20]). Our dynamic scheduling uses an hourly based prediction, where in every area the average number of requests for Windows and Linux servers is the same i.e., $\lambda_{Linux}^j(t) = \lambda_{Windows}^j(t)$. We set the demand parameter to be a periodic function that increases during day time and decreases during night time (usually, between 4 Pm till 4 Am) as done in other studies (such as [20]). Also, in some web applications (such as [4]) the arrival rate is considered to be unpredictable with a higher variability due to some noise factor. In order to simulate such arrival rates, we add an Additive White Gaussian Noise to the arrival rate formula. Thus, the arrival rate is

$$\lambda_i^j(t) = 3000 \cdot \sin(\frac{t_j \cdot \pi}{24}) + 300 \cdot \zeta, \tag{41}$$

where ζ is a white noise, generated by the standard Gaussian (Normal) distribution, and t_j is the local time at area 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 $t_{Asia} = ((t+12) \bmod 24)$.

8.2. 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. (41)) over a time range of $t = 0, 1, \ldots, 47$ and compare the following algorithms: 1) The optimal unconstrained placement.[11] 2) The SCO [12]. algorithm (Section 5) with reposition cost $r = 4$. 4) The Hybrid algorithm (Section 6) with thresholds $\epsilon = \$2000$, $r_{min} = 2$, $r_{max} = 4$. 5) A placement strategy called **Proportional Mean**, as proposed in [5], where the number of type i servers allocated in every area j is proportional to demand, i.e., there exists 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 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 RCC represents the percentage of servers that were changed.

[11] 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 4), the optimal unconstrained placement (reposition constraint set to infinity) is solvable in polynomial time.

[12] The actual algorithm used in the simulation is an approximate implementation of SCO, called SCC (details of SCC are in [6])

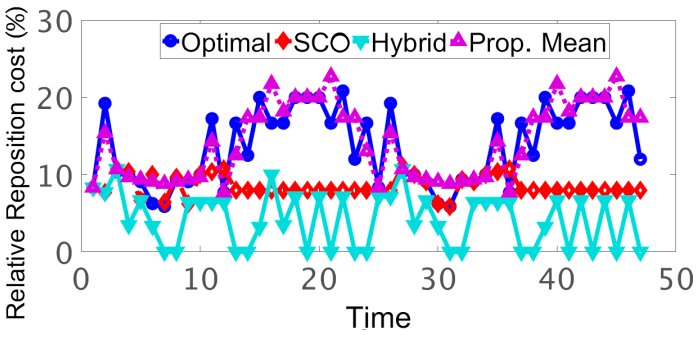

Figure 7: Relative Reposition Cost

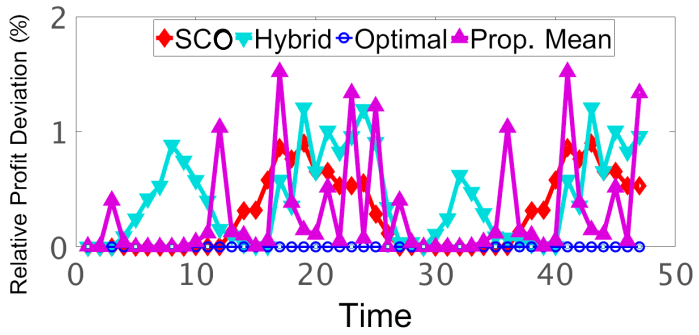

Figure 8: Relative deviation of the placement profit

We observe the following results: 1) The Hybrid algorithm incurs the lowest reposition cost of at most 10%, 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 respectively 15.8% and 16.8%. 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) SCO has (almost) a constant reposition cost, which is always around 10%. Its average reposition cost is 9.8%, which is higher 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 we depict in Figure 8 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[13].

We observe that the Relative Profit Deviation of all the dynamic algorithms studied is at most 2%. Note that the Hybrid placement algorithm achieves very efficient placements, whose relative deviation is bounded by 1.3%, while obeying strict reposition constraints.

For the sake of completeness we depict in Figure 9 the absolute (non relative) profit deviation achieved by the algorithms in these runs. We observe that: 1) SCO has the minimal profit deviation (at most \$130 per hour) with low variability. 2) The Hybrid algorithm is better than Proportional Mean (deviations of Hybrid

[13]Recall that as opposed to the optimal reposition problem under reposition constraint (as studied in this work) that is a hard problem(See Section 4), the optimal unconstrained placement is solvable in polynomial time.

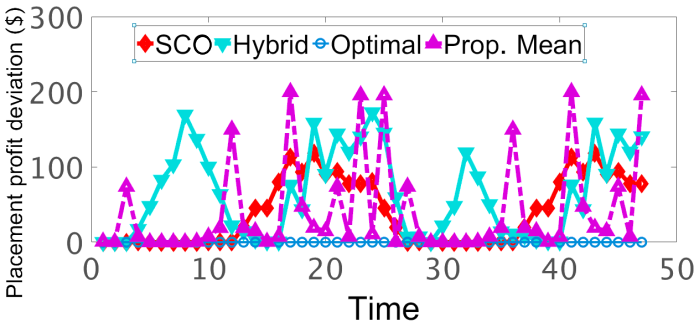

Figure 9: absolute deviation of the placement profit

and Proportional Mean are bounded by \$200 and \$400 respectively). 3) Proportional Mean suffers from high variability compare to the Hybrid algorithm and SCO.

8.3. 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 at most 1.3%, better than Proportional Mean.

Although SCO 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.

8.4. Comparing the performance of SCO to the parametrized unconstrained solution

We use real demand traffic, taken from [3], to numerically evaluate SCO and compare it to the optimal solutions of the Parametrized Unconstrained Placement problem described in Section 7.2. We have shown that if a parametrized solution $L(\lambda)$ uses r repositions then $L(\lambda)$ solves the Constrained Reposition Problem 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 larger than that of a remote request. In the lack of real demand data we use data from [3] 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. .

In Figure. 10 we depict the profit of SCO and compare it to the profit of the parametrized solutions. We run SCO 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 SCO is the optimal strategy, and have similar performance to the parametrized solutions.

We note that the parametrized solutions do not cover all the optimal solutions of the Constrained Reposition 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 significantly larger than those of $L(1)$. Thus, we see the need to used our SCO algorithm that can provide (nearly-optimal) solutions for the Constrained Reposition Problem for any value r .

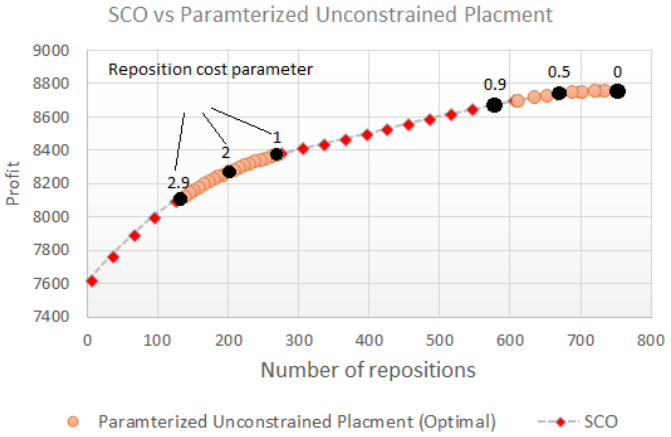

Figure 10: Performance of SCO compared with the parametrized unconstrained solutions. Reposition cost parameters values are indicated above the curve.

9. Related Work

The problem of resource placement in distributed systems often falls under facility location theory [21]. This area has received significant attention from the viewpoint of both analysis and algorithmic solutions. Our version of the problem differs from traditional facility location problems in that it incorporates stochastic demands, and that it focuses on how to reposition the resources in response to changes in the demand.

Early works on distributed resource placement primary focused on placing content replicas across a content distribution or a web cache network (see e.g. [22, 23]). Other more modern works studied resource replication in P2P systems. For example, [5] examined file placement in hierarchically-organized topology; and [24] proposed an optimal movie replication algorithm across peers. However, most of these works have focused on static allocations that account mainly for service costs; while others did consider dynamic settings (e.g., [25]), but payed little attention to resource placement and reconfiguration overhead.

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, [26] focused on dynamically optimizing service placement while ensuring performance requirements; and [27] 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 newly proviosined virtual machine to distributed datacenters [28]. 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 a dynamic algorithm that inherently accounts for the full demand distribution and for repositioning costs, yielding a cost-effective solution better geared for modern distributed settings.

Other policies for server placement look at the long-term horizon given predicted stochastic demand [29, 30], and stochastic load balancing of VMs in datacenters [31]. They use heuristics to compute an efficient (in time) solution, while our work finds an optimal solution in polynomial time. Further, [29, 30] do not capture the reposition cost overhead, and thus allow for large number of repositions, a costly task in practice.

A similar model to ours is also used to optimize server and task scheduling in cloud settings [32],[33]. In these works, every server (or task) execution has a a start and finish time, making these problems harder then placement problems, and resulting either in non optimal heuristic strategies (though with polynomial runtime), or close to optimal solutions but with exponential run-times.

This work solves the problem of dynamic resource placement with dynamically changing stochastic demands under a general and rich cost model. Our related work [15] considered a more limited model of a single-type multi-region setting, which allows for a greedy-like optimal solution. In contrast, this work solves a multi-type multi-region setting, that requires dealing with a complex cost function that accounts for area-specific, resource-type specific, and combined area-type marginal costs (see Eq. (1)). Consequently, we show that the problem is hard, and thus it requires new techniques. Further, unlike previous work, this paper deals with sensitivity analysis of the problem: we demonstrate that reposition problems can be mapped to a min-cost flow problem (in graph theory), yielding a general technique to derive sensitivity bounds on placement profit.

10. Concluding Remarks

We dealt with the problem of dynamic resource placement, accounting for dynamically changing stochastic demands with arbitrary distributions as well as for a very rich cost model. We showed that dynamic demand fluctuations may inflict huge reposition costs and therefor a dynamic placement algorithm must account for them. We showed that, in contrast, small demand deviations result with minor effect on revenues and profits.

We analyzed the constrained reposition problem and provided strong evidence that under the wide setting of this work the problem at large is hard. We therefore provided a heuristic algorithm, called SCO, that is guaranteed to find the optimal solution in several important special cases. For a multi-period online setting we proposed Hybrid, an algorithm that avoids repositioning when the demand changes are small and conducts SCO when they are large. Simulation results show that Hybrid achieves very good placements while avoiding large repositions. The subject of how to deal with a multi-period problem where all future demands are known in advance is the subject of ongoing research.

Acknowledgment

We would like to thank Noga Alon for helpful discussions on the hardness of the discussed problems. This research was supported by grants from the Israeli ministry of Science and from the Blavatnik Fund. We thank anonymous referees for their helpful comments and proposing the reposition-cost model.

References

[1] Amazon EC2 home page. http://aws.amazon.com/ec2 (2013).
[2] Microsoft Azure home page. http://www.windowsazure.com (2013).
[3] I. Narayanan, A. Kansal, A. Sivasubramaniam, B. Urgaonkar, S. Govindan, Towards a leaner geo-distributed cloud infrastructure, in: Proceedings of HotCloud, 2014.
[4] V. Cardellini, E. Casalicchio, F. L. Presti, L. Silvestri, Sla-aware resource management for application service providers in the cloud., in: NCCA, 2011, pp. 20–27.
URL http://dblp.uni-trier.de/db/conf/ncca/ncca2011.html#CardelliniCPS11
[5] S. Tewari, L. Kleinrock, Proportional replication in peer-to-peer networks, in: IEEE INFOCOM, Barcelona, Spain, 2006.
[6] Y. Rochman, H. Levy, E. Brosh, Dynamic placement of resources in cloud computing and network applications - technical report, https://drive.google.com/folderview?id=0B5v_NLuOQ1TAOVBQd3pTXzFXMlU&usp=sharing (2014).
[7] E. Weisstein, Kolmogorov-smirnov test, http://mathworld.wolfram.com/Kolmogorov-SmirnovTest.html.
[8] A. Klenke, J. G. universitat Mainz, L. Mattner, Stochastic ordering of classical discrete distributions (2010).
[9] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, 1993.
[10] Y. Rochman, H. Levy, On sensitivity of dynamic min cost flows - technical report, https://goo.gl/SWsQ8V, to be submitted.
[11] Y. Rochman, H. Levy, 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, 2012.
[12] C. H. Papadimitriou, M. Yannakakis, The complexity of restricted spanning tree problems, J. ACM 29 (2) (1982) 285–309.

[13] Y. Raphael, Almost exact matchings, Algorithmica 63 (1) (2012) 39–50.
[14] Exact matching in red-blue bipartite graphs- egervry research group on combinatorial optimization (egres). (2016).
URL http://lemon.cs.elte.hu/egres/open/Exact_matching_in_red-blue_bipartite_graphs
[15] Y. Rochman, H. Levy, E. Brosh, G. Gilboa-Freedman, Resource repositioning in distributed clouds - technical report, https://goo.gl/Tjntfw, to be Submitted.
[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Second Edition, The MIT Press and McGraw-Hill Book Company, 2001.
[17] Amazon EC2 pricing page, http://aws.amazon.com/ec2/pricing/ (2014).
[18] Think Gaming: top paid games, http://thinkgaming.com/app-sales-data/top-paid-games/ (2014).
[19] X. Nan, Y. He, L. Guan, Optimal allocation of virtual machines for cloud-based multimedia applications., in: MMSP, IEEE, 2012, pp. 175–180.
[20] B. Zhang, G. Kreitz, M. Isaksson, J. Ubillos, G. Urdaneta, J. A. Pouwelse, D. H. J. Epema, Understanding user behavior in spotify., in: INFOCOM, IEEE, 2013, pp. 220–224.
[21] Z. Drezner, H. W. Hamacher, Facility Location: Applications and Theory, Springer, 2002.
[22] L. Qiu, V. N. Padmanabhan, G. M. Voelker, On the placement of web server replicas, in: IEEE INFOCOM, Anchorage, AK, USA, 2001.
[23] F. L. Presti, C. Petrioli, C. Vicari, Distributed dynamic replica placement and request redirection in content delivery networks, in: MASCOTS, 2007, pp. 366–373.
[24] Y. P. Zhou, T. Z. J. Fu, D. M. Chiu, Statistical modeling and analysis of p2p replication to support vod service, in: IEEE INFOCOM, Orlando, FL , USA, 2011.
[25] Y. Chen, R. H. Katz, J. D. Kubiatowicz, Dynamic replica placement for scalable content delivery, in: International Workshop on Peer-to-Peer Systems, Springer, 2002, pp. 306–318.
[26] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, J. L. Hellerstein, Dynamic service placement in geographically distributed clouds, IEEE Journal on Selected Areas in Communications 99 (2013) 762–772.
[27] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, F. C. Lau, Scaling Social Media Applications into Geo-Distributed Clouds, in: IEEE INFOCOM, Orlando, Florida, USA, 2012.
[28] H. Xu, B. Li, Joint Request Mapping and Response Routing for Geo-distributed Cloud Services,, in: IEEE INFOCOM, Turin, Italy, 2013.
[29] S. Chaisiri, B. Lee, D. Niyato, Optimal virtual machine placement across multiple cloud providers, in: APSCC, 2009.
[30] S. Chaisiri, R. Kaewpuang, B. Lee, D. Niyato, Cost minimization for provisioning virtual servers in amazon elastic compute cloud, in: MASCOTS, 2011.
[31] L. Yu, L. Chen, Z. Cai, H. Shen, Y. Liang, Y. Pan, Stochastic load balancing for virtual resource management in datacenters, in: IEEE Transactions On Cloud Computing, 2016.
[32] S. T. Maguluri, R. Srikant, L. 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, 2012, pp. 702–710.
[33] D. Hatzopoulos, I. Koutsopoulos, G. Koutitas, W. Van Heddeghem, Dynamic virtual machine allocation in cloud server facility systems with renewable energy sources, in: Communications (ICC), 2013 IEEE International Conference on, IEEE, 2013, pp. 4217–4221.
[34] R. G. Busacker, 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).

Appendix A. Model Extensions and Related Problems – Unsatisfied Requests

Our general model and problems can be extend to account for special costs (negative profits) attributed to unsatisfied requests. We show that this problem can be reduced to the Constrained Reposition Problem where unsatisfied requests do not incur cost, and use solutions such as SCO (described in Section 5) to solve the problem.

Suppose that the (negative) profit of every type-i unsatisfied request in region j is $U_i^j < 0$. As explained in the modeling section (Section 2.1), the number of type-i requests satisfied by some resource in region j equals 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 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 marginal local area+type function $\zeta_i^j(L_i^j, D_i^j)$. According to Eq. (6), this equal to:

$$\zeta_i^j(L_i^j, D_i^j) = R_i^j \cdot E_{D_i^j}[\min(L_i^j \cdot B_i, D_i^j)] - C_i^j(L_i^j) + U_i^j \cdot E_{D_i^j}[D_i^j - \min(L_i^j \cdot B_i, D_i^j)], \tag{A.1}$$

or alternatively,

$$\zeta_i^j(L_i^j, D_i^j) = (R_i^j - U_i^j) \cdot E_{D_i^j}[\min(L_i^j \cdot B_i, D_i^j)] - C_i^j(L_i^j) + U_i^j \cdot E_{D_i^j}[D_i^j]. \tag{A.2}$$

Note that $U_i^j \cdot E_{D_i^j}[D_i^j]$ is a constant that does not depend on the placement L, and $R_i^j - U_i^j > R_i^j \geq 0$ is the (positive) profit of every type-i region-j request 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 requests costs and 2) An optimal solution for the Constrained Reposition Problem with profit function P is the same optimal solution for the Constrained Reposition Problem with profit function P'. In other words, we show that Constrained Reposition Problem where the unsatisfied requests incur costs, can be reduced to the Constrained Reposition Problem 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 area+type local revenue parameter of P' equals to $R'^j_i = R_i^j - U_i^j > 0$. One can verify that the profit P equals 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 solution of the Constrained Reposition Problem, with a profit function P (where unsatisfied requests have impacts) has the same solution as the Constrained Reposition Problem with profit function P' (where unsatisfied requests do not incur costs).

Appendix B. Min-Cost flow preliminaries and the reduction of resource placements into flows in a 4-layer min-cost flow setting

Appendix B.1. Introduction to the min-cost flow problem

We start describing the **min-cost flow problem** [34], which is a generalization of the notable max flow problem (see [16]), where we considera directed graph $G = (V, E)$ where every edge $e \in E$ has integer **capacity** $c(e)$. Let x, y be two nodes in G, where x is called a source and y a sink. Every flow f defined over the graph edges must obey two properties: 1) for every edge e we have $0 \leq f(e) \leq c(e)$, and 2) conservation of flow, i.e., for every vertex $v \neq x, y$, if $e_1 = (v, u_1), \ldots e_n = (v, u_n)$ are edges outcoming from v, and

$e'_1 = (u'_1, v), \dots e'_{n'} = (u_{n'}, v)$ are edges incoming to v, then $f(v) = \sum_{i=0}^{n} f(e) = \sum_{i=1}^{n'} f(e'_i)$ [14]. Given a flow f, we define $f(v)$ as the **flow in node**. The **flow value** $|f|$ of f is the flow in the source x (and sink y), $|f| = \sum_{(x,v)\in E} f(x,v) = \sum_{(v,y)\in E} f(v,y)$. In addition, every edge is associates with a real-value **weight** $w(e)$ (alternatively, called **cost**). The **weight (or cost)** of a flow f with respect to weight w is $W(f,w) = \sum_{e\in E} f(e)w(e)$.

The classic **min-cost flow** problem with required flow $|f| = k$ is to find a flow $f_{opt}(k)$ of value k that has lowest weight among all flows of value k. This means that for every flow f' such that $|f'| = |f_{opt}(k)| = k$ we have $W(f_{opt}(k), w) \leq W(f', w)$. It is well-known that if the capacities $c(e)$ of a min-cost flow network are integers, then its min-cost flow f has an integer value flow (that is the flow in every edge is integer); a proof can be found in [9], Theorem 9.10.

In this article we generalized the min-cost flow problem, so that the input graph $G = (V, E)$ can be a multigraph, i.e., there are multiple edges between every two vertices in G.

Appendix B.2. The 4-layer graph G^4

To devise the construction of G^4 (Theorem 3.6 and Figure 3) we recall that the profit function $P(D, L)$ is composed of the sum of marginal 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. (5). That means

$$P(L,D) = \underbrace{\sum_{i=1}^{m}\sum_{j=1}^{k}\zeta_i^j(L_i^j, D)}_{\text{area \& type}} + \underbrace{\sum_{i=1}^{m}\zeta_i(L_i, D)}_{\text{type}} + \underbrace{\sum_{j=1}^{k}\zeta^j(L^j)}_{\text{area}}. \tag{B.1}$$

The marginal functions ζ_i^j, ζ_i depend on the demand D, while ζ^j does not depend. We denote the **differential** of a discrete function ζ as $\Delta\zeta(n, D) = \zeta(n+1, D) - \zeta(n, D)$. Also, we recall that the size of every placement is bounded by the storage constant s, which can be as large as one wishes.

The 4-layer multigraph G^4 (graph with parallel edges between vertices) is composed of a source x, region nodes $j_1, j_2, \dots j_k$, resource type nodes $i_1, i_2 \dots i_m$ 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., $(-1)\cdot\Delta\zeta^j(0), (-1)\cdot\Delta\zeta^j(1), \dots, (-1)\cdot\Delta\zeta^j(s-1)$, and the edges are denoted respectively by $j^1, j^2, \dots, j^s$ or by $(x,j)^1, (x,j)^2, \dots, (x,j)^s$. The weight of edges between region nodes j and resource type node i are minus the area-j type i marginal function differential $(-1), (-1)\cdot\Delta\zeta_i^j(0, D), (-1)\cdot\Delta\zeta_i^j(1, D), \dots, (-1)\cdot\Delta\zeta_i^j(s-1, D)$ denoted by $(j,i)^1, (j,i)^2, \dots, (j,i)^s$. The weight of edges between resource type node i and the sink y of 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 B.11 we depict the 4-layer multigraph. We omit some edges for the sake of presentation.

Appendix B.3. The corresponding flow

For every placement L we define its **corresponding flow** f_L in the 4-layer graph G^4 in the following way: 1) For every region j and resource type i we send through edges $(j,i)^1, (j,i)^2, \dots (j,i)^{L_i^j}$ one unit of flow. 2) For every region j we send through edges $j^1, j^2, \dots j^{L^j}$ one unit of flow. 3) For every resource type i we send through edges $i^1, i^2, \dots i^{L_i}$ one unit of flow. 4) The flow between x and y equals to $s - \sum_j L^j$. Note the required flow f_L equals to the storage constant s. An example can be seen in Section 3, Figure 3.

[14] Note that the min-cost flow problem is sometimes defined differently with *mass balance constraints* as done in [9]

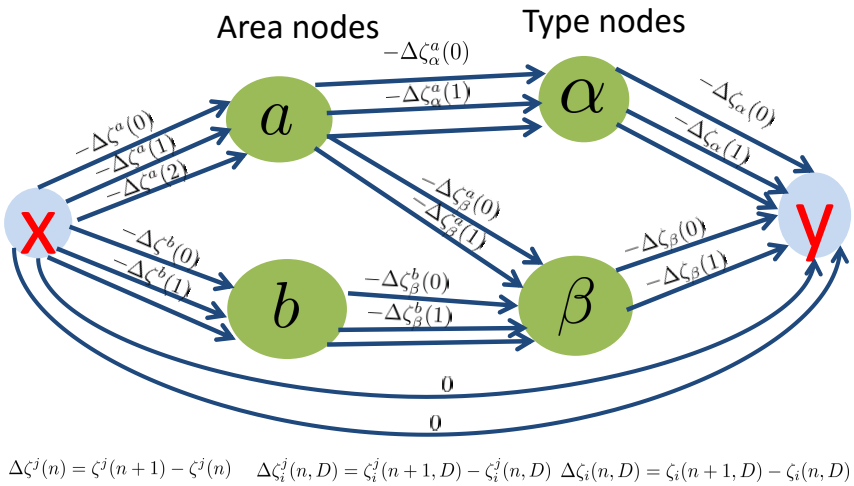

Figure B.11: The 4-layer multigraph G^4. We omit some dges from the graph for the sake of presentation.

Appendix C. Proof of Claim 3.7 from Section 3

Claim 3.7. *Let L be a placement. Then its profit equals to a constant c minus the weight of its corresponding flow f_L i.e.,*

$$P(D,L) = c - W(f_L, w(D)), \tag{C.1}$$

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.

Proof of Claim 3.7. The proof follows from the fact that the sum of differential weights $(-1)\cdot\Delta\zeta^j(0,D), (-1)\cdot\Delta\zeta^j(1,D), \ldots, (-1)\cdot\Delta\zeta^j(L^j-1)$ forms a telescoping series. The formal proof is presented below.

Let j be a region node, and denote the flow weight between x and j as $w_f(x,j) = \sum_{e \text{ is an edge between } x \text{ and } j} f(e)w(e)$. Since f_L sends a single unit of flow through edges $j^1, j^2, \ldots j^{L^j}$ one unit of flow with differential weights $(-1)\cdot\Delta\zeta^j(0), (-1)\cdot\Delta\zeta^j(1), \ldots, (-1)\cdot\Delta\zeta^j(L^j-1)$ then

$$w(x,j) = \sum_{n=0}^{L_i^j-1} (-1)\cdot\Delta\zeta^j(n) \underbrace{=}_{\text{defintion of } \Delta g} \sum_{n=0}^{L^j-1} (\zeta^j(n) - \zeta^j(n+1)) \underbrace{=}_{\text{telescoping series}} \zeta^j(0) - \zeta^j(L^j),$$

and therefore,

$$\zeta^j(L^j) = \zeta^j(0) - w(x,j). \tag{C.2}$$

Note that according to Eq. (8) $\zeta^j(0) = -C^j(0)$ is constant, and depends neither on the placement L nor on the demand D.

We can similarly define for every region node j and resource type node i 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 imply that

$$\zeta_i^j(L_i^j, D) = \zeta_i^j(0,D) - w_f(j,i), \tag{C.3}$$
$$\zeta_i(L_i, D) = \zeta_i(0,D) - w_f(i,y),, \tag{C.4}$$

and according to Eqs. (7)),(6)), $\zeta_i(0,D) = -C_i(0), \zeta_i^j(0,D) = -C_i^j(0)$ are constants that do not depends on the placement L nor the demand D.

The weight of f_L equals 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. Also, according to Eq. B.1 the profit $P(D,L)$ equals 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. (C.2),(C.3),(C.4) over all region nodes j and resource type nodes i will result in the required result.

□

Appendix D. Proof of Claim 3.8 from Section 3

Claim 3.8. *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 unconstrained placement for demand D.*

Proof of Claim 3.8. The correctness of the Claim stems from the follows three facts: 1) the marginal functions g are concave functions (See Section 2.3). I.e., Δg are monotonically non-increasing functions 2) if the capacities of a min-cost flow network are integer, then its min-cost flow f has integer flow (that is the flow on every edge is integer). The latter is a well-known theorem (the Integrality Property in [9], Theorem 9.10). 3) Claim 3.7 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 a single unit capacity $c(e) = 1$. Thus, the flow f_{min} in each edge can be either or 0 or 1.

Let j be a region node and i be a resource type node. We denote the n-indexed edges $j^n, (j,i)^n, i^n$ as respectively the edge between x and j, the edge between j and i, and the edge between i and 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 convex 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 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) one 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 to $s - \sum_{j=1}^k L^j$. 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 $f_{L'}$. By the definition of a min-cost flow, the weight of $f_{L_{opt}}$ is smaller than the flow of $f_{L'}$. By Claim 3.7, we imply that L_{opt} has larger profit than L', thus L_{opt} is the optimal unconstrained placement. □

Appendix E. Proof of Claim 3.10 from Section 3

Claim 3.10. *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.,*

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

Proof of Claim 3.10. 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:

$$E_X(\min(n, X)) = \sum_{k=1}^{n} \Pr(X \geq k). \tag{E.2}$$

The correctness of the equation is shown in [11] Claim 6.4.

Using Eq E.2 we can now compute the weights of G^4 edges. In Section 2, we expressed the marginal profit functions by the model parameters, which equals to

$$\begin{aligned} \zeta_i^j(L_i^j, D_i^j) &= R_i^j \cdot E_{D_i^j}[\min(L_i^j \cdot B_i, D_i^j)] - C_i^j(L_i^j) \\ \zeta_i(L_i, D_i) &= R_i \cdot E_{D_i}[\min(L_i \cdot B_i, D_i)] - C_i(L_i^j) \\ \zeta^j(L^j) &= -C^j(L^j). \end{aligned} \tag{E.3}$$

Using Eqs. (E.3), (E.2) we observe that the edge weights of the n^{th} edge between respectively 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

$$-\Delta\zeta_i^j(n-1, D) = \Delta C_i^j(n-1) - R_i^j \cdot \sum_{k=(n-1)\cdot B_i+1}^{n\cdot B_i} \Pr(D_i^j \geq k), \tag{E.4}$$

$$-\Delta\zeta_i(n-1, D) = \Delta C_i(n-1) - R_i \cdot \sum_{k=(n-1)\cdot B_i+1}^{n\cdot B_i} \Pr(D_i \geq k) \tag{E.5}$$

$$-\Delta\zeta^j(n-1) = \Delta C^j(n-1). \tag{E.6}$$

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 equals to

$$|w(e) - w'(e)| = \begin{cases} R_i^j \cdot |\sum_{k=(n-1)\cdot B_i+1}^{n\cdot B_i} \Pr(D_i^j(t) \geq k) - \Pr(D_i^j(t+1) \geq k)| & e = (i,j)^n, \\ R_i \cdot |\sum_{k=(n-1)\cdot B_i+1}^{n\cdot B_i} \Pr(D_i(t) \geq k) - \Pr(D_i(t+1) \geq k)| & e = i^n, \\ 0 & \text{Otherwise.} \end{cases} \tag{E.7}$$

Let j be a region node and i a resource type node. Then summing the weight difference over all edges $(j,i)^n$ is

$$\begin{aligned} \sum_{n=1}^{\infty} |w((i,j)^n) - w'((i,j)^n)| &\underbrace{\leq}_{\text{using triangular inequality}} R_i^j \cdot \sum_{k=1}^{\infty} |\Pr(D_i^j(t) \geq k) - \Pr(D_i^j(t+1) \geq k)| \\ &\underbrace{=}_{\Pr(X\geq n)=1-\Pr(X\leq n-1)} R_i^j \cdot \sum_{k=0}^{\infty} |\Pr(D_i^j(t) \leq k) - \Pr(D_i^j(t+1) \leq k)| = R_i^j \cdot d(D_i^j(t), D_i^j(t+1)). \end{aligned} \tag{E.8}$$

Similarly, the sum of weight difference over all edges i^n is

$$\begin{aligned} \sum_{n=1}^{\infty} |w(i^n) - w'(i^n)| &\underbrace{\leq}_{\text{using triangular inequality}} R_i \cdot \sum_{k=1}^{\infty} |\Pr(D_i(t) \geq k) - \Pr(D_i(t+1) \geq k)| \\ &\underbrace{=}_{\Pr(X\geq n)=1-\Pr(X\leq n-1)} R_i \cdot \sum_{k=0}^{\infty} |\Pr(D_i(t) \leq k) - \Pr(D_i(t) \leq k)| = R_i \cdot d(D_i(t), D_i(t+1)). \end{aligned} \tag{E.9}$$

If we sum up Eqs. (E.8), (E.9) over all regions j and resource types i then we have

$$\sum_{e\in E}|w(e)-w'(e)| \underbrace{=}_{Eq.\ (E.7)} \sum_{i=1}^{m}\sum_{n=1}^{s}|w(i^n)-w'(i^n)| + \sum_{j=1}^{k}\sum_{i=1}^{m}\sum_{n=1}^{s}|w((j,i)^n)-w'((j,i)^n)|$$

$$\underbrace{\leq}_{\text{Eqs. (E.8), (E.9)}} \sum_{i=1}^{m} R_i \cdot d(D_i(t), D_i(t+1)) + \sum_{j=1}^{k}\sum_{i=1}^{m} R_i^j \cdot d(D_i^j(t), D_i^j(t+1)) \underbrace{=}_{\text{Definition of demand distance}} d(D(t), D(t+1)), \quad \text{(E.10)}$$

as required. □

Appendix F. Proof of Theorem 4.4 (Hardness of the Fair Christmas Game Problem)

In this section we prove Theorem 4.4 from Section 4, using the Exact Cycle Sum problem.

Definition 7. *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 [12] 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 4.4 from Section 4.

Theorem 4.4. *There is a polynomial reduction from the Cycle Sum Problem to the Fair Christmas Game Problem.*

Proof of Theorem 4.4. 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 $T=\{C_1,C_2,\ldots C_n\}$. 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** $C^{game}=(u^1,u^2,\ldots u^m,u^1)$ 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 $C_i=(v^1,v^2,\ldots,v^m,v^1)$ with m edges is corresponding to a game cycle with $2m$ gifts of the form $C^{game}=(v^1_{in},v^1_{out},v^2_{in},v^2_{out},\ldots,v^m_{in},v^m_{out},v^1_{in})$. 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} to be 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 $C^{game}=(v^1_{in},v^1_{out},v^2_{in},v^2_{out},\ldots,v^m_{in},v^m_{out},v^1_{in})$ where $2m$ gifts are sent and it contains player v^{in}. Such game cycle is corresponding to a cycle $C=(v^1.v^2,\ldots)$ in G, with m edges. Thus,

$C_1^{game}, C_2^{game}, \dots C_n^{game}$ 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 $C_1, C_2, \dots C_n$ have disjoint vertices, since by the construction the game cycles are unique to every player. □

Appendix G. Full version of Claim 5.5, and its proof

Lemma Appendix G.1 (Extended version of Lemma 5.5)**.** *Let o be an extended chain operation $O(j_1 \to j_2 \to \dots \to j_n)$. Suppose that between j_k to j_{k+1} it moves a resource of type i_k ($k = 1, 2 \dots n-1$). 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 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,

$$\Delta_i(j_1 \to j_2) = \Delta g_i^{j_2}(L_i^{j_2}) - \Delta g_i^{j_1}(L_i^{j_1} - 1) \tag{G.1}$$
$$\Delta^+(j) = -\Delta g^j(L^j - 1) \tag{G.2}$$
$$\Delta_i^+(j) = \Delta g_i^j(L_i^j) + \Delta g_i(L_i) \tag{G.3}$$
$$\Delta^-(j) = \Delta g^j(L^j) \tag{G.4}$$
$$\Delta_i^-(j) = -\Delta g_i^j(L_i^j - 1) - \Delta g_i(L_i - 1). \tag{G.5}$$

Proof. Let $\hat{L} = o(L)$. Then by the definition of profitability and Eq. (29), the profitability of o over L equals 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 original lemma, (Lemma 5.5). If the operation adds a type i resources 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 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, in all the above classes the operation 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 type i_0 resource from region j_n to region j_1. Thus, in this case, the operation contributes $\Delta_{i_0}(j_n \to j_1)$.

For the first and third class operations o does not add a resource to j_1 and thus the number of resource in j_1 is reduced, i.e., the operation o contributes to the profitability $\Delta^+(j_1) = -\Delta g^{j_1}(L^{j_1} - 1)$. For operations of the second, fourth and fifth classes the number of resources in region j_1 does not change. These operations 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 the fifth class operations this term appears as part of $\Delta_{i_0}(j_n \to j_1) = \Delta g_{i_0}^{j_1}(L_{i_0}^{j_1}) - \Delta g_{i_0}^{j_n}(L_{i_0}^{j_n} - 1)$). In operation of the second and fourth classes 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 $(i, j) \neq (i_k, j_k), (i_k, j_{k-1})$, and they do not contribute to the profitability. Also 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 changed, and they do not contribute to the profitability. □

Appendix H. 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 5.5. 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: at first stage they create a graph G, and in the second stage they run a variation of shortest path algorithm to find the shortest profitable operation.

Given the placement L, in the first stage all the algorithms create a complete digraph of region vertices $1, 2 \ldots, k$. 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. (31)).

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 (for the first and third class operations) are $w^+(j) = (-1) \cdot \Delta^+(j)$. Algorithms A_2 and A_4 are finding 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 are finding 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 all algorithms use the Bellman-Ford algorithm [16], which computes in the i^{th} iteration the shortest path between two vertices s and t that uses at most i edges. 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 finds in the i^{th} iteration the shortest path between $s = j$ to itself ($t = j$) that uses at most i edges. 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 $O(j_1 \to j_2, \ldots \to j_{i_{min}-2})$ that 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 algorithm must find the shortest profitable operation in their respected classes:

Claim Appendix H.1. *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 5.6.

□

The complexity of $A_1 \ldots, A4$ 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 at most $O(VE)$. 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 equals to $O(k^4)$ and in both stages equals to $O(k^2(k^2 + m))$, which is the time complexity of our full algorithm.

Appendix I. Description of residual graphs and the Augmenting Cycle Theorem for multi-graphs

In Theorem 5.4 we use properties over the **residual graph**, which is defined next:

Definition 8. *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 to G_f edges from G, such that edge $e \in E$ will have weight $w_f(e) = w(e)$ and 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 weight $w_f(v', v) = -w(v, v')$ and 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 residual capacity $c_f(e) = 0$ are removed from the graph.*

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 residual weight function w_f and new capacity c_f. We define the residual weight w_f of cycle C 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 Appendix I.1 (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 there exist cycles $C_1, C_2, \ldots C_m$ in the residual graph G_f such that 1) f' equals 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', equals to the weight of f plus the residual weight of the flowing cycles C_i i.e.,*

$$W(f', w) = W(f, w) + \sum_{i=1}^{m} w_f(C_i)\Delta_i \tag{I.1}$$

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 [9] [15]. □

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 4-layer multigraph G^4. However, we extend that theorem and show that the theorem applies to a multigraph as well.

Theorem Appendix I.2 (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 there exists cycles $C_1, C_2, \ldots C_m$ in the residual graph G_f such that 1) f' equals to*

[15] Note that [9] is using slightly different version than ours.

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', equals to the weight of f plus the residual weight of the flowing cycles C_i i.e.,

$$W(f', w) = W(f, w) + \sum_{i=1}^{m} w_f(C_i) c_f(C_i). \tag{I.2}$$

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) will be equal to the weight of the edge e in G, while the weight of the edge (v_e, v_2) to be set to zero. The capacity of both edges $(v_1, v_e), (v_e, v_1)$ in $\hat{G}$ equals to the capacity of the edge e.

Every flow in the multigraph G has a corresponding flow in G, and vice versa. Simply, if a flow f in G goes through edge e, then the appropriate flow $\hat{f}$ in $\hat{G}$ will go through edges (v_1, v_e) and (v_e, v_2) in $\hat{G}$. The weights of both flows equal to each other ($W(f, w) = W(\hat{f}, \hat{w})$). 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 flow $\hat{f}$ and $\hat{f}'$ in G'. Then, according 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'. □

Appendix J. Proof of Lemma 5.7 from Section 5

Lemma 5.7. *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 is corresponding 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 5.7. To prove Lemma 5.7 we use Observation Appendix J.1 (given below) that 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 Appendix J.2), and then prove that $o^*(o, L)$ is an extended chain operation (Claim Appendix J.3).

To define $o^*(o, L)$ we use the following observation that characterizes the edges of the residual operation multigraph $G(o, L)$.

Observation Appendix J.1. *The residual operation multigraph $G(o, L)$ should contain only the following edges from the 4-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 $L_i^j < o(L)_i^j$ then $G(o,L)$ contains original edges $(j,i)^{L_i^j+1},(j,i)^{L_i^j+2},\ldots,(j,i)^{o(L)_i^j}$ of the residual graph $G^4_{f_L}$ i.e., with respective weights $-\Delta\zeta_i^j(L_i^j,D),-\Delta\zeta_i^j(L_i^j+1,D),\ldots,-\Delta_i^j(o(L)_i^j-1,D)$. If $L_i^j > o(L)_i^j$ then $G(o,L)$ contains reverse edges $(i,j)^{L_i^j},(i,j)^{L_i^j-1},\ldots,(i,j)^{o(L)_i^j+1}$ of $G^4_{f_L}$ with respective weights $\Delta\zeta_i^j(L_i^j-1,D),\Delta\zeta_i^j(L_i^j-2,D),\ldots,\Delta\zeta_i^j(o(L)_i^j,D)$. If $L_i^j = o(L)_i^j$ 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 $-\Delta\zeta_i(L_i^j,D),-\Delta\zeta_i(L_i+1,D),\ldots,-\Delta_i(o(L)_i,D)$ 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 $\Delta\zeta_i(L_i-1,D),\Delta\zeta_i(L_i^j-2,D),\ldots,\Delta\zeta_i(o(L)_i,D)$. 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 $(y,x)^{L_i^j},(y,x)^{L_i^j-1},\ldots,(y,x)^{o(L)_i^j+1}$ 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 Appendix J.1. Suppose that we send a single 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 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 Appendix I.2) we get that $G' = G(o,L)$.

Let a be a region node. According to its definition, the corresponding flow of L augments one 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 one unit of flow through G', i.e., through original edges $(x,a)^{L^a+1},(x,a)^{L^a+2},\ldots,(x,a)$ will change the flow f_L so that edges $(x,a)^1$, $(x,a)^2$, …, $(x,a)^{o(L)^a}$ will contain one unit of flow, similar to the corresponding flow of $o(L)$.
2. If $L^a > o(L)^a$ then augmenting one unit of flow through G', i.e., through reverse edges $(a,x)^{L^a},(a,x)^{L^a-1},\ldots,(x,a)^{o(L}$ will cancel the flow in edges $(x,a)^{L^a},(x,a)^{L^a-1},\ldots,(x,j)^{o(L)^a+1}$. That means the flow changes so it contains one 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 one unit of flow through G', does not change the flow of edges between source x and a, and the flow still contains one 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 Appendix J.1 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 contains the edges between i and j, then $o^*(o,L)$ will not change the number of type i resources from region j.

Now we will show that there is a lowest weight cycle C' that represents $o^*(o,L)$ by the following claim:

Claim Appendix J.2. *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 Appendix J.2. 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 between vertices j and i, denoted by $j \to i$. According to Observation Appendix J.1, 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 edges of the residual operation multigraph of $o^*(o,L)$, $G(o^*(o,L),L)$ according to Observation Appendix J.1, for the original edges.

If C contains a reverse edge $i \to j$ then according to Observation Appendix J.1, 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 *tox*, $y \to i$ 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 Appendix J.1 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 $o^*(o,L)(L)$.

□

Now in order to complete Lemma 5.7 we show the following claim:

Claim Appendix J.3. *The operation $o^*(o,L)$ is an extended chain operation, or a composition of two disjoint extended chains.*

Proof of Claim Appendix J.3. 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 cycle (According to Observation Appendix J.1), 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 edge between 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 between 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 uses 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 from j_1.
2. **Second class cycle :** A cycle that includes the source x, the sink y and goes through 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., $E(j_n \to j_{n-1} \to \ldots j_1)$), adds a resource of type i_n to j_n but does removes a resource from j_1.
3. **Third class cycle:** A cycle that includes the source x, the sink y, and goes through the original edge (x, y) is of the format $j_1 \to i_1 \to j_2 \to i_2, \ldots, \to j_n \to x \to y \to i_0 \to j_1$. 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., $E(j_n \to j_{n-1} \to \ldots j_1)$), does not add a resource to j_n but removes a resource of type i_0 from j_1.
4. **Forth class cycle:** A cycle that does not includes x, but includes the sink y is of the format $j_1 \to i_1 \to j_2 \to i_2, \ldots, \to j_n \to i_n \to y \to i_0 \to j_1$. 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., $E(j_n \to j_{n-1} \to \ldots j_1)$), 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 $j_1 \to i_1 \to j_2 \to i_2, \ldots, \to j_n \to i_n \to j_1$. 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., $E(j_n \to j_{n-1} \to \ldots j_1)$), 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 from two disjoint paths: One path from x to y i.e., $P_1 = x \to j_1 \to i_1 \to j_2 \to i_2, \ldots, \to j_n \to i_n \to y$ and the other path from y to x, i.e., $P_2 = y \to i'_0 \to j'_1 \ldots, \to j'_{n'} \to x$. The resource type nodes i'_0 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 does not use the sink y or the source x, and thus of the above format.
   If C' is a sixth class cycle, then $o^*(o, L)$ is composed from 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., $E(j_n \to j_{n-1} \to \ldots j_1)$), 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., $E(j'_n \to j'_{n-1} \to \ldots j'_1)$), 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 5.7 is completed.

□

Appendix K. Cost details for Performance Evaluation

Revenue constants parameters	R_i^{loc}	R_i^{glo}
Windows	$ 0.5	$ 2
Linux	$ 0.1	$ 1.9

Table K.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 K.2: Amazon EC2 price system