12-2019

Server Assignment with Time-Varying Workloads in Mobile Edge Computing

Quynh Vo

SERVER ASSIGNMENT WITH TIME-VARYING WORKLOADS IN MOBILE EDGE COMPUTING

A Dissertation Presented

by

QUYNH VO

Submitted to the Office of Graduate Studies, University of Massachusetts Boston, in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2019

Computer Science Program

SERVER ASSIGNMENT WITH TIME-VARYING WORKLOADS IN MOBILE EDGE COMPUTING

A Dissertation Presented

by

QUYNH VO

Approved as to style and content by:

Duc A. Tran, Associate Professor
Chairperson of Committee

Dan Simovici, Professor
Member

Bo Sheng, Associate Professor
Member

Honggang Zhang, Associate Professor
Member

Dan Simovici, Program Director
Computer Science Program

Marc Pomplun, Chairperson
Computer Science Department

ABSTRACT

SERVER ASSIGNMENT WITH TIME-VARYING WORKLOADS IN MOBILE EDGE COMPUTING

December 2019

QUYNH VO,
B.S., Hanoi University of Science and Technology
Ph.D., University of Massachusetts Boston

Directed by Associate Professor Duc A. Tran

Mobile Edge Computing (MEC) has emerged as a viable technology for mobile operators to push computing resources closer to the users so that requests can be served locally without long-haul crossing of the network core, thus improving network efficiency and user experience. In MEC, commodity servers are deployed in the edge to form a distributed network of mini datacenters. A consequential task is to partition the user cells into groups, each to be served by an edge server, to maximize the offloading to the edge. The conventional setting for this problem in the literature is: (1) assume that the interaction workload between two cells has a known interaction rate, (2) compute a partition optimized for these rates, for example, by solving a weighted-graph partitioning problem, and (3) for a time-varying workload, incrementally re-compute the partition when the interaction rates change. This setting is suitable only for infrequently changing workloads. The operational and computation costs of the partition update can be expensive and it is difficult to estimate interaction rates if they are not stable for a long period. Hence, this dissertation is motivated by the following questions: is there an

efficient way to compute just one partition, no update needed, that is robust for a highly time-varying workload? Especially, what if we do not know the interaction rates at any time? By "robust", we mean that the cost to process the workload at any given time remains small despite unpredictable workload increases. Another consideration is geographical awareness. The edge servers should be geographically close to their respective user cells for maximizing the benefits of MEC. This dissertation presents novel solutions to address these issues. The theoretical findings are substantiated by evaluation studies using real-world data.

Table of Contents

CHAPTER 1

INTRODUCTION

Mobile Edge Computing (MEC) [ETS14] has emerged as a viable technology for mobile operators to push computing resources closer to the users so that requests can be served locally without long-haul crossing of the network core, thus improving network efficiency and user experience. In a nutshell, a typical MEC architecture consists of four layers of entities: the mobile users, the base-stations, the edge servers, and the cloud datacenter. The edge servers are introduced in the edge of the network connecting the base-stations to the network core, each server being an aggregation hub, or a mini datacenter, to offload processing tasks from the remote datacenter. Because the region, also known as "cloudlet" [SBC09], served by an edge server is much smaller, a commodity virtualization-enabled computer can be used to run computation and network functions that would otherwise be provided by the datacenter.

MEC can benefit many computation-hungry or latency-critical applications involving video optimization [XLT17], content delivery [SHZ17], big data analytics [NRS17], roadside assistance [PGF18], and augmented reality [AS17], to name a few. Originally initiated for cellular networks to realize the 5G vision, MEC has been generalized for broader wireless and mobile networks [MYZ17]. It is becoming more of a phenomenon with the Internet of Things; more than 5 billion IoT devices would be connected to MEC by 2020 according to a January 2017 forecast

by BI Intelligence [New17].

1.1 Motivation and Goals

A challenge with MEC is how to assign a given set of edge servers to the user cells to maximize the edge computing benefit. To address this challenge is application-specific. We focus on applications involving pairwise interactions between devices, such as voice-over-ip calls made from one user to another, peer-to-peer video streaming, and multi-player online gaming.

The server assignment problem in general is not new outside MEC. Indeed, it belongs to the body of work on distributed allocation of resources (virtual machines) widely studied in the area of cloud computing [HKL14, AL12, Man15, DHY17]. The MEC assignment problem is similar, but it has unique objectives and constraints. Firstly, the servers in MEC need be near the user side, not the datacenter side, and so the communication cost to be minimized is due to the use of the backhaul network (towards the datacenter), not the front-haul (towards the cells). Secondly, the geographic spread of the cells served by a MEC server should be a design factor, which is not a typical priority for a distributed cloud solution.

Specifically, there are several important considerations:

- If the two interacting devices are served by different edge servers, the cloud must get involved, hence a backhaul cost. This cost is avoided if the same server serves both devices. Intuitively, those devices that frequently interact should be assigned to the same server.

- However, it makes no practical sense if they are located in far remote geographic locations because a server should be geographically close to where it

serves to avoid high installation cost and long latency [BC17, MSG15]. Although geographic proximity tends to imply high transactional activity, this relationship is not straightforward.

- In practice, the interaction rates are heterogeneous, and, even for the same pair of devices, the rate can change over time. As such, with a limited number of edge servers that have bounded capacity, we cannot equally please all the users.

The MEC server assignment problem has been addressed in some forms, only recently [CPS17, BC17]. Obviously, with zero knowledge about the interaction workload, we cannot do better than a random assignment: assigning each cell to an arbitrary edge server. Therefore, the conventional problem setting in the literature is: (1) assume that the interaction workload between two cells has a known interaction rate, (2) compute a partition optimized for these rates, for example, by solving a weighted-graph partitioning problem, and (3) for a time-varying workload, incrementally re-compute the partition when the interaction rates change.

The above setting is suitable only for infrequently changing workloads. The operational and computation costs of the partition update can be expensive and it is difficult to estimate interaction rates if they are not stable for a long period. Hence, we ask: is there an efficient way to compute just one partition, no update needed, that is robust for a highly time-varying workload? Especially, what if we do not know the interaction rates at any time? By "robust", we mean that the cost to process the workload at any given time remains small despite unpredictable workload increases.

The goal of this dissertation is to address these questions. Ideally, in looking

for a server assignment solution, we should take into account the geographical constraint that should be enforced in the assignment.

1.2 Contributions

We reckon that in practice the changes in the workload demand over the time should follow a pattern rather than be completely random. We may not know the exact interaction rates, nor the form of their pattern, but we can measure the sample mean and variance. Our hypothesis is that this modest information can be useful for finding a good robust partition. We make the following contributions:

- We propose an optimization formulation to quantify the optimality of a partition for a highly time-varying workload given a number of edge servers. We show that even if the interaction rates are known a priori for all the time, to compute an optimal partition is NP-hard.
- We propose an effective probabilistic partitioning algorithm that does not require known interaction rates over the time, but only their sample mean and variance. A nice property is that this algorithm does not depend on any user-defined parameter.
- The proposed algorithm is based on our idea of approximating the time-varying interaction rate between each pair of cells by a stochastic variable. We know of no prior work on edge server assignment where the time-varying workloads are represented by probability distributions.
- We extend the probabilistic partitioning approach so that it can be applied to the server assignment problem with geographical constraints. We also

propose an alternative algorithm for the case these constraints are formulated in a certain way.

The work of this dissertation extends upon two refereed publications of mine. The findings of the probabilistic partitioning approach have been published in the Proceedings of the 2019 IEEE International Conference on Computer Communication and Networks [VT19]. Some findings on the geographically aware server assignment have been published in the International Journal on Parallel, Distributed, and Emergent Systems [TV18].

1.3 Organization

This dissertation is organized as follows. Chapter 2 provides a review of the pertinent literature, the general research problem and theoretical results showing their computational hardness. The probabilistic partitioning algorithm is presented and evaluated in Chapter 3. The server assignment problem with geographical constraints is addressed in Chapter 4. A combination of both problems will be presented in Chapter 5. The dissertation is concluded in Chapter 6 with pointers to future research.

CHAPTER 2

PRELIMINARIES

In this chapter, we describe the problem setting, its goal and objectives formally and discuss the related work.

2.1 Problem Setting

Consider a geographic region that is partitioned into a set of cells, $\mathcal{C} = \{1, 2, ..., N\}$ cells, each cell serving a geographic area of service. Each cell $i \in \mathcal{C}$ is operated by a base-station; we also refer to this base-station as base-station i. The meanings of "base-station" and "cell" are generic, not necessarily understood in conventional meaning as in cellular networks; for example, as a Wi-Fi access router and its coverage area in Wi-Fi networks.

For simplicity, time is divided into slots, $t = 1, 2, 3, ..., T$. The system workload at time t is the set of interaction volumes for all pairs of cells at t. We express the time-varying workload as a third-order tensor,

$$W_T = \begin{bmatrix} W^{(1)} \\ W^{(2)} \\ \cdots \\ W^{(T)} \end{bmatrix}$$

where $W^{(t)}$ is a symmetric $N \times N$ matrix of non-negative real values

$$W^{(t)} = \begin{bmatrix} w_{11}^{(t)} & w_{12}^{(t)} & \dots & w_{1N}^{(t)} \\ w_{21}^{(t)} & w_{22}^{(t)} & \dots & w_{2N}^{(t)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N1}^{(t)} & w_{N2}^{(t)} & \dots & w_{NN}^{(t)} \end{bmatrix}$$

with $w_{ij}^{(t)}$ representing the interaction rate (or demand) at time t between users in cell $i \in \mathcal{C}$ with users in cell $j \in \mathcal{C}$. Note that $w_{ii}^{(t)}$ is the interaction rate at time t between users in the same cell i.

Suppose that we can afford to deploy a given number of edge servers, $\mathcal{S} = \{1, 2, ..., M\}$, each having a compute capacity κ. Each server is exclusively allocated to a group of cells to help process their interactions. A cell may be assigned to a server or no server at all. The cells not assigned to any server are referred to as "unassigned". The server assignment is denoted by an integer variable $f_i \in \mathcal{S} \cup \{0\}$, such that f_i is set to 0 for an unassigned cell i, or else it denotes the server to whom cell i is assigned. We need to compute f. In what follows, without explicitly specified otherwise, we use indices i, j to represent a cell/base-station, and s to represent a server.

2.2 Cost Formulation

We want to minimize the backhaul cost. The *backhaul cost* is incurred when the data center has to process:

1. The interactions involving the unassigned cells.

2. The interactions between cells assigned to different servers,

3. The interactions assigned to the same server that lead to a violation of its capacity.

In contrast, the *edge cost* is incurred due to the processing of the interactions involving assigned cells of the same server, provided the workload does not exceed this server's capacity. At any time t, the sum of the backhaul cost and edge cost is equal to the total workload at time t, $\sum_{i<j} w_{ij}^{(t)}$.

The backhaul cost at time t to process all the interactions involving the unassigned cells is

$$\mathsf{cost}_{unassigned}^{(t)} = \sum_{i:f_i=0} \left(\sum_{j:j\geq i \wedge f_j=0} w_{ij}^{(t)} + \sum_{j:f_j>0} w_{ij}^{(t)} \right). \tag{2.1}$$

The backhaul cost at time t to process all the interactions involving cells assigned to different servers is

$$\mathsf{cost}_{crossserver}^{(t)} = \sum_{i,j:0<f_i<f_j} w_{ij}^{(t)}. \tag{2.2}$$

The backhaul cost at time t to process all the interactions involving cells of the same server, that exceed its capacity, is

$$\mathsf{cost}_{overload}^{(t)} = \sum_{s} \underbrace{\max\left(0, (w_s^{(t)} - \kappa)\right)}_{overload\ amount\ of\ server\ s} \Bigg). \tag{2.3}$$

Here,

$$w_s^{(t)} = \sum_{i,j:j\geq i \wedge f_i=f_j=s} w_{ij}^{(t)}$$

is the workload demand on server s at time t (i.e., the amount of workload this server is supposed to process). Ideally, this demand should not exceed κ, the server capacity. In the otherwise case, the datacenter has to process the amount of workload that goes beyond κ.

The total backhaul cost at time t is the sum of the three costs,

$$\begin{aligned}\mathsf{cost}^{(t)} &= \mathsf{cost}^{(t)}_{unassigned} + \mathsf{cost}^{(t)}_{crossserver} + \mathsf{cost}^{(t)}_{overload} \\ &= \sum_{i:f_i=0} \left(\sum_{j:j\geq i \wedge f_j=0} w^{(t)}_{ij} + \sum_{j:f_j>0} w^{(t)}_{ij} \right) + \sum_{i,j:0<f_i<f_j} w^{(t)}_{ij} + \sum_{s} \max\left(0, (w^{(t)}_{s} - \kappa)\right).\end{aligned} \tag{2.4}$$

Consequently, the total backhaul cost over period $[1, T]$ is

$$\mathsf{cost} = \sum_{t=1}^{T} \mathsf{cost}^{(t)}. \tag{2.5}$$

2.3 Optimization Objective

Ideally, we want a partition f such that when applied to a (future) time-varying workload W_T the total backhaul cost is minimal. If we know *nothing* about W_T, we can do no better than the random assignment, which assigns each cell to an arbitrary server. On the other hand, if we assume to know *everything* about W_T, i.e., the interaction rates $w^{(t)}_{ij}$ for all cell pairs (i, j) at every time t are known, this assumption may be too strong to be practical for time-varying W_T.

In this dissertation, for each cell pair (i, j), instead of assuming that $w^{(t)}_{ij}$ is known for all t, we assume to know only their mean μ_{ij} and variance σ^2_{ij}. This is a realistic assumption because we can easily measure these quantities from samples (past interaction rates). In practice, the interaction demand between two cells may change frequently over the time, but the change should follow a pattern, and so the sample mean and sample variance are reliable.

Let

$$\boldsymbol{\mu} = \left(\mu_{ij}\right)_{N\times N},\ \boldsymbol{\sigma}^2 = \left(\sigma^2_{ij}\right)_{N\times N}.$$

We say that a time-varying workload W_T is *consistent* with $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ if μ_{ij} and σ_{ij}^2 are (reasonable) sample mean and variance of $w_{ij}^{(t)}$ over time $t \in [1, T]$.

Our optimization problem is as follows.

Problem 2.3.1. *Given $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, compute a partition f such that when applied to a future time-varying workload W_T the total backhaul cost over the time period $[1, T]$ is minimal; i.e.,*

$$\min \left\{ \mathsf{cost} = \sum_{t=1}^{T} \mathsf{cost}^{(t)} \right\}, \tag{2.6}$$

provided that W_T is consistent with $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$.

2.4 Computational Hardness

Unfortunately, knowing only the mean and variance information of the future workload W_T, it is impossible to solve this optimization problem (because the precise interaction rates need be known). Even if all the rates at all times are known in advance, the optimization of Problem 2.3.1 is NP-hard. In deed, consider a simple configuration: $T = 1$ and $w_{ij}^{(1)} = 0$ for all $i \neq j$ and $w_{ii}^{(1)} > 0$ for all i. Then,

$$\begin{aligned}
\mathsf{cost}_{unassigned}^{(1)} &= \sum_{i:f_i=0} w_{ii}^{(1)} \\
\mathsf{cost}_{crossserver}^{(1)} &= 0 \\
\mathsf{cost}_{overload}^{(1)} &= \sum_{s} \max\left(0, \left(\sum_{i:f_i=s} w_{ii}^{(1)} - \kappa \right)\right).
\end{aligned}$$

and so,

$$\begin{aligned}\mathsf{cost} &= \mathsf{cost}^{(1)} \\ &= \sum_{i:f_i=0} w_{ii}^{(1)} + \sum_{s} \max\left(0, \left(\sum_{i:f_i=s} w_{ii}^{(1)} - \kappa\right)\right).\end{aligned}$$

Suppose that we set $|\mathcal{S}| = 2$ and $\kappa = \frac{1}{2}\sum_i w_{ii}^{(1)}$. Then, cost is minimized by assigning the cells to the two servers $s = 1$ and $s = 2$ such that $\sum_{i:f_i=1} w_{ii}^{(1)}$ and $\sum_{i:f_i=2} w_{ii}^{(1)}$ differ the least. Thus, this is is equivalent to the following NP-hard partition problem: partition a given set of positive integers, $\{x_1, x_2, ..., x_n\}$, into two subsets such that the respective subset sums differ the least (here, simply set $x_i = w_{ii}^{(1)}$). For more details of this equivalence, see [TV18]. The optimization of Eq. (5.1), due to its generality, is therefore NP-hard.

2.5 Geographical Awareness

Ideally, a server would be placed at the center of its cluster. Consequently, for shorter latency and easier management, we should keep the geographic region of this cluster from spreading too far. We can formulate this property as an objective to optimize or as a constraint to enforce.

As a constraint, we can require that the cells form a geographically contiguous region [BC17, MSG15]; we refer to this requirement as being geo-contiguity. Alternatively, we can set an upper bound on the geographical diameter of each cluster resulted in from the server assignment; we refer to this requirement as geo-compactness.

As an objective, we can quantify the geo-spread of a cluster as the sum of its distance to each assigned base-station, weighted by transaction demand. We want to minimize the total geo-spread for all the servers.

In Chapter 3, we propose a probabilistic partitioning framework to find an approximate solution for Problem 2.3.1 without consideration of geographical awareness. In Chapter 4, we extend this framework to find a solution that satisfies the geo-contiguity constraint, and also propose a different solution that is aimed to optimize geo-compactness (together with backhaul cost).

2.6 Related Work

The MEC server assignment problem can arise in various scenarios. The assignment problem in [WZT17] applies to a MEC network supporting multiple applications of known request load and latency expectation and the challenge is to determine which edge servers to run the required virtual machines (VMs), constrained by server capacity and inter-server communication delay. In [BG17,WZL17], where only one application is being considered and consists of multiple inter-related components organizable into a graph, the challenge is how to place this component graph on top of the physical graph of edge servers to minimize the cost to run the application. In the case that edge servers must be bound to certain geographic locations, a challenge is to decide which among these locations we should place the servers and inter-connect them for optimal routing and installation costs [CPS17].

The above works do not take into account the geographic spread of the region served by a server. The cells served by the same server can be highly scattered geographically, causing long latency and high management cost. This motivates the work in [BC17] proposing a spatial partitioning of the geographic area such that the cells of the same server are always contiguous. For this partitioning, a graph-based clustering algorithm is proposed that repeatedly merges adjacent cells to form clusters as long as the merger results in better offloading and no

cluster exceeds the server capacity. In a similar research [MSG15], where the cells served by each server are also contiguous, the objectives are to minimize the server deployment cost, the front-haul link cost for each server to reach its assigned base-stations, and the cell-to-cell latency via the edge; the proposed algorithm is to repeatedly select the next remaining server of the least deployment cost and assign it to all the nearby base-stations of the least front-haul link cost so long as the server capacity is met.

For time-varying workloads, the conventional or implicit approach in all the existing techniques is to either to recompute the partition or make incremental adjustments. They all require that interaction rates be known prior to the update. In contract, our goal is to compute one single partition that stays unchanged but is robust for a long sequence of changing workloads and, importantly, without knowing the interaction rates. To the best of our knowledge, this is an original problem.

CHAPTER 3

PROBABILISTIC PARTITIONING

In this chapter, we focus on solving Problem 2.3.1 without consideration of geographical constraints. The variables $\{f_i\}$ represent a partition of the cells into $(M+1)$ groups, M of which are assigned to the M servers and the last is the group of unassigned cells. A group of cells is called a "supercell". We use letters x, y, ... to denote supercells.

As discussed in the previous chapter, this problem is NP-hard even if we know all the interaction rates between the cells in advance. Therefore, we seek an approximate solution. We propose an iterative partitioning algorithm based on the idea of modularity optimization in complex networks [New06]: repeatedly merge the vertices of the network, represented as a weighted graph, to obtain a smaller graph of super-vertices, called "modules", that is better in terms of "modularity". The modularity is some quality function such that, conventionally, networks with high modularity have dense connections between the nodes within modules but sparse connections between nodes in different modules.

In our context, the vertices of the original graph are the original cells, which are merged into super-vertices (supercells) in the subsequent graphs. The key ingredient in our algorithm is the definition of the quality function. For ease of explanation, we focus first on the special case where the workload is time-invariant (zero variance), and then the general case where the workload is time-varying

(positive variance).

3.1 Time-Invariant Workload

Suppose that the workload does not change over the time, i.e., $w_{ij}^{(t)} = \mu_{ij}$ for all $t \in [1, T]$ and $\sigma_{ij} = 0$ for all $i, j \in C$. Let u_{xy} denote the workload demand between the users in a supercell x and the users in a different supercell y and u_x denote the workload demand between users in the same supercell x. We can write

$$u_{xy} = \sum_{i \in x} \sum_{j \in y} \mu_{ij}, \; u_x = \frac{1}{2} \sum_{i \in x} \sum_{j \in x} \mu_{ij}.$$

Here, "$i \in x$" means that cell i is a member of supercell x.

The optimization problem becomes a graph partitioning problem with fixed edge weights. To apply modularity optimization, we start with a graph where each vertex is an original cell and after each merging step we obtain an immediate graph of supercells; here, the weight of the edge connecting supercell x and supercell y is u_{xy}. The vertex merging goes on as long as we can increase u_x for some vertex x. Constrained by the capacity bound κ, a merger to form a supercell x is allowed only if the new $u_x \leq \kappa$. The detailed steps of this algorithm are as follows.

Algorithm 3.1.1 (Time-Invariant Workload)**.**

1. *Initially, we start with a graph of N supercells (vertices), each supercell being an original cell: $\mathcal{X} = \{\{1\}, \{2\}, ..., \{N\}\}$.*

2. *While (true)*

   - *Maintain a list of edges $(x, y) \in \mathcal{X} \times \mathcal{X}$ sorted in descending order of u_{xy}.*

- *Find the first edge (x, y) in this list such that $u_{x \cup y} \leq \kappa$, i.e., the workload demand of the supercell merging x and y would be under the server capacity. If this edge is not found, go to Step 3.*
- *Merge vertex x and vertex y to form a new vertex corresponding to supercell $x \cup y$.*
- *Update $\mathcal{X}$ and its edge weights.*

3. *At this time, we obtain a graph of m vertices (supercells), corresponding to a partition into m supercells.*
4. *If $m \geq M$, we choose the M supercells x of largest u_x and assign them respectively to the M servers. The cells in the remaining supercells are unassigned.*
5. *Else, the partition has fewer supercells than servers. Each supercell is assigned to a server. In this case, there is no unassigned cell and one or more servers may be idle (suggesting that m servers would be sufficient).*

In understanding of this algorithm, intuitively, we can think of u_{xy} as the inter-module density and u_x as the intra-module density, and if we merge x and y we would improve modularity because of the reduction in inter-module density and increase in intra-module density. Translating this to our cost model, by enforcing the condition $u_x \leq \kappa$, we keep $\mathsf{cost}_{overload} = 0$ (Eq. (2.3)) and after each vertex merging we obtain reductions in $\mathsf{cost}_{unassigned}$ (Eq. (2.1)) and $\mathsf{cost}_{crossserver}$ (Eq. (2.2)). By preferring the edge (x, y) with largest u_{xy} in Step 2, we want to maximize the modularity improvement.

3.2 Time-Varying Workload

In the general case, we consider a time-varying workload and the only information we assume given is the sample mean and variance of the interaction rate for each cell pair. Our approach is to approximate the workload by a Gaussian distribution. Specifically, we model the interaction rate $w_{ij}^{(t)}$ of each cell pair (i, j), which is unknown, as a Gaussian random variable with mean μ_{ij} and variation σ_{ij}^2,

$$w_{ij}^{(t)} \sim \mathcal{N}\Big(\mu_{ij}, \sigma_{ij}^2\Big).$$

The Gaussian distribution has the property that the sum of Gaussian variables is also Gaussian. Consequently, the workload demand between supercell x and supercell y at a time t is the following Gaussian variable

$$u_{xy}^{(t)} \sim \mathcal{N}\Big(\sum_{i\in x}\sum_{j\in y}\mu_{ij}, \sum_{i\in x}\sum_{j\in y}\sigma_{ij}^2\Big).$$

Similarly, the workload demand of a supercell x is the following Gaussian variable

$$u_x^{(t)} := \frac{1}{2}u_{xx}^{(t)} \sim \mathcal{N}\Big(\frac{1}{2}\sum_{i\in x}\sum_{j\in x}\mu_{ij}, \frac{1}{2}\sum_{i\in x}\sum_{j\in x}\sigma_{ij}^2\Big).$$

Denote

$$\mu_{xy} = \sum_{i\in x}\sum_{j\in y}\mu_{ij},\ \ \sigma_{xy}^2 = \sum_{i\in x}\sum_{j\in y}\sigma_{ij}^2$$
$$\mu_x = \frac{1}{2}\sum_{i\in x}\sum_{j\in x}\mu_{ij},\ \ \sigma_x^2 = \frac{1}{2}\sum_{i\in x}\sum_{j\in x}\sigma_{ij}^2.$$

We propose an assignment algorithm similar to Algorithm 5.2.1 for the time-invariant workload case, except for the merging criterion in the iterative merging step (Step 2 in Algorithm 5.2.1). Specifically, two supercells x and y are chosen to merge if the would-be resultant supercell $x \cup y$ has a high probability, at least a given threshold θ, of having a workload demand below the server capacity κ.

For a Gaussian variable $X \sim \mathcal{N}(\mu, \sigma^2)$, its cumulative distribution function at κ is

$$\mathsf{cdf}(\kappa|\mu,\sigma) = \mathsf{Prob}[X \leq \kappa] = \frac{1}{2}\left[1 + \mathsf{erf}\left(\frac{\kappa-\mu}{\sigma\sqrt{2}}\right)\right]$$

where erf is the Gaussian error function

$$\mathsf{erf}(x) = \frac{2}{\pi}\int_0^x e^{-t^2}dt.$$

Therefore, we merge supercells x and y into a new supercell $x \cup y$ if

$$\mathsf{cdf}(\kappa|\mu_{x\cup y}, \sigma_{x\cup y}) \geq \theta.$$

Note that the workload demand of the new supercell $x \cup y$ follows a Gaussian distribution with mean $\mu_{x\cup y}$ and variance $\sigma^2_{x\cup y}$, which can easily be computed :

$$\mu_{x\cup y} = \mu_x + \mu_y + \mu_{xy}$$
$$\sigma^2_{x\cup y} = \sigma^2_x + \sigma^2_y + \sigma^2_{xy}.$$

Following this merging criterion, the final partition result depends on the choice of threshold θ. In the extreme case $\theta = 0$, the server capacity is ignored and the algorithm produces one big supercell merging all the original cells. In the other extreme case $\theta = 1$, no cells are merged and so the algorithm results in a partition that is the set of the original cells. In general, a larger/smaller θ tends to result in a partition with more/fewer supercells. Therefore, it is critical to choose a good value for θ that should result in the best partition for the given number of servers, M.

Because we do not know this "optimal" θ value in advance, our proposed algorithm starts with the largest $\theta = 1$ and gradually decrements it as long as the resultant partition has more supercells than servers. The merging continues with this new θ. The algorithm stops when the number of supercells reaches M or smaller. The proposed algorithm works in detail as follows.

Algorithm 3.2.1 (Time-Varying Workload)**.**

1. *Initially, we start with a graph of N supercells (vertices), each supercell being an original cell: $\mathcal{X} = \{\{1\}, \{2\}, ..., \{N\}\}$.*

2. *Initialize $\theta = 1$.*

3. *While (true)*

   - *Maintain a list of edges $(x, y) \in \mathcal{X} \times \mathcal{X}$ sorted in descending order of μ_{xy}.*
   - *Find the first edge (x, y) in this list such that*

   $$\mathsf{cdf}(\kappa|\mu_{x\cup y}, \sigma_{x\cup y}) \geq \theta$$

   - *If this edge is not found, go to Step 3. Else, merge vertex x and vertex y to form a new vertex corresponding to supercell $x \cup y$.*
   - *Update $\mathcal{X}$ and its edge weights.*

4. *At this time, we obtain a graph of m vertices (supercells), corresponding to a partition into m supercells.*

5. *If $m > M$*

   - *Set $\theta = \theta - \epsilon$ (step size ϵ, sufficiently small, e.g., 0.05 in our evaluation).*
   - *Go back to Step 2 to repeat the merging procedure.*

6. *Else, the partition now has the same or fewer supercells than servers. Each supercell is assigned to a server. In this case, there is no unassigned cell and one or more servers may be idle (suggesting that m servers would be sufficient).*

3.3 Evaluation

We conducted an evaluation[1] on three real-world datasets we had constructed from the collection of geo-referenced Call Detail Records over the city of Milan during November 2013 and December 2013 (https://dandelion.eu/).

3.3.1 Setup

The datasets correspond to three different days (Fri Nov 1, Sat Nov 2, and Tue Nov 5), each representing a time-varying workload for evaluation. Specifically, we partitioned the area, 23.5km $\times$ 23.5km, into a grid of $N = 20 \times 20 = 400$ cells and counted calls between every two cells for each 10-minute period during an entire day. The time duration of the entire workload is therefore T = 24 hours / 10 minutes = 144 time slots. In each dataset, we compute the mean and variance representing the interaction rate of a cell pair (i, j) as follows:

$$\mu_{ij} = \frac{1}{T}\sum_{t=1}^{T} w_{ij}^{(t)},\ \sigma_{ij}^2 = \frac{1}{T-1}\sum_{t=1}^{T}(w_{ij}^{(t)} - \mu_{ij})^2.$$

Note that these μ_{ij} and σ_{ij}^2 values are the only information we assume known in our research problem.

We investigated different configurations by varying the number of MEC servers in the range $M \in \{5, 10, 15, 20, 25\}$ and the server capacity in the range $\kappa \in \{1\%$, 5%,10%, 15%, 20%$\}$. A choice of $\kappa = 10\%$ means that a server cannot process more than 10% of the average total workload (i.e., the total workload at an average time instant, which is the ratio of total workload over $[0, T]$ to the duration T).

We evaluated the proposed algorithm (Algorithm 5.2.2) to two benchmark algorithms: 1) RAND: the random assignment algorithm; and 2) BC (by **B**ouet and

[1] http://github.com/quynhtavo/probabilistic-partitioning

Conan [BC17]): this is the closest work to ours. It is an algorithm for a time-invariant workload very similar to Algorithm 5.2.1, but their cell merging criterion considers only geographically adjacent cells. To use it as benchmark for our evaluation, we take as input the workload where the interaction rates are the mean μ_{ij}'s and apply BC to obtain the assignment. For ease of presentation, we refer to our algorithm as VT (initials of the authors' names).

The main metric for comparison is the backhaul cost, normalized as the ratio of cost in Eq. (2.5) to the total workload over the time. For example, a ratio of 0.8 indicates that the backhaul cost of the MEC solution is only 80% that of the solution without MEC; hence a 20% cost saving by MEC.

3.3.2 Results

Most of the discussions below are on the results for the Friday and Tuesday datasets because they offer two very different workload scenarios. The result for the Saturday dataset is similar to that of the Friday and we will provide some of the former's results at the end of this section.

As a preview of our in-depth analysis, Figure 5.1 (for the Friday dataset) and Figure 5.2 (for the Tuesday dataset) provide a 3D illustration of the comparison of RAND, BC, and VT for all configurations of the evaluation. Although these datasets represent very different workloads, they share the following pattern: in the unrealistic case with too limited capacity and too few servers, there is no clear winner, but in most realistic cases where the capacity is not too low, VT is the best and RAND the worst, especially when the number of servers or the capacity increases.

The superiority of VT is more noticeable for the Tuesday dataset which repre-

sents a weekday with busy workloads during the working hours. In contrast, the Friday dataset represents a near-weekend day with busy workloads only during the rush hours. It is therefore implied that the effectiveness of VT is best showcased with busy workloads, which is a desirable property.

Figure 3.3 provides a sample visualization of how the cell-to-server assignment of each algorithm is laid out on the geographic map. In this sample, using the Friday dataset with five servers deployed and 20% capacity, VT results in a partition of five supercells and no unassigned cells. The partition of BC, however, consists of six supercells, leaving one supercell unassigned (the gray cells on the map). In this configuration, the backhaul cost of VT is only 86% of BC. Furthermore, although VT does not enforce the geographical contiguity in the cell merging criterion as in BC, the cells belonging to the same server as a result of VT do satisfy this property almost perfectly. In all configurations, this property does exhibit for most of the cells. There are several cases with remote cells in the same server but that might be due to a lot of interactions between them. Consequently, a nice property of VT is that its merging procedure though geographically blind still naturally offers geographical contiguity.

We discuss the comparison in more detail below.

3.3.2.1 Effect of the Server Capacity

Figure 3.4 demonstrates for the Friday dataset the effect on the backhaul cost as the server capacity increases using the same number of servers. With only five servers (Figure 3.4(a)), there is no clear difference among the three algorithms until the server capacity is sufficiently large. For example, when $\kappa \geq 10\%$, we start seeing VT and BC better than RAND. With 15 servers (Figure 3.4(b)), VT and

BC become substantially better than RANDOM when the server capacity exceeds 5%. With 25 servers (Figure 3.4(c)), we only need a server capacity of 1% or more to see this. In all configurations, the best cost RAND can achieve is 70% of the non-MEC cost; this figure is 57% for BC and 50% for VT.

Between VT and BC, Figure 3.4(d) illustrates the improvement of the former compared to the latter, here plotting the ratio of the backhaul cost of VT to that of BC. In all configurations except for the extreme case of too low capacity and too few servers, VT is the superior and the improvement is noticeable when the server capacity is sufficiently large. For example, for all cases of the number of servers, the cost of VT is less than 95% of BC when the capacity reaches 15% and less than 90% when the capacity is 25%. The trend in Figure 3.6(f) suggests that the improvement should continue to extend if the capacity can increase further.

We observe similar patterns for the Tuesday dataset, as seen in Figures 3.5(a), 3.5(c), and 3.5(e). It thus becomes obvious that VT performs the best, followed by BC, and RAND the worst. There is also an interesting observation. Whereas it is expected that the backhaul cost should improve as the server capacity increases, the improvement rate is not the same for the three algorithms. It is slowest with RAND and fastest with VT. For example, with the Friday dataset using 15 servers (Figure 3.4(b)), increasing capacity beyond 5% does not make RAND any better; this number is 15% for BC while VT continues to benefit from the increase beyond 20%. In other words, VT is not only better than BC (up to 14% in this evaluation) but would substantially be better (higher than 14%) if we could afford higher server capacity; of course, until the capacity becomes so high that it makes no significant difference between the two algorithms.

3.3.2.2 Effect of the Number of Servers

Figure 5.6 (for the Friday dataset) and Figures 3.5(b), 3.5(d), 3.5(f) (for the Tuesday dataset) demonstrate the effect on the backhaul cost as we add more servers, provided the server capacity is unchanged. Since the comparison patterns are similar for both datasets, we discuss the Friday dataset in what follows.

When the capacity is extremely low (1%, Figure 3.6(a)), the cost does improve, which is understandable because a server has too little capacity to make any edge computing benefit and so having more servers will help. However, when the capacity approaches a realistic value, it is observed that having more servers beyond a sufficient quantity does not help at all. We need only 10 servers for 10% capacity (Figure 3.6(c)) or 5 servers for 20% capacity (Figure 3.6(e)). This is because with more servers the intra-server cost is decreased but the cross-server cost can increase. For RAND, this tradeoff leads to even a worse overall backhaul cost. For VT and BC, the cost stays roughly the same.

Figure 3.6(f) plots the ratio of the cost of VT to that of BC. Similarly observed, increasing the number of servers beyond a certain threshold (10 servers) does not improve this ratio, meaning if we want to make VT increasingly better than BC we should not achieve that by adding more servers.

This study together with the study on the effect of the number of servers implies that it is the server capacity, not the number of servers, that has more influence on the efficiency of the server assignment. To lower the backhaul cost, it is better to increase the server capacity for each server than to add more servers of the same capacity.

3.3.2.3 Analysis of Cost in Real Time

In this study, we investigate the backhaul cost incurred when processing the workload at each time instant $t \in [0, T = 144]$. As we mentioned earlier, the Tuesday and Friday datasets represent two very different workloads. The former is for a weekday with busy workloads during the working hours (Figure 3.7(a)), while the latter is for a day with busy workloads only during the rush hours (Figure 3.7(a)). Also, there are some activities during the hours before and after midnight entering the Friday, while there are no such activities for the Tuesday.

For the Friday dataset, Figure 3.7(a) shows the absolute backhaul cost over the time and Figure 3.7(b) shows the backhaul cost normalized as a percentage of the total workload at the given time. Similarly, we have Figure 3.7(e) and Figure 3.7(f) for the Tuesday dataset. We also include the plots for another dataset (Saturday Nov 2) so that we have a diversity of different workloads (Figure 3.7(c) and Figure 3.7(d)).

It is understandable to see the normalized backhaul cost fluctuate wildly during light workload hours. This is understandable because when the workload is too light and the normalized cost is computed as a ratio to the workload, the value range of this ratio is wide. The more important observation here, however, is that the improvement of VT over BC is more significant during busy hours. For example, this applies to the Friday dataset around noon and 6PM (Figure 3.7(b)). The pattern is much more visible for the Tuesday dataset (Figure 3.7(f)), showing approximately a 45% cost saving by VT versus a 35% saving by BC.

This study further substantiates the cost efficiency of our proposed algorithm and supports our hypothesis that the mean and variance information can be very useful to finding a robust partition for highly time-varying workloads.

3.4 Summary

We have proposed an effective probabilistic algorithm to produce a partition of the cells to assign to the servers that is robust to the workload change dynamics. The idea is to approximate the interaction rate for each cell pair as a random Gaussian variable. Our evaluation has substantiated the effectiveness of the proposed algorithm for some real-world datasets. In the future, we will evaluate with real-world workloads following different probabilistic distributions and explore the use of non-Gaussian models for representing the interaction rates.

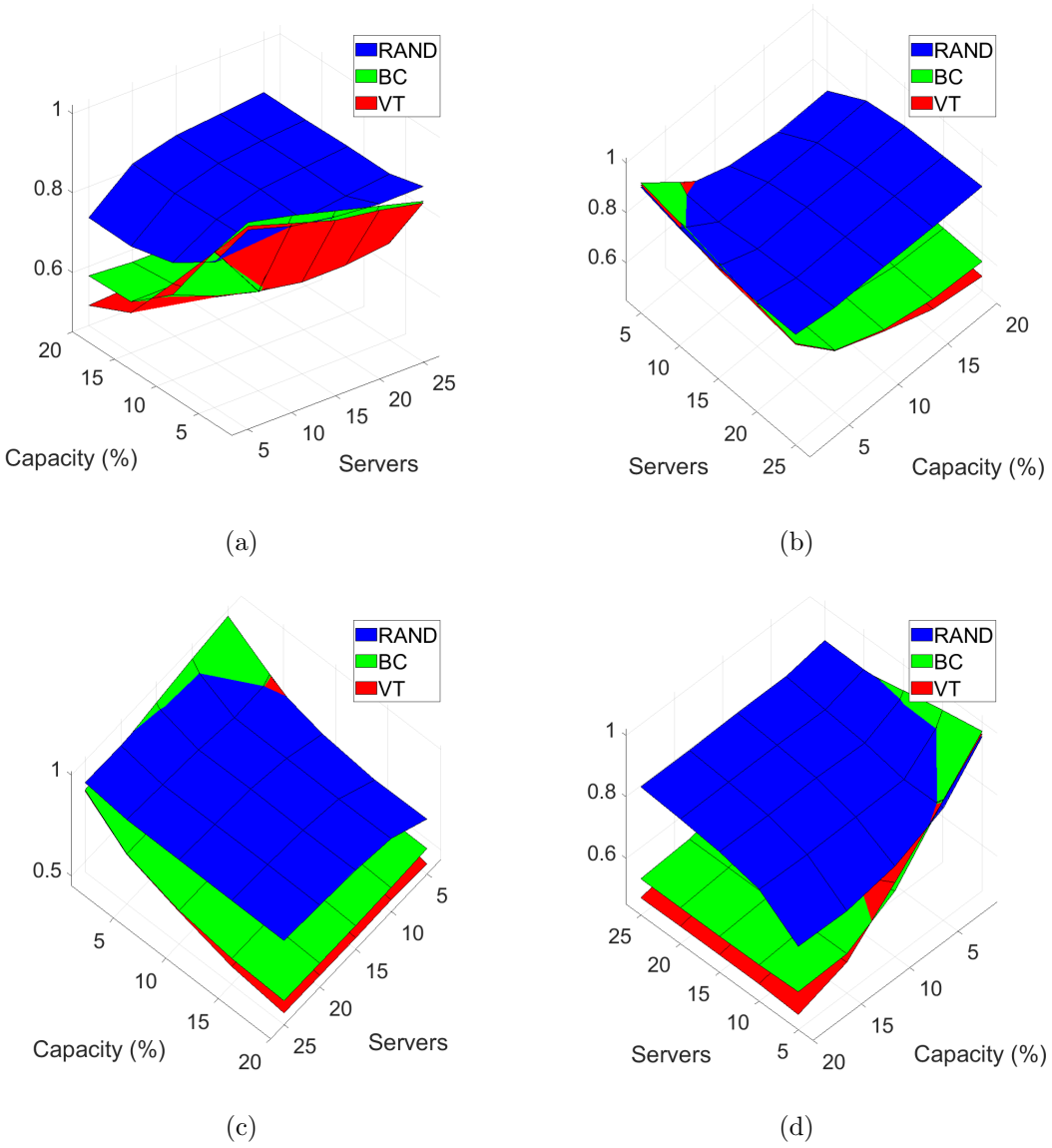


Figure 3.1: Friday Dataset: Comparison of the algorithms in all evaluation configurations, seen in different views. The z-axis is the backhaul cost.

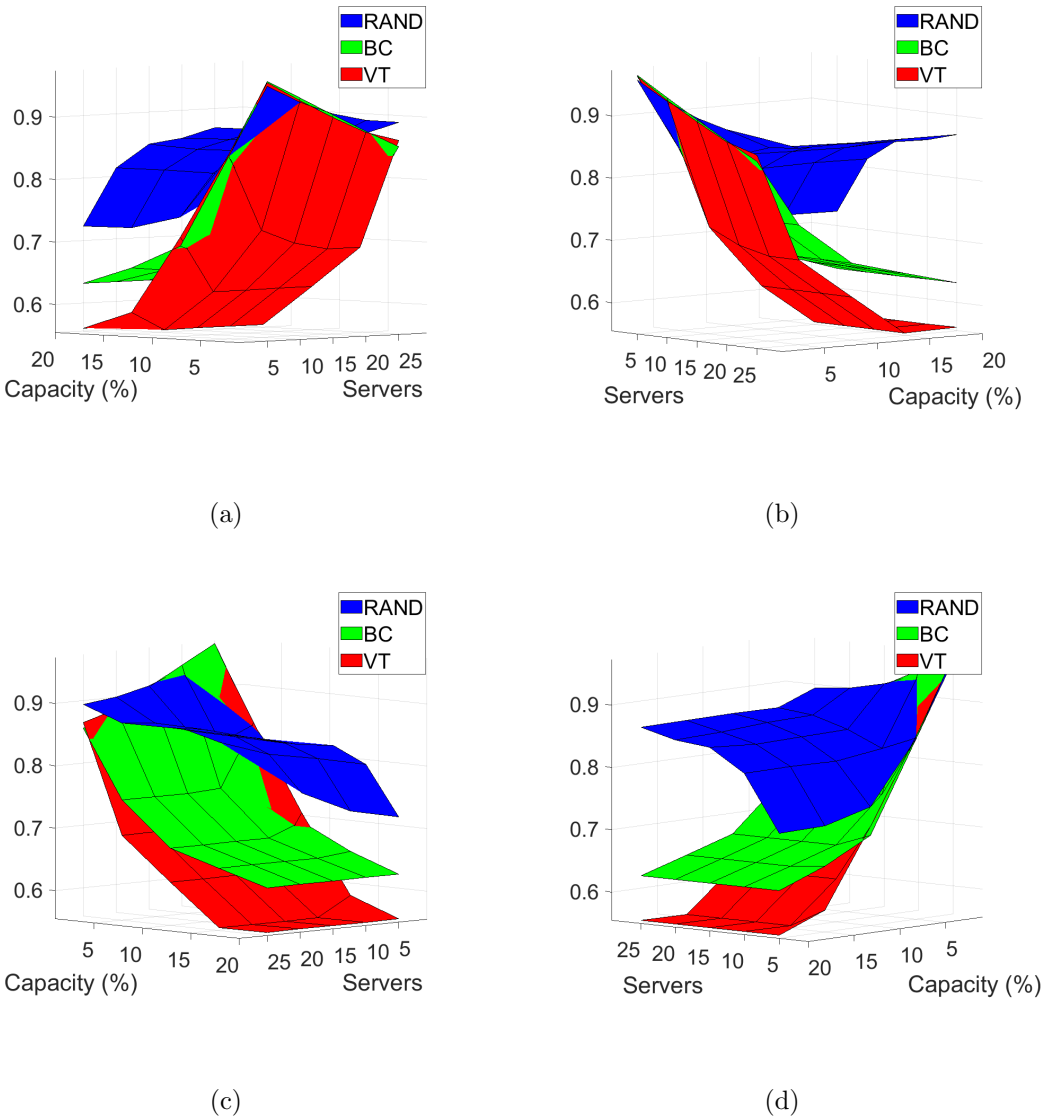
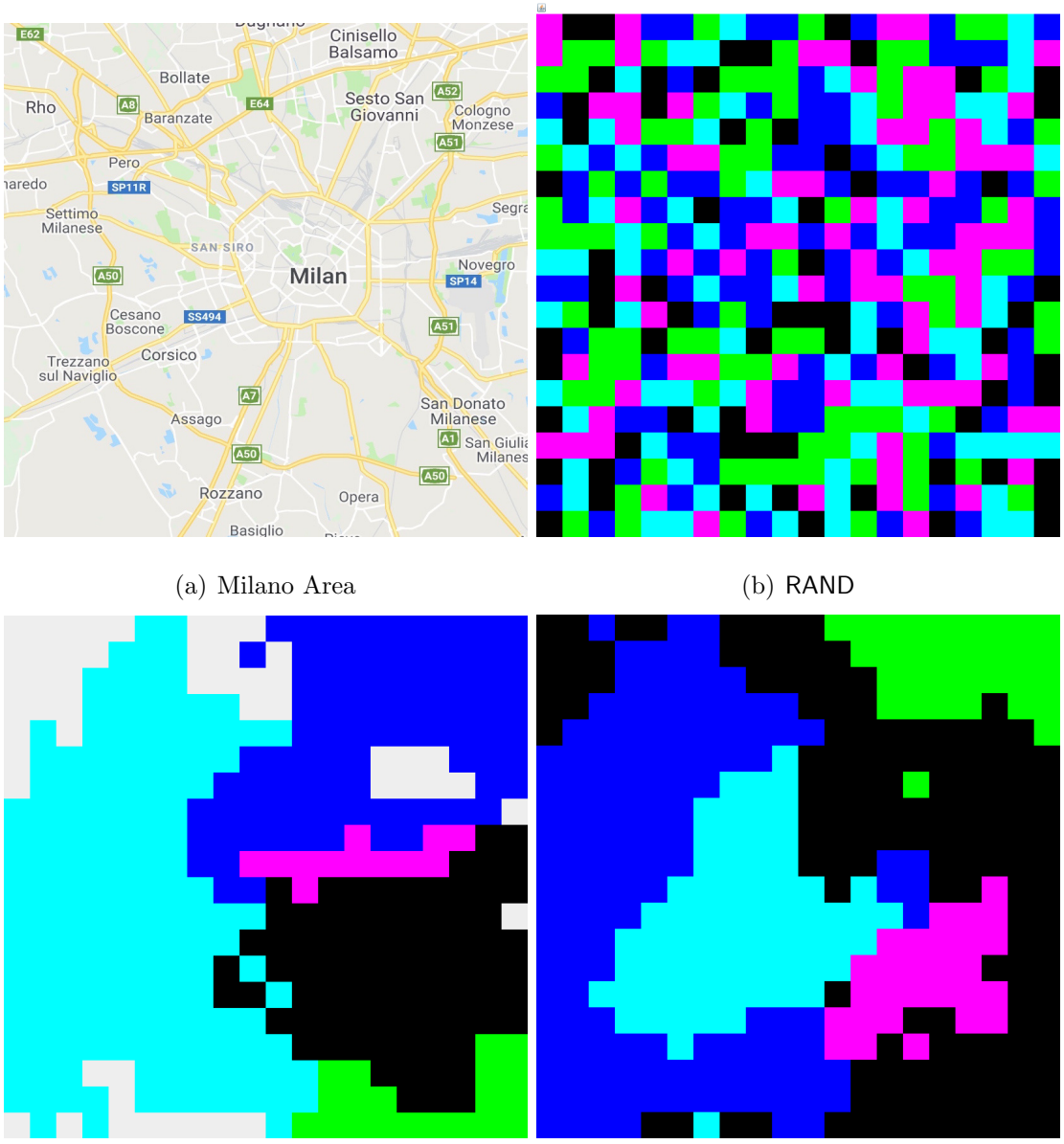


Figure 3.2: Tuesday Dataset: Comparison of the algorithms in all evaluation configurations, seen in different views. The z-axis is the backhaul cost.



(a) Milano Area

(b) RAND

(c) BC

(d) VT

Figure 3.3: The layout of the cell-to-server assignment resulted from each algorithm. These maps are for five servers and 20% capacity using the Friday dataset.

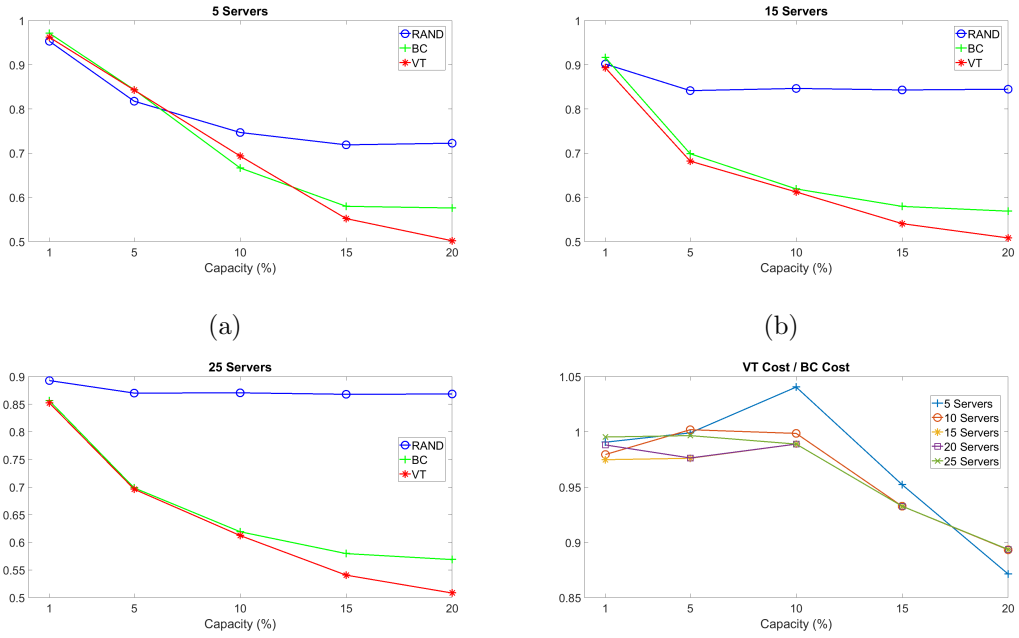


Figure 3.4: Friday Dataset: Effect of the server capacity.

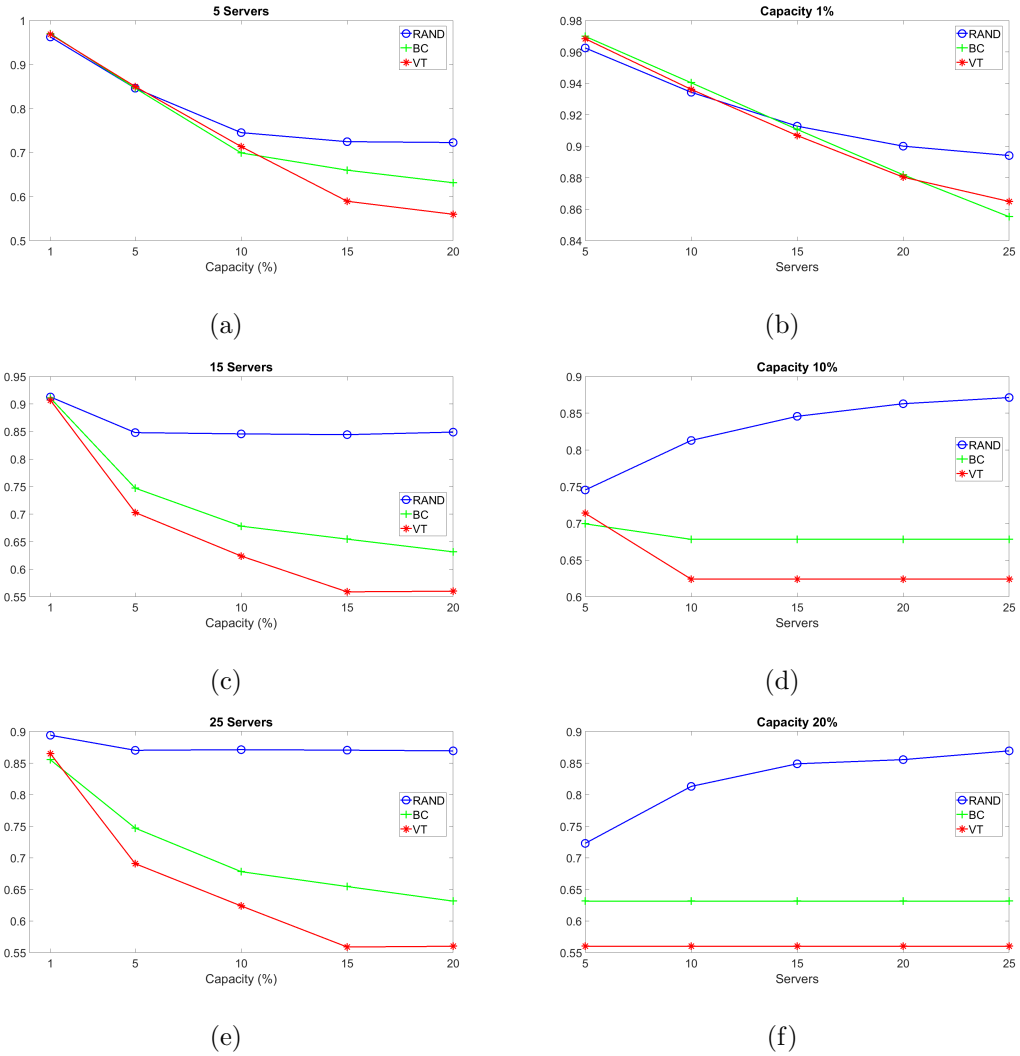


Figure 3.5: Tuesday Dataset: (a), (c), (e) Effect of the server capacity; (b), (d), (f): Effect of the number of servers.

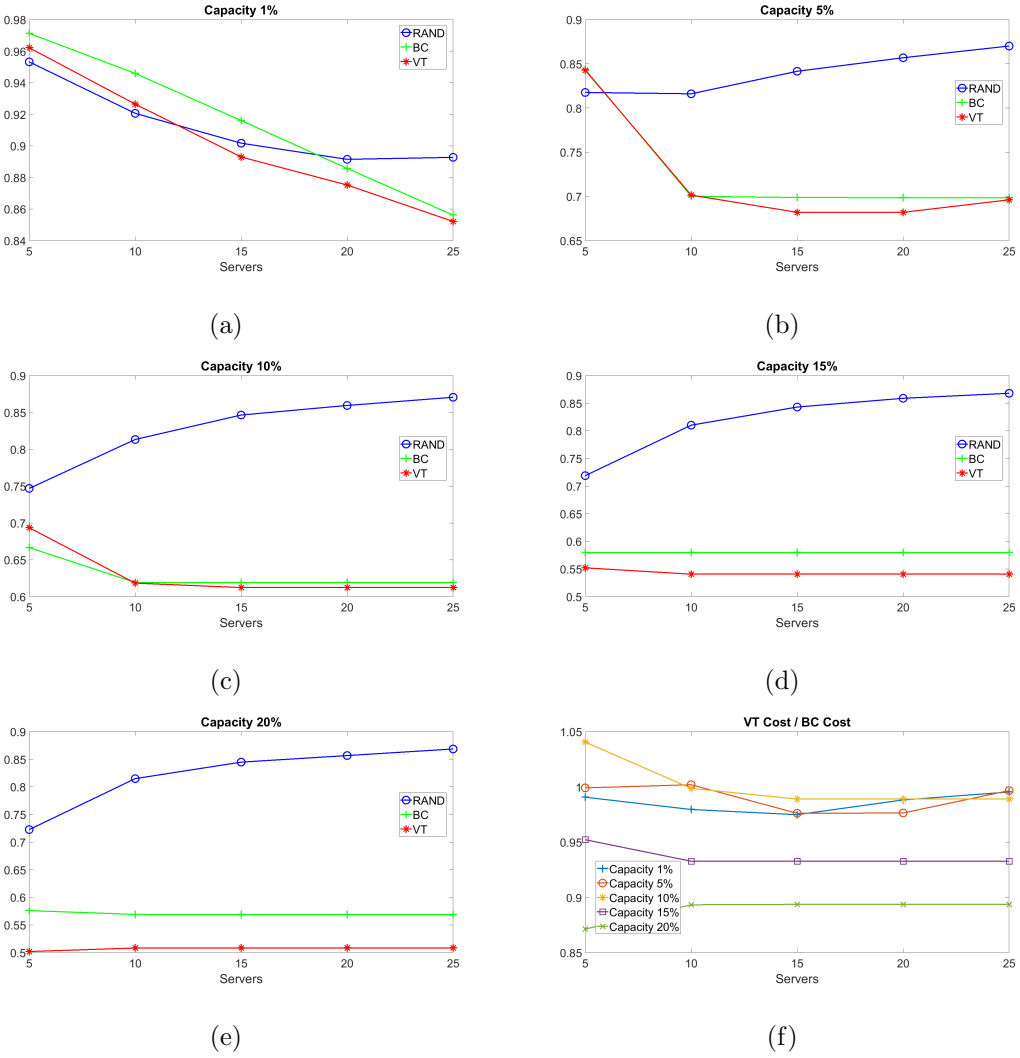
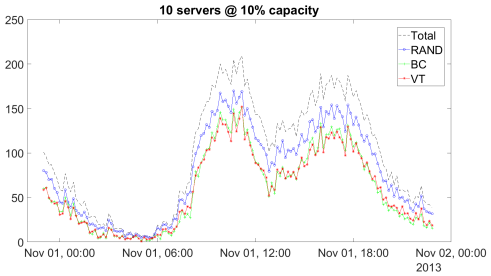
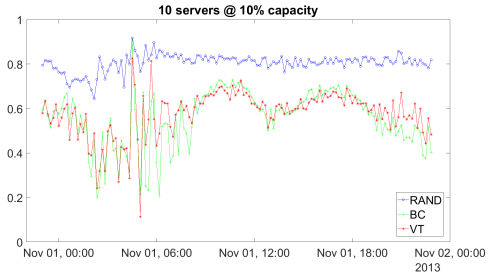


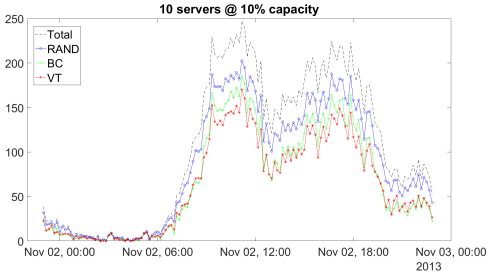
Figure 3.6: Friday Dataset: Effect of the number of servers.



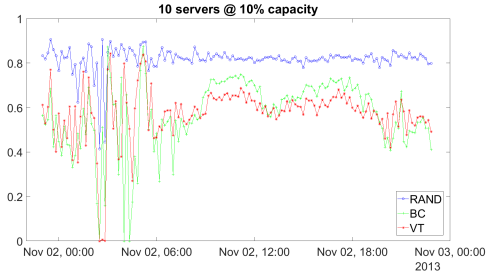
(a) Friday: absolute cost



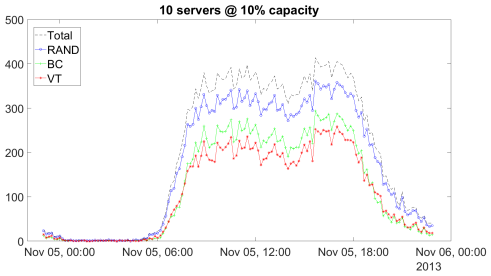
(b) Friday: normalized cost



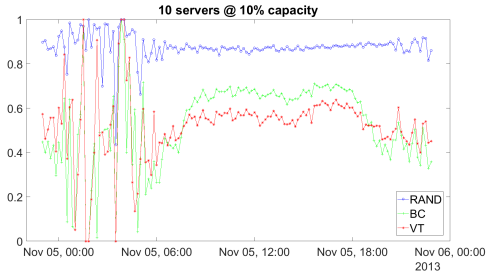
(c) Saturday: absolute cost



(d) Saturday: normalized cost



(e) Tuesday: absolute cost



(f) Tuesday: normalized cost

Figure 3.7: Real-time cost for the case of 10 servers at 10% capacity, shown for three different days. The left plots show the absolute value of the backhaul cost incurred by each algorithm together with the total workload at each time instant. The right plots show the backhaul cost normalized as a fraction of the total.

CHAPTER 4

GEO-AWARE PARTITIONING

A desirable property for the MEC server assignment is to place the edge servers geographically close to the users they respectively serve. As discussed in Chapter 2, the geographical awareness can be represented as a degree of geographical contiguity or compactness. In this chapter, we focus on achieving geographical compactness by formulating it as an objective to optimize.

4.1 Geo-Awareness Formulation

We introduce a geo-aware version for Problem 2.3.1. This version assumes given a set $\mathcal{L} = \{1, 2, ..., L\}$ of $L \geq M$ candidate locations for the edge servers. A server needs to be given a location in this set. For example, $\mathcal{L}$ can be a subset of base-station locations; in this case, an edge server must be co-located with some base-station (as in [MSG15]). In general, we admit any arbitrary candidate location.

Our goal consists of two tasks: (1) server location assignment (SLA): assign the best location for each server and (2) cell-server assignment (CSA): assign the best server for each cell. We introduce the following notations:

- An integer variable $g_s \in \mathcal{L}$ denotes the location of server s. Note that no servers are located at the same location.

- If a cell i is assigned to a server and this server is at location l, we have a front-haul link, whose usage cost should increase with their distance; denote this cost by d_{il} (assumed given, e.g., equal to distance).

We quantify the geo-spread of a server as the sum of its distances to the assigned cells, weighted by workload demand. At time t, the geo-spread of a server s is

$$\mathsf{spread}^{(t)}(s) = \sum_{i \in \mathcal{C}: f_i = s} d_{ig_s} w_i^{(t)}. \tag{4.1}$$

where

$$w_i^{(t)} = \sum_{j \in C} w_{ij}^{(t)}$$

is the total workload (sum of all interaction rates) at time t involving cell i. The total geo-spread for all the servers at time t is

$$\begin{aligned} \mathsf{spread}^{(t)} &= \sum_{s \in \mathcal{S}} \mathsf{spread}^{(t)}(s) && (4.2)\\ &= \sum_{i \in \mathcal{C}: f_i = s} d_{ig_s} w_i^{(t)} && (4.3)\\ &= \sum_{i \in \mathcal{C}: f_i = s} d_{ig_s} \sum_{j \in C} w_{ij}^{(t)}. && (4.4) \end{aligned}$$

To make the server assignment geo-compact, we want to minimize the total geo-spread over time period $[1, T]$

$$\mathsf{spread} = \sum_{t=1}^{T} \mathsf{spread}^{(t)}. \tag{4.5}$$

Our geo-aware server assignment with time varying workload is the following extension of Problem 2.3.1.

Problem 4.1.1. *Given $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, compute partitions f and z such that when applied to a future time-varying workload W_T the total backhaul cost and the total geo-*

spread over the time period $[1, T]$ *are both minimal; i.e.,*

$$\min \left\{ \mathsf{cost}, \mathsf{spread} \right\}, \tag{4.6}$$

provided that W_T *is consistent with* $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$.

Recall

$$\boldsymbol{\mu} = \left(\mu_{ij} \right)_{N \times N}, \ \boldsymbol{\sigma}^2 = \left(\sigma_{ij}^2 \right)_{N \times N}$$

where μ_{ij} and σ_{ij}^2 are the mean and variance of the time-varying interaction rate between cell i and cell j. We say that a time-varying workload W_T is *consistent* with $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ if μ_{ij} and σ_{ij}^2 are (reasonable) sample mean and variance of $w_{ij}^{(t)}$ over time $t \in [1, T]$.

Problem 4.1.1 is a multi-objective optimization problem. Without the geo-spread objective, the problem is already computationally hard as previously discussed in Chapter 2.

4.2 Time-Invariant Workload

We propose a solution for a simplified version of Problem 4.1.1 where it is assumed that the workload does not change over the time, i.e., $w_{ij}^{(t)} = \mu_{ij}$ for all $t \in [1, T]$ and $\sigma_{ij} = 0$ for all $i, j \in C$. Hence, we can omit the superscript $^{(t)}$ in all the formulas. Furthermore, without loss of generality, assume that $\sum_{i=1}^{N} \sum_{j=1}^{N} \mu_{ij}$=1 and that $\kappa < 1$ (the capacity of a server should not be larger than the total workload).

In the proposed solution, we look for one that does not leave any cell unassigned: $f_i > 0$ for every $i \in \mathcal{C}$. Hence, the "unassigned" cost in Eq. (2.1) is $\mathsf{cost}_{unassigned} =$

0. The formulation for the backhaul cost becomes (simplified from Eq. (2.4)):

$$\mathsf{cost}_{crossserver} = \sum_{i,j:0<f_i<f_j} w_{ij} \tag{4.7}$$

$$\mathsf{cost}_{overload} = \sum_{s} \max\left(0, \sum_{i,j:j\geq i \wedge f_i = f_j = s} w_{ij} - \kappa\right) \tag{4.8}$$

$$\mathsf{cost} = \mathsf{cost}_{crossserver} + \mathsf{cost}_{overload} \tag{4.9}$$

We propose a heuristic approach consisting of three phases: focus on geo-spread optimization first, then refine the solution based on the cost, which as a side effect may worsen the spread, and, finally, improve the solution again, this time for a better spread. As local search is widely used for hard combinatorial optimization problems, we present below the local search methods to optimize each individual objective and how to apply them in the three-phase approach.

4.2.1 Cost Optimization

Because cost does not involve geography, we need compute only the best cell-server assignment based on the workload demand; any location choice for the servers would work. In a nutshell, our algorithm starts with a random assignment (random cell-server assignment and random server-location assignment), and repeatedly apply a local operation such that the new assignment improves cost. A local operation, denoted by move_cells(l_0, l_1), runs an algorithm to migrate cells between a pair of servers at locations l_0 and l_1; the servers are referred to as the l_0-server and l_1-server, respectively. As long as we can find a local operation that improves cost,

$$\mathsf{cost}_{new} < \mathsf{cost}, \tag{4.10}$$

we make the new assignment permanent and repeat the same process until no such local operation is found.

Let $A_0(A_0')$ and $A_1(A_1')$ denote the cellsets of the l_0-server and l_1-server before (after) the location, respectively. According to Eq. (4.9), cost will decrease if the quantity

$$\Delta_{\mathsf{cost}}(A_0, A_1) = \sum_{i \in \mathcal{A}_0} \sum_{j \in \mathcal{A}_1} w_{ij} + \max\left(0, \sum_{i \leq j \in \mathcal{A}_0} w_{ij} - \kappa\right) + \max\left(0, \sum_{i \leq j \in \mathcal{A}_1} w_{ij} - \kappa\right) \tag{4.11}$$

decreases as a result of replacing (A_0, A_1) with (A_0', A_1'). Consequently, we should design the cell-moving algorithm such that its objective is to minimize Δ_{cost}.

This challenge can be translated into a graph bipartitioning problem:

- Let H be a weighted graph (self-loop possible) where each cell in C is a vertex and an edge connects cell i and cell j if they have transactions; the weight of edge (i, j) is w_{ij}. A feasible solution is a partition of H into two components. The first additive term of Eq. (4.11) is the cut weight of this partition and the second and third additive terms represent a capacity-constrained quality for the partition.

We derive an algorithm to compute the best partition based on the Fiduccia-Mattheyses (FM) heuristic [FM82]. FM is effective for solving the classic graph min-cut bipartitioning problem whose objective is to minimize the cut weight while balancing the vertex weight. FM is fast (linear time in terms of the number of vertices) and simple (each local operation involves moving only one vertex across the cut). Because our objective is different (minimizing Δ_{cost}), we need to modify FM.

The cell-moving algorithm works as follows (see Algorithm 4.2.1). Resembling FM, the algorithm runs in passes:

- In each pass, compute a sequence of cell migrations each moving a vertex from A_0 to A_1 or from A_1 to A_0 such that $\Delta_{\mathsf{cost}}(A_0, A_1)$ after this series is maximally improved.
- Stop when no improvement can be made.

To determine which vertex to move, let us define for each vertex a quantity called "gain", which is the cost reduction if the vertex were moved from its component to the other component. Consider a vertex i and, without loss of generality, suppose that $i \in A_0$. If vertex i were moved to A_1, its gain in the cut weight is

$$gain_{cut}(i) = \sum_{j \in A_1} w_{ij} - \sum_{j \in A_0, j \neq i} w_{ij}.$$

The edge weight sum of A_0 and that of A_1 would be changed to

$$W'(A_0) = W(A_0) - \sum_{j \in A_0} w_{ij}$$
$$W'(A_1) = W(A_1) + w_{ii} + \sum_{j \in A_1} w_{ij}$$

and so the gain in the capacity-constrained quality is

$$\begin{aligned} gain_{cap}(i) = {} & \max(0, W(A_0) - \kappa) + \max(0, W(A_1) - \kappa) \\ & - \max(0, W'(A_0) - \kappa) - \max(0, W'(A_1) - \kappa). \end{aligned}$$

The gain of vertex i is

$$gain(i) = gain_{cut}(i) + gain_{cap}(i).$$

Intuitively, a positive (negative) gain would result in a smaller (larger) Δ_{cost} if the vertex switched its component.

At the beginning of each pass, we construct a priority queue of vertices based on their gain. This queue includes only those vertices whose migration would result in

$$\max(W'(A_0), W'(A_1)) \leq \max(\kappa, W(A_0), W(A_1)). \tag{4.12}$$

In other words, we consider moving a vertex only if the moving improves the load balancing between the two servers or keeps them under the server capacity.

During the current pass, we repeatedly select the vertex of highest gain from the priority queue, move it, and update the queue. After this vertex is moved, it is "locked" so that it cannot be moved again in the current pass. Then, we repeat this process until the queue is empty. We keep track of the gain accumulation after each k^{th} step:

$$GAIN_k = \sum_{t=1}^{k} gain(i_t),$$

where i_t is the vertex chosen in step $1 \leq t \leq k$. The best move decision would be to move vertices $i_1, i_2, ..., i_{k^*}$ such that $k^* = \arg\max_{k \leq N} GAIN_k$.

If $GAIN_{k^*} > 0$, these moves would result in better Δ_{cost} because the new Δ_{cost} is $\Delta_{\mathsf{cost}} - GAIN_{k^*} < \Delta_{\mathsf{cost}}$; we make these moves permanent and go on to the next pass which repeats the same procedure. Else, the algorithm makes no change (i.e., keep the same partition as that before the pass starts) and stops.

When a local operation $\mathsf{move\_cells}(l_0, l_1)$ finishes, we will use the final assignment resulted from this operation. The algorithm continues repeatedly with finding the next $\mathsf{move\_cells}()$ local operation that can further improve cost and stops when no such local operation is found.

4.2.2 Geo-Spread Optimization

Minimizing spread can be reduced to solving a k-median problem [KH79]. In k-median, given a set of n clients and a set of m facilities, the goal is to choose k facilities to open and assign an open facility to each client such that the total assignment cost is minimum, assuming that the assignment cost to service client i by facility l is γ_{il} (by default, a metric). We can consider each server a facility to open ($k = M$, $m = L$) and each cell a client ($n = N$). The cost to assign client i to facility l (if open) is $\gamma_{il} = w_i d_{il}$ (alternatively, we can think of d_{il} as the assignment cost per unit of service and w_i as the service demand).

Let binary variable z_{il} represent that $z_{il} = 1$ iff cell i is assigned to a server and this server is at location l. Then the total service cost for the corresponding k-median problem is

$$\sum_{l \in \mathcal{L}:\ open} \sum_{i \in \mathcal{C}} z_{il}\gamma_{il} = \sum_{l \in \mathcal{L}} \sum_{i \in \mathcal{C}} z_{il} d_{il} w_i = \mathsf{spread}.$$

Therefore, any k-median solution z_{il} ($z_{il} = 1$ iff facility l is open and client i is served by this facility) is a solution that minimizes spread for our problem.

K-median is NP-hard [KH79] and the best approximation factor known to date is $2.675 + \epsilon$, achieved by Byrka et al. [BPR17]. Using the local search approach, one can obtain an approximation factor of $(3 + 2/p)$, for example by Arya et al.'s polynomial-time algorithm [AGK01]. This algorithm starts with a feasible assignment and repeatedly perform a p-facility swap until no further cost reduction.

Similarly, our algorithm starts with a random assignment and then repeatedly applies a series of local operations. Let assign_cells(L) denote an algorithm that assigns the cells in $\mathcal{C}$ to the servers located at a given subset of locations, $L \subset \mathcal{L}$,

such that a cell is always assigned to the nearest server; i.e.,

$$\mathsf{assign\_cells}(L) : i \mapsto \arg\min_{l \in L} d_{il}.$$

A local operation, denoted by $\mathsf{swap\_locations}(l_0, l_1)$, involves a pair of a server location l_0 in the current server location set L and a non-server location $l_1 \notin L$, and does the following:

- Remove location l_0 from the server set
- Add location l_1 as a new server location set
- Run $\mathsf{assign\_cells}(L')$ to obtain a new cell-server assignment where $L' = L - \{l_0\} + \{l_1\}$ is the new server set.

A local operation is chosen to take place permanently if spread of the resultant assignment is improved by at least a constant factor $\lambda \in (0, 1)$; i.e.,

$$\mathsf{spread}_{new} < (1 - \lambda) \cdot \mathsf{spread}. \tag{4.13}$$

Subsequently, the algorithm goes on repeatedly with finding another local operation satisfying this inequality until none is found.

4.2.3 Cost and Geo-Spread Optimization

The above algorithms are designed for only one objective, cost or spread. We propose the following three-phase algorithm; a summary is given in Algorithm 4.2.2.

In Phase 1 (lines 1-4 of Algorithm 4.2.2), we run the spread-only algorithm presented above to obtain an assignment with the (approximately) best spread.

In Phase 2 (lines 5-7 of Algorithm 4.2.2), we start with this assignment and adjust the cell-server assignment to improve cost. During the process, the server-location assignment is intact. For the adjustment, we apply the same local search algorithm (same local operation) as in the cost-only algorithm except for one small modification. Specifically, a local operation, $\mathsf{move\_cells}(l_0, l_1)$, is made permanent not only if the resultant cost is less (Eq. (4.10), but also the resulted spread remains below a threshold,

$$\mathsf{spread}_{new} \leq (1+\epsilon) \cdot \mathsf{spread}_0, \tag{4.14}$$

Because a local operation, while lessening the cost may worsen the spread, the threshold is introduced to keep the spread within a reasonable factor of spread_0 that is the spread at the start of Phase 2. Here, $\epsilon \in (0, \infty)$; we can set $\epsilon = \infty$ if the goal is to bring down the cost aggressively.

In Phase 3 (lines 8-10 of Algorithm 4.2.2), we start with the assignment of Phase 2 and recompute the server locations for better spread (which has been worsen as tradeoff during Phase 2 compared to that in Phase 1). During the process, the cell-server assignment is intact. Denote the server set by $\mathcal{S}$ and the cellset of each server $s \in \mathcal{S}$ by $cellset(s)$. The unknown to compute is the binary variable y_{sl}, set to 1 iff server s is placed at location l. We have

$$\mathsf{spread} = \sum_{l \in \mathcal{L}} \sum_{i \in \mathcal{C}} z_{il} d_{il} w_i = \sum_{s \in \mathcal{S}} \sum_{l \in \mathcal{L}} y_{sl} \underbrace{\sum_{i \in cellset(s)} d_{il} w_i}_{A_{sl}}. \tag{4.15}$$

Because a server must be assigned to an exclusive location, to minimize

$$\sum_{s \in \mathcal{S}} \sum_{l \in \mathcal{L}} y_{sl} A_{sl}$$

is equivalent to finding a min-cost maximal matching in a complete bipartite graph $(\mathcal{S}, \mathcal{L})$ where an edge connects a vertex $s \in \mathcal{S}$ to a vertex $l \in \mathcal{L}$ with cost A_{sl}

(which is known from the intact cell-server assignment). Therefore, we apply the Hungarian Algorithm [KY55] to compute this matching (y_{sl}), which runs in polynomial time (cubic in the number of vertices).

4.2.4 Evaluation

We conducted an evaluation with both synthetic and real-world datasets. The goal of this evaluation is to assess the effectiveness of the proposed three-phase algorithm compared to intuitive solutions.

4.2.4.1 Setup

The synthetic dataset (Synthetic500) represents a workload that has no relationship with geography. The real-world dataset (Milano625) represents a workload in which demand is higher between cells of increasing proximity.

- Synthetic500: The service area is a 2D square area $\mathcal{A} = [0, 1]^2$ where $N = 500$ random locations are chosen for the base-stations and their corresponding cells are the Voronoi cells of $\mathcal{A}$. The workload demand w_{ij} between cell i and cell j is generated uniformly at random: $w_{ij} \sim uniform(0, 1)$.

- Milano625: We constructed this dataset from the collection of geo-referenced Call Detail Records over the city of Milan during Nov 1st, 2013 - Jan 1st, 2014 (https://dandelion.eu/). Specifically, we partition the area into a grid of $25 \times 25 = 625$ cells of size $0.94km \times 0.94km$, and count the calls between these cells made during the Monday of Nov 4th, 2013.

In both studies, the transaction demand values are normalized such that they sum to 1. The number of MEC servers is set to $M = 10$ whose location is chosen

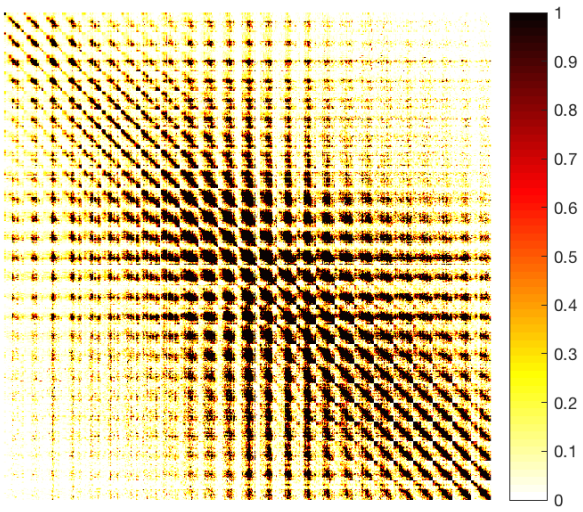


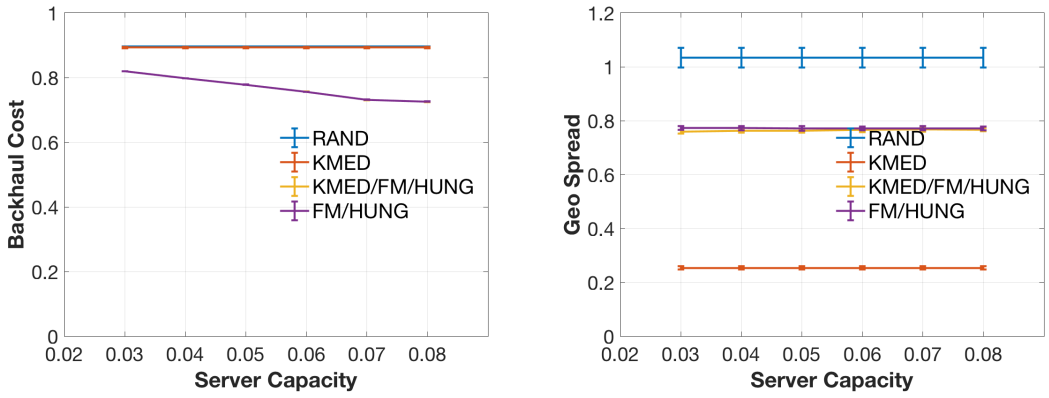
Figure 4.1: Heat map of workload w_{ij} for Milano625 , where i is the x-axis, j the y-axis, and $(0, 0)$ at top-left.

from $L = 50$ random locations. These quantities are reasonable given the number of cells. The server capacity is set to $\kappa \in \{0.03, 0.04, ..., 0.08\}$, meaning {3%, 4%..., 8%} of the total workload. Figure 4.2.4.1 shows the heat map of the workload demand for the Milano625 dataset.

For convenience, we refer to our three-phase algorithm as KMED/FM/HUNG, as it applies k-median, Fiduccia-Mattheyses (FM), and Hungarian algorithms in the three phases, respectively. Serving as benchmark for comparison are:

- RAND: the random assignment algorithm.
- KMED: the spread-only algorithm using k-median.
- FM/HUNG: the cost-only algorithm using FM with another step using Hungarian to improve spread.

Note that the only difference between KMED/FM/HUNG and FM/HUNG is that



(a) (b)

Figure 4.2: Synthetic500: synthetic workload, no relationship with geography

the former starts with a KMED assignment while the latter starts with a random assignment. The metrics for comparison are cost (cost) and spread (spread). RAND offers a good upper-bound for both cost and spread, while KMED represents a good lower bound (supposedly best) for spread and FM/HUNG a good lower-bound (supposedly best) for cost.

The λ parameter in Eq. (4.13) is set to 0.0001 for k-median and ϵ in Eq. (4.14) is set to ∞ (no spread constraint in the second phase). The simulation runs on 10 random sets of candidate server locations and, for each set, 5 random choices for the initial assignment. The results are averaged over these 50 runs and plotted with 100% confidence interval.

4.2.4.2 Workload Without Geography Correlation

Figure 4.2 shows the results for the synthetic case in which geography is no factor in workload demand. In terms of cost (Figure 4.2(a)), KMED is almost identical to RAND, both incurring a high backhaul cost of almost 0.9 (i.e., 90% of the total

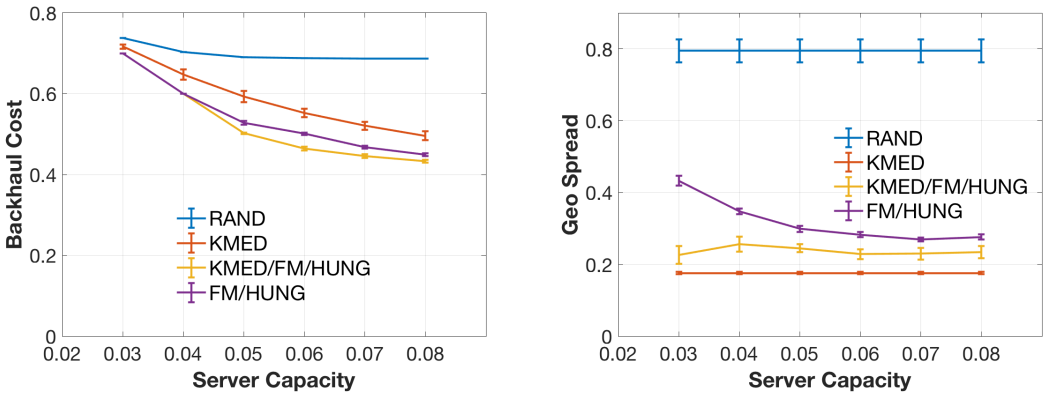
workload), even when the server capacity increases. This is not surprising because KMED is cost-blind and so when workload has no relationship with geography, minimizing spread results in a cost as bad as that of a random assignment. Perhaps for the same reason, the other two methods, KMED/FM/HUNG and FM/HUNG, also incur almost the same cost. In other words, whether we start with a RAND assignment or a KMED assignment, an application of FM+HUNG would result in similar costs. It is important to note that the FM step is effective, especially as the server capacity increases. With a server capacity of 0.08, applying FM reduces the backhaul cost to 0.73, a 20% improvement from the initial assignment.

In terms of spread (Figure 4.2(b)), KMED is the best (expected) and RAND the worst (understandable because it is spread-blind). Between the other two, more interestingly, FM/HUNG has a similar spread (only slightly larger) compared to that of KMED/FM/HUNG. This study suggests that, for a workload input that has no relationship with geography, (1) we can do better than a random assignment, (2) the 3-phase algorithm can run without Phase 1 (KMED) which has almost zero benefit, and (3) there is no clear winner between FM/HUNG and KMED; which one should be chosen depends on whether we prefer minimizing cost or spread.

4.2.4.3 Workload With Geography Correlation

Figure 4.3 shows the results for the real-world dataset (Milano625), in which the workload has a strong correlation with geography; specifically, higher between cells of shorter distance [BC17]. Similar to the above study, and, expectedly, RAND is worse than all the other algorithms in both objectives and KMED has the best spread. There are, however, key differences.

First, KMED has a substantially lower cost than RAND's; this implies that, due



(a) (b)

Figure 4.3: Milano625: real-world workload, strongly correlated to geography.

to the correlation between workload and geography, by minimizing spread there is, to some extent, a benefit in reducing the cost. Indeed, because KMED tends to cluster cells near each other and workload demand is high between close cells, heavy workloads tend to be served by the edge, hence less workload going backhaul (compared to a random assignment). Second, KMED/FM/HUNG is clearly better than FM/HUNG in both objectives; this substantiates the effectiveness of having Phase 1 (KMED) in our algorithm, leading to not only better cost but also better spread. Third, KMED/FM/HUNG has a spread only slightly worse than KMED; this shows the effectiveness of Phase 3 (HUNG) in improving spread. In short, all the three phases in the proposed algorithm (KMED/FM/HUNG) are important to achieving both objectives.

4.2.4.4 Other Observations

Figure 4.4 gives a visual representation of the assignment map according to KMED and KMED/FM/HUNG. Both methods are consistent with the physical map of

Milan (Figure 5.3); that is, because most transactions involve the inner neighborhoods, a server closer to the the central area covers fewer cells (which have high activity) than those in the outskirt (which have low activity). While KMED places the servers spatially nicely (Figure 4.4(a), Figure 4.4(b)), KMED/FM/HUNG allows for some dis-contiguity in the cells that belong to the same server (Figure 4.4(c), Figure 4.4(d)); the latter does so to reduce the backhaul cost. For example, when the capacity $W = 0.05$, (cost, spread) is (0.43, 0.21) for KMED/FM/HUNG and (0.49, 0.18) for KMED. This is a tradeoff between cost versus spread. Although we cannot avoid this tradeoff, it is important to point out that KMED/FM/HUNG offers a better workload balance, as clearly illustrated in Figure 4.4(e). The ratio of the maximum to the minimum workload demand is at least two times less with KMED/FM/HUNG than with KMED.

4.3 Time-Varying Workload

For the problem in which the workload changes over the time, we can apply the following approximation heuristics: set the interaction rate $w_{ij}^{(t)}$ at each time t to the average (mean) interaction rate μ_{ij}, replace the objectives with the objectives that use μ_{ij} instead of $w_{ij}^{(t)}$, and then solve the corresponding problem using the three-phase algorithm above for time-invariant workload.

Another way is to extend the Probabilistic Partitioning algorithm (Algorithm 5.2.2) proposed in Chapter 3. Recall that, in the while loop of step 3 in this algorithm, when looking for an edge (x, y) connecting vertex x and vertex y so that we will merge these vertices to form a super vertex, the criterion to find this edge is such that

$$\mathsf{cdf}(\kappa|\mu_{x\cup y}, \sigma_{x\cup y}) \geq \theta.$$

We revise this step by introducing an additional condition that vertices x and y must be geographically contiguous.

4.4 Summary

We have addressed a server assignment problem for MEC, whose decision making needs to be made for where geographically to place the servers and how to assign them to the user cells based on interaction workloads. The formulation of the two objectives with respect to the backhaul cost and geographic spread has not appeared in the MEC literature. We have proposed and evaluated a heuristic solution leveraging k-median, Fiduccia-Mattheyses, and Hungarian methods. The solution is not optimal (due to the NP-hardness of the problem), but an effective approximation. This solution assumes a fixed workload, but it could be extended for time-varying workload. For the future work, our next step is to investigate the effectiveness of this extension. We will also compare it with the approach that extends the probabilistic algorithm framework presented in the previous chapter to accommodate geo-contiguity.

Algorithm 4.2.1: move_cells(l_0, l_1)

A_0, A_1: cellsets of l_0-server and l_1-server, respectively;

Compute $W(A_0), W(A_1)$: edge weight sum (compute load) of A_0 and A_1, respectively;

while *true* **do**

  $k^* = 0$, $GAIN_{max} = 0$, $GAIN = 0$;

  Unlock all vertices;

  Compute gain for every vertex;

  Save the current partition $C = A_0 \cup A_1$;

  for $k = 1, 2, ..., |C|$ **do**

    if $\exists$ *an unlocked vertex* $i_k \in C$ *of highest gain satisfying InEq. (4.12)* **then**

      $GAIN \mathrel{+}= gain(i_k)$;

      if $GAIN > GAIN_{max}$ **then**

        $GAIN_{max} = GAIN$;

        $k^* = k$;

      Move vertex i_k to the other component;

      Update gain for every unlocked vertex;

      Update $W(A_0)$ and $W(A_1)$;

      Lock vertex i_k;

    else break ;

  if $GAIN_{max} > 0$ **then**

    Retrieve the original current partition $C = A_0 \cup A_1$;

    Move vertices $i_1, i_2, ..., i_{k^*}$ each from its respective original component to the other component;

    Update gain for every unlocked vertex;

    Update $W(A_0)$ and $W(A_1)$;

  else break ;

return;

Algorithm 4.2.2: Three-Phase Algorithm

`/* Phase 1: K-median to minimize spread */`

Initial assignment: Choose a set L of M random locations for the servers and assign cells to these servers randomly;

while $\exists\ l_0 \in L$ *and* $\exists\ l_1 \notin L$ *such that* swap_locations(l_0, l_1) *would result in an assignment satisfying InEq. (4.13)* **do**

  Permanently apply the assignment resulted from swap_locations(l_0, l_1);

`/* Phase 2: FM to improve cost */`

while $\exists\ l_0, l_1 \in L$ *such that* move_cells(l_0, l_1) *would result in an assignment satisfying InEq. (4.10) and InEq. (4.14)* **do**

  Permanently apply the assignment resulted from move_cells(l_0, l_1) ;

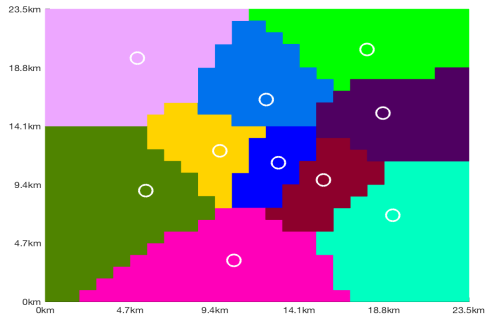
`/* Phase 3: Hungarian to improve spread */`

Let $\mathcal{S}$ be the set of servers (i.e., those serving cells according to the above assignment);

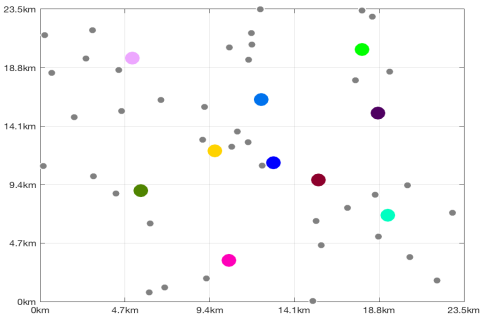
Compute matrix $[A_{sl}]_{\mathcal{S}\times\mathcal{L}}$ as defined in Eq. (4.15) ;

Run Hungarian Algorithm on the cost matrix $[A_{sl}]_{\mathcal{S}\times\mathcal{L}}$;

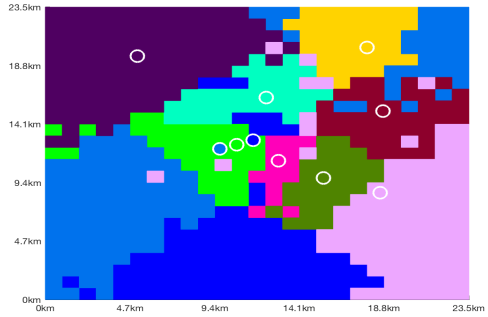
return;



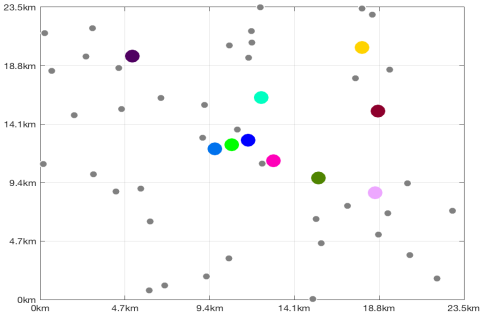
(a) KMED



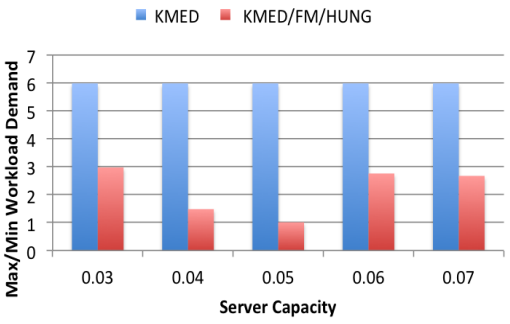
(b) KMED



(c) KMED/FM/HUNG



(d) KMED/FM/HUNG



(e) KMED/FM/HUNG

(f) Milano area

Figure 4.4: Assignment map of (a,b) KMED and (c,d) KMED/FM/HUNG for a single random run on Milano625 with capacity $W = 5\%$, where each colored-circle represents a server (out of 50 possible locations) and each server and its cells share the same color. Also shown is (e) the ratio of maximum to minimum workload and (f) the physical map of Milan.

CHAPTER 5

GEO-CONTIGUOUS PROBABILISTIC PARTITIONING

In chapter 3 we presented our solution of Problem 2.3.1 without consideration of geographical constraints. In chapter 4 we achieved the geo-awareness objective by formulating it as geographical compactness optimization problem. Now we want to solve both problem at once with our new algorithm.

5.1 Optimization Objective and Constraint

Ideally, we want a partition f such that when applied to a (future) time-varying workload W_T the total backhaul cost is minimal. If we know *nothing* about W_T, we can do no better than the random assignment, which assigns each cell to an arbitrary server. On the other hand, if we assume to know *everything* about W_T, i.e., the interaction rates $w_{ij}^{(t)}$ for all cell pairs (i, j) at every time t are known, this assumption may be too strong to be practical for time-varying W_T.

In this paper, for each cell pair (i, j), instead of assuming that $w_{ij}^{(t)}$ is known for all t, we assume to know only their mean μ_{ij} and variance σ_{ij}^2. This is a realistic assumption because we can easily measure these quantities from samples (past interaction rates). In practice, the interaction demand between two cells may change frequently over the time, but the change should follow a pattern, and so

the sample mean and sample variance are reliable.

Let

$$\boldsymbol{\mu} = \left(\mu_{ij}\right)_{|\mathcal{C}|\times|\mathcal{C}|},\ \boldsymbol{\sigma}^2 = \left(\sigma_{ij}^2\right)_{|\mathcal{C}|\times|\mathcal{C}|}.$$

We say that a time-varying workload W_T is *consistent* with $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$ if μ_{ij} and σ_{ij}^2 are (reasonable) sample mean and variance of $w_{ij}^{(t)}$ over time $t \in [1, T]$.

A constraint for the partition is that the cells in the same cluster must form a contiguous geographical region. We call such a partition geo(graphically)-contiguous. To set up our optimization, therefore, we need to input

Our optimization problem is as follows.

Problem 5.1.1. *Given $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, compute a geo-contiguous partition f such that when applied to a future time-varying workload W_T the total backhaul cost over the time period $[1, T]$ is minimal; i.e.,*

$$\min\left\{\mathsf{cost} = \sum_{t=1}^{T} \mathsf{cost}^{(t)}\right\}, \tag{5.1}$$

provided that W_T is consistent with $(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$.

5.2 Heuristic Solution

The variables $\{f_i\}$ represent a partition of the cells into $(|\mathcal{S}| + 1)$ groups, $|\mathcal{S}|$ of which are assigned to the $|\mathcal{S}|$ servers and the last is the group of unassigned cells. A group of cells is called a "supercell". We use letters x, y, ... to denote supercells.

We propose an iterative partitioning algorithm based on the idea of modularity optimization in complex networks [New06]: repeatedly merge the vertices of the network, represented as a weighted graph, to obtain a smaller graph of super-vertices, called "modules", that is better in terms of "modularity". The modularity

is some quality function such that, conventionally, networks with high modularity have dense connections between the nodes within modules but sparse connections between nodes in different modules.

In our context, the vertices of the original graph are the original cells, which are merged into super-vertices (supercells) in the subsequent graphs. At any time, each supercell formed must correspond to a geographically contiguous region. The key ingredient in our algorithm is the definition of the quality function. For ease of explanation, we focus first on the special case where the workload is time-invariant (zero variance), and then the general case where the workload is time-varying (positive variance).

5.2.1 Time-Invariant Workload

Suppose that the workload does not change over the time, i.e., $w_{ij}^{(t)} = \mu_{ij}$ for all $t \in [1, T]$ and $\sigma_{ij} = 0$ for all $i, j \in C$. Let u_{xy} denote the workload demand between the users in a supercell x and the users in a different supercell y and u_x denote the workload demand between users in the same supercell x. We can write

$$u_{xy} = \sum_{i \in x} \sum_{j \in y} \mu_{ij}, \;\; u_x = \frac{1}{2} \sum_{i \in x} \sum_{j \in x} \mu_{ij}.$$

Here, "$i \in x$" means that cell i is a member of supercell x.

The optimization problem becomes a graph partitioning problem with fixed edge weights. To apply modularity optimization, we start with a graph where each vertex is an original cell and after each merging step we obtain an immediate graph of supercells; here, the weight of the edge connecting supercell x and supercell y is u_{xy}. The vertex merging goes on as long as we can increase u_x for some vertex x. Constrained by the capacity bound and geo-contiguity, a merger to form a supercell x is allowed only if the new $u_x \leq \kappa$ (capacity bound) and x forms a

contiguous region of cells (geo-contiguity). The detailed steps of this algorithm are as follows.

Algorithm 5.2.1 (Time-Invariant Workload)**.**

1. *Initially, we start with a graph of $|\mathcal{C}|$ supercells (vertices), each supercell being an original cell: $\mathcal{X} = \{\{1\}, \{2\}, ..., \{|\mathcal{C}|\}\}$.*

2. *While (true)*

   - *Maintain a list of edges $(x, y) \in \mathcal{X} \times \mathcal{X}$ sorted in descending order of u_{xy}.*
   - *Find the first edge (x, y) in this list such that (i) the geographical regions corresponding to x and y are adjacent and (ii) that $u_{x \cup y} \leq \kappa$, i.e., the workload demand of the supercell merging x and y would be under the server capacity. If this edge is not found, go to Step 3.*
   - *Merge vertex x and vertex y to form a new vertex corresponding to supercell $x \cup y$.*
   - *Update $\mathcal{X}$ and its edge weights.*

3. *At this time, we obtain a graph of m vertices (supercells), corresponding to a partition into m supercells.*

4. *If $m \geq |\mathcal{S}|$, we choose the $|\mathcal{S}|$ supercells x of largest u_x and assign them respectively to the $|\mathcal{S}|$ servers. The cells in the remaining supercells are unassigned.*

5. *Else, the partition has fewer supercells than servers. Each supercell is assigned to a server. In this case, there is no unassigned cell and one or more servers may be idle (suggesting that m servers would be sufficient).*

In understanding of this algorithm, intuitively, we can think of u_{xy} as the inter-module density and u_x as the intra-module density, and if we merge x and y we would improve modularity because of the reduction in inter-module density and increase in intra-module density. Translating this to our cost model, by enforcing the condition $u_x \leq \kappa$, we keep $\mathsf{cost}_{overload} = 0$ (Eq. (2.3)) and after each vertex merging we obtain reductions in $\mathsf{cost}_{unassigned}$ (Eq. (2.1)) and $\mathsf{cost}_{crossserver}$ (Eq. (2.2)). By preferring the edge (x, y) with largest u_{xy} in Step 2, we want to maximize the modularity improvement.

It is noted that this algorithm is essentially the same as that proposed by Bouet and Conan in [BC17]).

5.2.2 Time-Varying Workload

In the general case, we consider a time-varying workload and the only information we assume given is the sample mean and variance of the interaction rate for each cell pair. Our approach is to approximate the workload by a Gaussian distribution. Specifically, we model the interaction rate $w_{ij}^{(t)}$ of each cell pair (i, j), which is unknown, as a Gaussian random variable with mean μ_{ij} and variation σ_{ij}^2,

$$w_{ij}^{(t)} \sim \mathcal{N}\Big(\mu_{ij}, \sigma_{ij}^2\Big).$$

The Gaussian distribution has the property that the sum of Gaussian variables is also Gaussian. Consequently, the workload demand between supercell x and supercell y at a time t is the following Gaussian variable

$$u_{xy}^{(t)} \sim \mathcal{N}\Big(\underbrace{\sum_{i\in x}\sum_{j\in y}\mu_{ij}}_{denoted\ by\ \mu_{xy}}, \underbrace{\sum_{i\in x}\sum_{j\in y}\sigma_{ij}^2}_{denoted\ by\ \sigma_{xy}^2} \Big).$$

Similarly, the workload demand of a supercell x is the following Gaussian variable

$$u_x^{(t)} := \frac{1}{2}u_{xx}^{(t)} \sim \mathcal{N}\Bigg(\underbrace{\frac{1}{2}\sum_{i\in x}\sum_{j\in x}\mu_{ij}}_{denoted\ by\ \mu_x}, \underbrace{\frac{1}{2}\sum_{i\in x}\sum_{j\in x}\sigma_{ij}^2}_{denoted\ by\ \sigma_x^2}\Bigg).$$

We propose an assignment algorithm similar to Algorithm 5.2.1 for the time-invariant workload case, except for the merging criterion in the iterative merging step (Step 2 in Algorithm 5.2.1). Specifically, two supercells x and y are chosen to merge if the would-be resultant supercell $x \cup y$ has a high probability, at least a given threshold θ, of having a workload demand below the server capacity κ.

For a Gaussian variable $X \sim \mathcal{N}(\mu, \sigma^2)$, its cumulative distribution function at κ is

$$\mathsf{cdf}(\kappa|\mu,\sigma) = \mathsf{Prob}[X \le \kappa] = \frac{1}{2}\left[1 + \mathsf{erf}\left(\frac{\kappa-\mu}{\sigma\sqrt{2}}\right)\right]$$

where erf is the Gaussian error function

$$\mathsf{erf}(x) = \frac{2}{\pi}\int_0^x e^{-t^2}dt.$$

Therefore, we merge supercells x and y into a new supercell $x \cup y$ if

$$\mathsf{cdf}(\kappa|\mu_{x\cup y}, \sigma_{x\cup y}) \ge \theta.$$

Note that the workload demand of the new supercell $x \cup y$ follows a Gaussian distribution with mean $\mu_{x\cup y}$ and variance $\sigma^2_{x\cup y}$, which can easily be computed :

$$\mu_{x\cup y} = \mu_x + \mu_y + \mu_{xy}$$
$$\sigma^2_{x\cup y} = \sigma_x^2 + \sigma_y^2 + \sigma_{xy}^2.$$

Following this merging criterion, the final partition result depends on the choice of threshold θ. In the extreme case $\theta = 0$, the server capacity is ignored and the algorithm produces one big supercell merging all the original cells. In the other

extreme case $\theta = 1$, no cells are merged and so the algorithm results in a partition that is the set of the original cells. In general, a larger/smaller θ tends to result in a partition with more/fewer supercells. Therefore, it is critical to choose a good value for θ that should result in the best partition for the given number of servers, $|\mathcal{S}|$.

Because we do not know this "optimal" θ value in advance, our proposed algorithm starts with the largest $\theta = 1$ and gradually decrements it as long as the resultant partition has more supercells than servers. The merging continues with this new θ. The algorithm stops when the number of supercells reaches $|\mathcal{S}|$ or smaller. The proposed algorithm works in detail as follows.

Algorithm 5.2.2 (Time-Varying Workload)**.**

1. *Initially, we start with a graph of $|\mathcal{C}|$ supercells (vertices), each supercell being an original cell:* $\mathcal{X} = \{\{1\}, \{2\}, ..., \{|\mathcal{C}|\}\}$.
2. *Initialize* $\theta = 1$.
3. *While (true)*
   - *Maintain a list of edges* $(x, y) \in \mathcal{X} \times \mathcal{X}$ *sorted in descending order of* μ_{xy}.
   - *Find the first edge* (x, y) *in this list such that (i) the geographical regions corresponding to* x *and* y *are adjacent and (ii) the following condition is met*
   $$\mathsf{cdf}(\kappa|\mu_{x\cup y}, \sigma_{x\cup y}) \geq \theta$$
   - *If this edge is not found, go to Step 3. Else, merge vertex* x *and vertex* y *to form a new vertex corresponding to supercell* $x \cup y$.

- *Update $\mathcal{X}$ and its edge weights.*

4. *At this time, we obtain a graph of m vertices (supercells), corresponding to a partition into m supercells.*

5. *If $m > |\mathcal{S}|$*

   - *Set $\theta = \theta - \epsilon$ (step size ϵ, sufficiently small, e.g., 0.05 in our evaluation).*
   - *Go back to Step 2 to repeat the merging procedure.*

6. *Else, the partition now has the same or fewer supercells than servers. Each supercell is assigned to a server. In this case, there is no unassigned cell and one or more servers may be idle (suggesting that m servers would be sufficient).*

5.3 Evaluation

We conducted an evaluation[1] on two real-world datasets we had constructed from the collection of geo-referenced Call Detail Records over the city of Milan during November 2013 and December 2013 (https://dandelion.eu/).

5.3.1 Setup

The datasets correspond to two different days (Fri Nov 1 and Tue Nov 5), each representing a time-varying workload for evaluation. They offer two very different workload scenarios. Specifically, we partitioned the area, 23.5km $\times$ 23.5km, into a grid of $|\mathcal{C}| = 20 \times 20 = 400$ cells and counted calls between every two cells for each 10-minute period during an entire day. The time duration of the entire

[1] http://github.com/quynhtavo/probabilistic-partitioning

workload is therefore $T = 24$ hours / 10 minutes $= 144$ time slots. In each dataset, we compute the mean and variance representing the interaction rate of a cell pair (i, j) as follows:

$$\mu_{ij} = \frac{1}{T}\sum_{t=1}^{T} w_{ij}^{(t)},\ \sigma_{ij}^2 = \frac{1}{T-1}\sum_{t=1}^{T}(w_{ij}^{(t)} - \mu_{ij})^2.$$

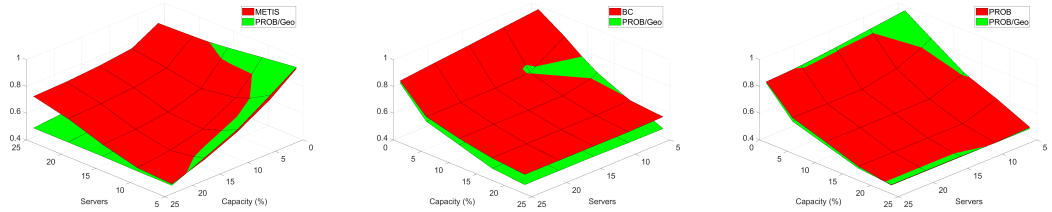
Note that these μ_{ij} and σ_{ij}^2 values are the only information we assume known in our research problem.

We investigated different configurations by varying the number of MEC servers in the range $|\mathcal{S}| \in \{5, 10, 15, 20, 25\}$ and the server capacity in the range $\kappa \in \{1\%, 5\%, 10\%, 15\%, 20\%\}$. For example, a choice of $\kappa = 10\%$ means that a server cannot process more than 10% of the average total workload (i.e., the total workload at an average time instant, which is the ratio of total workload over $[0, T]$ to the duration T).

For ease of presentation, we refer to our proposed algorithm (Algorithm 5.2.2) as PROB/Geo ("probabilistic" +"geo-contiguous"). We compared it to three benchmarks:

- METIS: this is arguably the best min-cut graph partitioning tool [KK98]. To use it for our evaluation, we take as input a graph of cells where the edge weight between cell i and cell j is set to their interaction mean μ_{ij} and the weight of a cell is set to 1. METIS can then produce a partition of balanced-size clusters minimizing cross-cluster interaction workload. Although METIS does not satisfy the geo-contiguity constraint, it serves as a reasonable good-case scenario in terms of the backhaul cost, for the sake of comparison.

- BC (by **B**ouet and **C**onan [BC17]): this is the closest work to ours, which is the same as the algorithm for a time-invariant workload (Algorithm 5.2.1).



(a) PROB/Geo vs. METIS (b) PROB/Geo vs. BC (c) PROB/Geo vs. PROB

Figure 5.1: Friday Dataset: Comparison of PROB/Geo to the benchmarks. The z-axis is the backhaul cost.

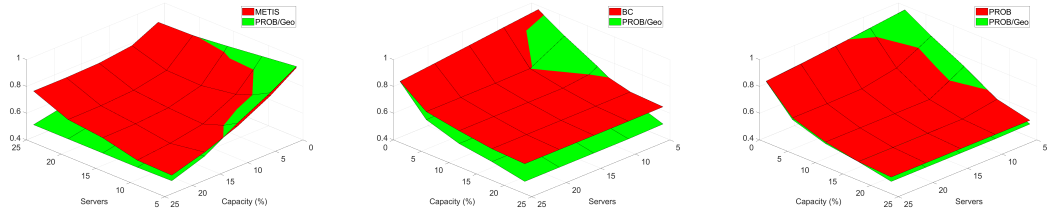
To use it as benchmark for our evaluation, we take as input the workload where the interaction rates are the mean μ_{ij}'s and apply BC to obtain the assignment.

- PROB: this is the same algorithm as PROB/Geo but without the geo-contiguity constraint when iteratively merging the cells to form a new supercell. Our goal to include PROB in the evaluation is to (i) confirm the need for this constraint to produce a geo-contiguous partition and (ii) assess how much backhaul cost PROB/Geo has to sacrifice for geo-contiguity compared to PROB.

The main metric for comparison is the backhaul cost, normalized as the ratio of cost in Eq. (2.5) to the total workload over the time. For example, a ratio of 0.8 indicates that the backhaul cost of the MEC solution is only 80% that of the solution without MEC; hence a 20% cost saving by MEC.

5.3.2 Results

As a preview of our in-depth analysis, Figure 5.1 (for the Friday dataset) and Figure 5.2 (for the Tuesday dataset) provide 3D illustrations of the comparison



(a) PROB/Geo vs. METIS (b) PROB/Geo vs. BC (c) PROB/Geo vs. PROB

Figure 5.2: Tuesday Dataset: Comparison of PROB/Geo to the benchmarks. The z-axis is the backhaul cost.

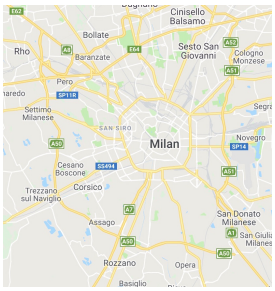
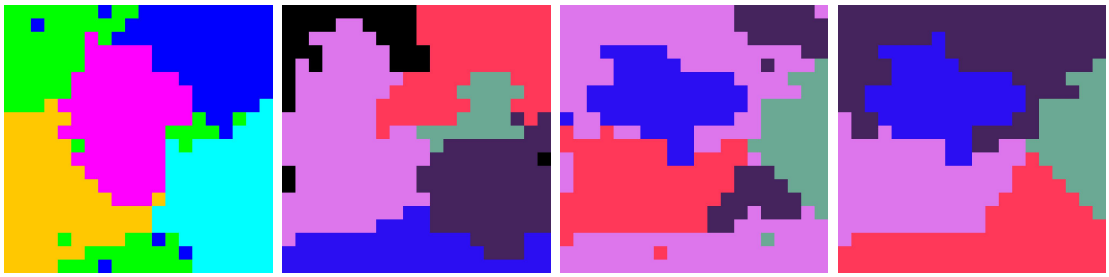


Figure 5.3: The area in the city of Milano for which the datasets were obtained.

of for all configurations of the evaluation. Although these datasets represent very different workloads, they share the following pattern: PROB/Geo incurs less backhaul cost than all the benchmarks in most cases. It is worse in the extreme cases where server capacity is too small and the number of servers is too small. The result is encouraging because it is shown here that applying PROB/Geo is better than applying a generic graph partitioning tool (METIS is known to be effective generally) for the problem in our paper.

Figure 5.4 provides a sample visualization of how the cell-to-server assignment of each algorithm is laid out on the geographical map (see Figure 5.3). In this sample, using the Friday dataset with five servers deployed and 20% capacity,

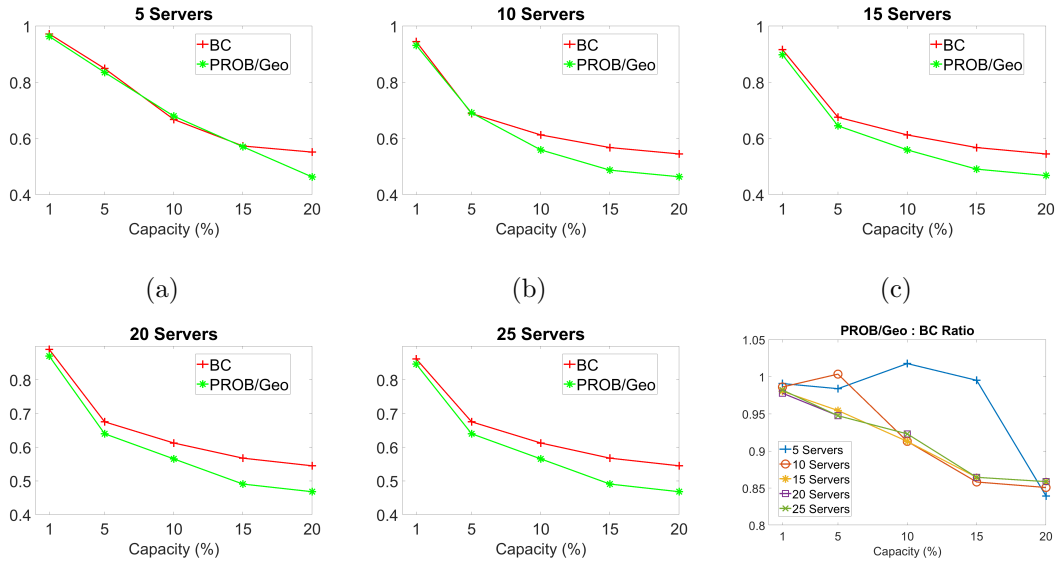


(a) METIS (b) BC (c) PROB (d) PROB/Geo

Figure 5.4: The layout of the cell-to-server assignment resulted from each algorithm. These maps are for five servers and 20% capacity using the Friday dataset. Black color represents those cells that are unassigned.

PROB/Geo results in a partition of five supercells and no unassigned cells (Figure 5.4(d)). The partition of METIS, expectedly, does not satisfy the geo-contiguity requirement (the blue and green cells in Figure 5.4(a)). So is the partition of PROB, albeit slightly (Figure 5.4(c)). This means that the geo-contiguity criterion on the merging step of our proposed algorithm, PROB/Geo, is necessary to satisfy the geo-contiguity constraint. The partition of BC is geo-contiguous. However, it consists of six supercells, but one supercell is unassigned (the black cells on the map). In this configuration, the backhaul cost of PROB/Geo is only 83% of BC (see Figure 5.5(f)).

Below, we provide more details of the evaluation. We pay more attention to comparing PROB/Geo to BC because the latter is the only compatible method that guarantees geo-contiguity, the constraint in our partitioning problem.



(a) (b) (c)

(d) (e) (f)

Figure 5.5: Friday Dataset: Effect of the server capacity.

5.3.2.1 Effects of the Server Capacity and the Number of Servers

Figure 5.5 demonstrates for the Friday dataset the effect on the backhaul cost as the server capacity increases using the same number of servers. With only five servers (Figure 5.5(a)), there is no significant difference between PROB/Geo and BC until the server capacity is sufficiently large (at 20%). With more servers, PROB/Geo becomes more quickly better than BC as the capacity increases. For example, with 15 servers (Figure 5.5(c)), PROB/Geo is already better than BC when the server capacity reaches 5%. Figure 5.5(f) gives the ratio of the backhaul cost of PROB/Geo to that of BC. Unless the number of servers is too small (five servers), the proposed partitioning algorithm is clearly the winner. Also, adding more capacity helps reduce the backhaul cost.

Figure 5.6 (for the Friday dataset) demonstrates the effect on the backhaul

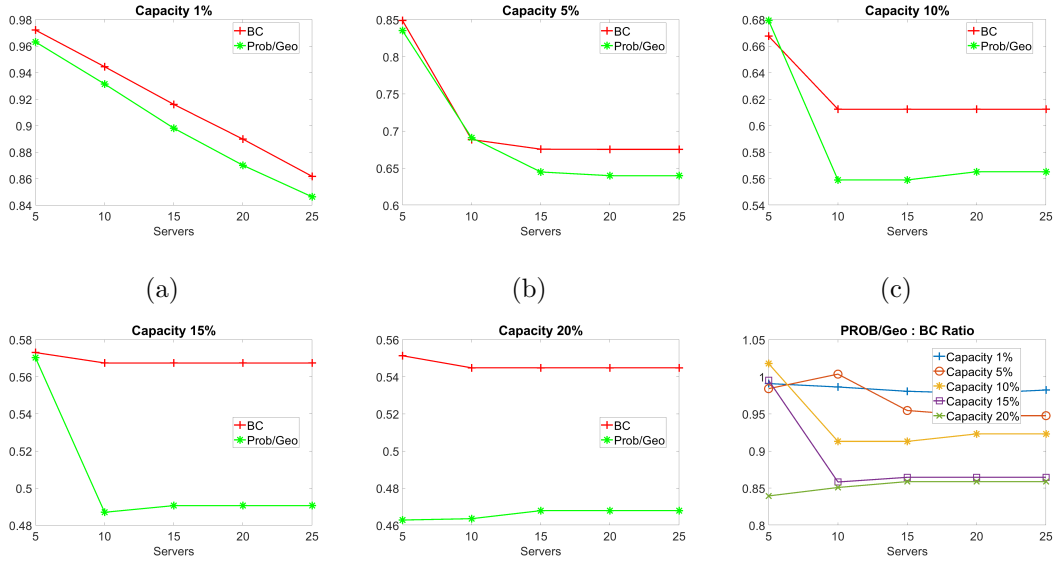
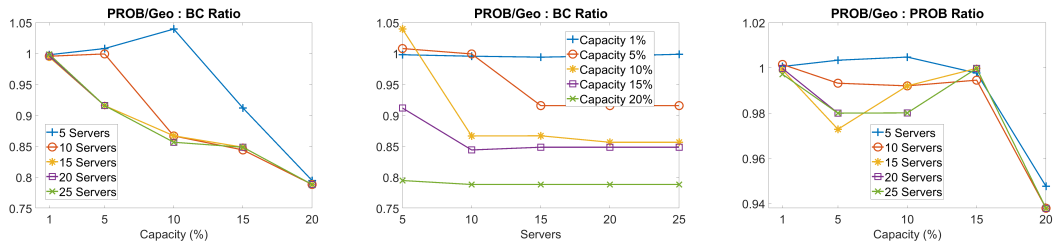


Figure 5.6: Friday Dataset: Effect of the number of servers.

cost as we add more servers, provided the server capacity is unchanged. When the capacity is extremely low (1%, Figure 5.6(a)), the cost does improve, which is understandable because a server has too little capacity to make any edge computing benefit and so having more servers will help. However, when the capacity approaches a realistic value, it is observed that having more servers beyond a sufficient quantity does not help at all. For example, we need only 10 servers for 10% capacity (Figure 5.6(c)) or 5 servers for 20% capacity (Figure 5.6(e)). This is because with more servers the intra-server cost is decreased but the cross-server cost can increase.

Figure 5.6(f) plots the ratio of the cost of PROB/Geo to that of BC as the number of servers increases for different cases of capacities. Similarly observed, increasing the number of servers beyond a certain threshold (10 servers) does not improve this ratio, meaning if we want to make PROB/Geo increasingly better



(a) (b) (c)

Figure 5.7: Tuesday Dataset: Effect of the server capacity.

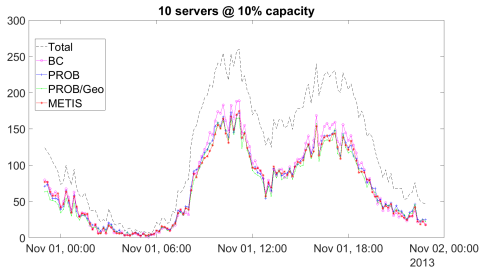
than BC we should not achieve that by adding more servers.

We observe similar patterns for the Tuesday dataset, as seen in Figure 5.7. In this figure, we also include the comparison between PROB/Geo and PROB (which relaxes the geo-contiguity constraint). When there are more than 5 servers or the capacity reaches 20%, although PROB/Geo has to satisfy this constraint, it still is more cost effective than PROB (Figure 5.7(c).

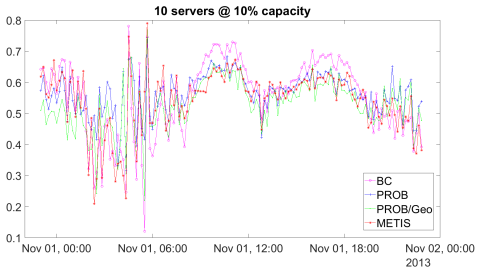
This study together with the study on the effect of the number of servers implies that it is the server capacity, not the number of servers, that has more influence on the efficiency of the server assignment. To lower the backhaul cost, it is better to increase the server capacity for each server than to add more servers of the same capacity.

5.3.2.2 Analysis of Cost in Real Time

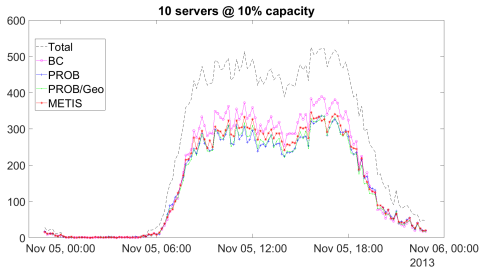
In this study, we investigate the backhaul cost incurred when processing the workload at each time instant $t \in [0, T = 144]$. As we mentioned earlier, the Friday and Tuesday datasets represent two very different workloads. The former is for a day with busy workloads only during the rush hours (Figure 5.8(a)), while the latter is for a weekday with busy workloads during the working hours (Figure 5.8(c)).



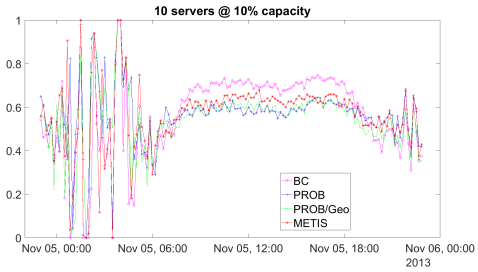
(a) Friday: absolute cost



(b) Friday: normalized cost



(c) Tuesday: absolute cost



(d) Tuesday: normalized cost

Figure 5.8: Real-time cost for the case of 10 servers at 10% capacity, shown for both Friday and Tuesday. The left plots show the absolute value of the backhaul cost incurred by each algorithm together with the total workload at each time instant. The right plots show the backhaul cost normalized as a fraction of the total.

Also, there are some activities during the hours before and after midnight entering the Friday, while there are no such activities for the Tuesday.

For the Friday dataset, Figure 5.8(a) shows the absolute backhaul cost over the time and Figure 5.8(b) shows the backhaul cost normalized as a percentage of the total workload at the given time. Similarly, we have Figure 5.8(c) and Figure 5.8(d) for the Tuesday dataset.

As seen in Figure 5.8(b) and Figure 5.8(d), the normalized backhaul cost fluctuate wildly during light workload hours. This is understandable because when

the workload is too light and the normalized cost is computed as a ratio to the workload, the value range of this ratio is wide. The more important observation here, however, is that of PROB/Geo is the most robust during busy hours. For example, this applies to the Friday dataset around noon and 6PM (Figure 5.8(b)). The pattern is also noticeable for the Tuesday dataset (Figure 5.8(d)), showing approximately a 50% cost saving by PROB/Geo versus a 50% saving by BC and 60% by METIS.

This study further substantiates the cost efficiency of our proposed algorithm and supports our hypothesis that the mean and variance information can be very useful to finding a robust partition for highly time-varying workloads.

5.4 Summary

We have addressed a novel server assignment problem for MEC for serving time-varying workloads. Instead of requiring known interaction rates between the cells, which is the case for existing techniques but unrealistic for many practical applications, we only assume to know their mean and variance estimates. Also, we aim to produce a geo-contiguous assignment, meaning that all the cells assigned to a server must form a geographical contiguous region. We have proposed an effective probabilistic algorithm to produce a geo-contiguous partition of the cells to assign to the servers that is robust to the workload change dynamics. The idea is to approximate the interaction rate for each cell pair as a random Gaussian variable. Our evaluation has substantiated the effectiveness of the proposed algorithm for some real-world datasets, demonstrating its superiority over earlier work and typical benchmark.

CHAPTER 6

CONCLUSION

As mobile devices have become ubiquitous, traffic at the edge of the network is growing faster than ever. To improve user experience, commodity servers are deployed in the edge to form a distributed network of mini datacenters. A consequential task is to partition the user cells into groups, each to be served by an edge server, to maximize the offloading to the edge.

Ideally, given a number of edge servers with limited capacity, a partition of the user cells into clusters (to assign to the edge servers respectively) should achieve there key goals:

- Cost-Efficiency: The total backhaul cost is minimal. This cost is incurred for processing requests not fulfilled by the edge servers due to capacity limitation;
- Geo-Awareness: The cells in each cluster should be close to each other. This makes the partition robust to user mobility and, also, results in the assigned server being geographically close to its users for efficient operation and shortened latency;
- Uncertainty-Robustness: The partition should remain efficient in the presence of time-varying workloads and the dynamics that a server's effective capacity may go up and down. Otherwise we would need to recompute the partition frequently to cope with these uncertainties.

The pertinent literature commonly assumes that the workload of each cell be known, or quantified by a value representing its computing demand over time. A common approach is to solve a graph partitioning problem, where a graph is used to represent the workload and the goal of the partition is to satisfy a given objective, for example, minimum cut. This approach is effective for a fixed workload. This dissertation is focused on finding a solution that satisfies all the three key goals above. In the following, I review the key contributions made, suggest future directions, and offer final remarks.

6.1 Summary of Contributions

What makes the research especially interesting is with the third goal: robustness. Indeed, while cost-efficiency is understandably a must-achieve in every MEC solution and geo-awareness has already been integrated in some recent partitioning techniques, we are aware of no prior research aimed at a solution that is geo-aware and at the same time robust to time-varying workloads.

A key contribution in this dissertation is a probabilistic partitioning approach, whose novelty is in the representation of the time-varying workload as a random variable of a probability distribution. Instead of requiring known interaction rates between the cells, which is the case for existing techniques but unrealistic for many practical applications, we only assume to know their mean and variance estimates. We have provided an algorithm for the case that this distribution is Gaussian and evaluated its effectiveness with real-world datasets. Another contribution is a local search based partitioning algorithm to result in a server assignment that is both cost effective and geographically compact. The probabilistic partitioning algorithm, whose main focus is to copy with time-varying workload, can be ex-

tended to accommodate geographical constraints, and the geo-aware partitioning algorithm, whose focus is to achieve geo-compactness in the assignment, can be useful as a starting point to derive a solution robust to time-varying workload.

6.2 Future Work

There are several directions I would like to pursue in the future. I would like to further investigate the probabilistic partitioning framework for the following cases:

- The probability distribution for the pairwise interaction rate may not necessarily be Gaussian; for example, one can be the lognormal distribution. For Gaussian distributions, their sum is also Gaussian. However, this property does not hold true for many other distributions and our algorithm will have to be revised accordingly.

- The server capacity may not be a fixed quantity. In practice, the effective capacity of a server may fluctuate from time to time due to the dynamics of its resource usage (the server may involve in other computing tasks). Therefore, my approach will be to also model the server capacity as a random variable of a probability distribution.

- The workload input in the current research of this dissertation is the interaction rate between two cells. I would like to solve a similar server assignment problem where the workload is due to requests for service initiated by and processed for individual users (devices). This case is plentiful in the real world. To partition this time-varying workload, we can think of it as partitioning of a stochastic graph where the weight of a node is a random variable.

6.3 Final Remarks

The server assignment problem is an important problem in MEC. I believe that this research has filled an important gap. Not only that, from theoretical perspective, the developments in this dissertation can be generalized for other application purposes, not necessarily only for MEC server assignment. For example, the proposed probabilistic partitioning approach offers a novel way to formulate and address the problem of partitioning a dynamic stochastic graph where the graph can be used to represent pairwise interactions that are not necessarily only for MEC. I hope that many of the ideas and findings in this research will be adopted in the future, including being used as benchmark for comparison or starting points for further extensions.

References

[AGK01] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. "Local Search Heuristic for K-median and Facility Location Problems." In *Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, STOC '01, pp. 21–29, New York, NY, USA, 2001. ACM.

[AL12] M. Alicherry and T. V. Lakshman. "Network aware resource allocation in distributed clouds." In *2012 Proceedings IEEE INFOCOM*, pp. 963–971, March 2012.

[AS17] A. Al-Shuwaili and O. Simeone. "Energy-Efficient Resource Allocation for Mobile Edge Computing-Based Augmented Reality Applications." *IEEE Wireless Communications Letters*, **6**(3):398–401, June 2017.

[BC17] Mathieu Bouet and Vania Conan. "Geo-partitioning of MEC Resources." In *Proceedings of the Workshop on Mobile Edge Communications*, MECOMM '17, pp. 43–48, New York, NY, USA, 2017. ACM.

[BG17] Tayebeh Bahreini and Daniel Grosu. "Efficient placement of multi-component applications in edge computing systems." In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing, San Jose / Silicon Valley, SEC 2017, CA, USA, October 12-14, 2017*, pp. 5:1–5:11, 2017.

[BPR17] Jaroslaw Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. "An Improved Approximation for k-Median and Positive Correlation in Budgeted Optimization." *ACM Trans. Algorithms*, **13**(2):23:1–23:31, March 2017.

[CPS17] Alberto Ceselli, Marco Premoli, and Stefano Secci. "Mobile Edge Cloud Network Design Optimization." *IEEE/ACM Trans. Netw.*, **25**(3):1818–1831, June 2017.

[DHY17] Hou Deng, Liusheng Huang, Chenkai Yang, Hongli Xu, and Bing Leng. "Optimizing virtual machine placement in distributed clouds with M/M/1 servers." *Computer Communications*, **102**:107 – 119, 2017.

[ETS14] ETSI. "Mobile-Edge Computing: Introductory Technical White Paper." The European Telecommunications Standards Institute (ETSI), September 2014.

[FM82] C. M. Fiduccia and R. M. Mattheyses. “A Linear-time Heuristic for Improving Network Partitions.” In *Proceedings of the 19th Design Automation Conference*, DAC ’82, pp. 175–181, Piscataway, NJ, USA, 1982. IEEE Press.

[HKL14] F. Hao, M. Kodialam, T. V. Lakshman, and S. Mukherjee. “Online allocation of virtual machines in a distributed cloud.” In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pp. 10–18, April 2014.

[KH79] O. Kariv and S. L. Hakimi. “An Algorithmic Approach to Network Location Problems. II: The p-Medians.” *SIAM Journal on Applied Mathematics*, **37**(3):539–560, 1979.

[KK98] George Karypis and Vipin Kumar. “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs.” *SIAM J. Sci. Comput.*, **20**:359–392, December 1998.

[KY55] H. W. Kuhn and Bryn Yaw. “The Hungarian method for the assignment problem.” *Naval Res. Logist. Quart*, pp. 83–97, 1955.

[Man15] Zoltán Ádám Mann. “Allocation of Virtual Machines in Cloud Data Centers - A Survey of Problem Models and Optimization Algorithms.” *ACM Comput. Surv.*, **48**(1):11:1–11:34, August 2015.

[MSG15] R. Mijumbi, J. Serrat, J. L. Gorricho, J. Rubio-Loyola, and S. Davy. “Server placement and assignment in virtualized radio access networks.” In *2015 11th International Conference on Network and Service Management (CNSM)*, pp. 398–401, Nov 2015.

[MYZ17] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. “A Survey on Mobile Edge Computing: The Communication Perspective.” *IEEE Communications Surveys Tutorials*, **19**(4):2322–2358, Fourthquarter 2017.

[New06] M E Newman. “Modularity and community structure in networks.” *Proc Natl Acad Sci U S A*, **103**(23):8577–8582, June 2006.

[New17] Peter Newman. “IoT Platforms : Examining the Wide Variety of Software That Holds IoT Together.” Source: BI Intelligence Store, January 2017.

[NRS17] S. Nastic, T. Rausch, O. Scekic, S. Dustdar, M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Ristov, and R. Prodan. "A Serverless Real-Time Data Analytics Platform for Edge Computing." *IEEE Internet Computing*, **21**(4):64–71, 2017.

[PGF18] G. Premsankar, B. Ghaddar, M. Di Francesco, and R. Verago. "Efficient placement of edge computing devices for vehicular applications in smart cities." In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, April 2018.

[SBC09] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. "The Case for VM-Based Cloudlets in Mobile Computing." *IEEE Pervasive Computing*, **8**(4):14–23, Oct 2009.

[SHZ17] X. Song, Y. Huang, Q. Zhou, F. Ye, Y. Yang, and X. Li. "Content Centric Peer Data Sharing in Pervasive Edge Computing Environments." In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 287–297, June 2017.

[TV18] Duc A. Tran and Quynh Vo. "A Geo-Aware Server Assignment Problem for Mobile Edge Computing." *International Journal of Parallel, Emergent and Distributed Systems*, pp. 1–16, 2018.

[VT19] Quynh Vo and Duc A. Tran. "Probabilistic Partitioning for Edge Server Assignment with Time-Varying Workload." In *Proceedings of the IEEE International Conference on Computer Communication and Networks (ICCCN 2019)*, Barcelona, Spain, August 2019.

[WZL17] S Wang, M Zafer, and KK Leung. "Online Placement of Multi-Component Applications in Edge Computing Environments." *IEEE ACCESS*, **5**:2514–2533, 2017.

[WZT17] W. Wang, Y. Zhao, M. Tornatore, A. Gupta, J. Zhang, and B. Mukherjee. "Virtual machine placement and workload assignment for mobile edge computing." In *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, pp. 1–6, Sept 2017.

[XLT17] X. Xu, J. Liu, and X. Tao. "Mobile Edge Computing Enhanced Adaptive Bitrate Video Delivery With Joint Cache and Radio Resource Allocation." *IEEE Access*, **5**:16406–16415, 2017.