RCP: A TEMPORAL CLUSTERING ALGORITHM FOR REAL-TIME CONTROLLER PLACEMENT IN SOFTWARE-DEFINED NETWORKS

BY

REZA SOLEYMANIFAR

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Industrial Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2023

Urbana, Illinois

Doctoral Committee:

Professor Carolyn Beck, Chair
Professor Srinivasa Salapaka
Professor Rayadurgam Srikant
Professor Dusan M Stipanovic

Abstract

In this comprehensive study we introduce a family of maximum entropy based clustering algorithms to address the problem of Controller Placement (CP) or equivalently Edge Controller Placement (ECP) [1]. The shared key advantage of our algorithms is utilizing a maximum entropy based framework that in terms of performance translates to avoiding poor locally optimum placements that most competitor ECP algorithms are susceptible to. Controller placement is recognized as one of the most important problems and a significant performance bottleneck in Software Defined Networks (SDN) which is a recent paradigm in telecommunication networks that disentangles data and control planes and brings flexibility and efficiency to the mobile network. SDN networks lie at the core of the fifth generation (5G) wireless systems and beyond and are increasingly being adopted into telecommunication networks over the recent years. CP can be simply stated as where to place and which network nodes to assign to each individual controller such that a desired utility or cost is optimized. The complexity of CP problem can drastically change with mobility of SDN network nodes and due to this observation we offer two classes of algorithms for static and dynamic placement cases.

For static controller placement problem where network nodes and controllers are assumed to be stationary, the algorithms, referred to as ECP-LL and ECP-LB, address the dominant leader-less and leader-based controller placement topologies and have linear computational complexity in terms of network size. Each algorithm tries to place controllers close to edge node clusters and not far away from other controllers to maintain a reasonable balance between synchronization and delay costs. While the ECP problem can be conveniently expressed as a multi-objective mixed integer non-linear program (MINLP), our algorithms outperform the state of art MINLP solver, BARON both in terms of accuracy and speed.

As for the mobile networks, we propose real-time controller placement algorithms RCP, and RCP+ to tackle the Dynamic Controller Placement (DCP) problem. More specifically these are temporal clustering algorithms that provide real-time solutions for DCP and provide adaptability to inherent variability in network components (traffic, locations, etc.) and is based on a control theoretic framework for which we show the solution converges to a near-optimal solution. The key contribution of these algorithms is the real-time aspect of placement of controllers which to our best of knowledge was never addressed prior to this study.

Our algorithms achieve linear $\mathcal{O}(N)$ iteration computational complexity with respect to the number of nodes in the network, N and can update new positions of network controller in real-time, and in accordance with mobility of SDN network nodes. This property allows utilization of an aerial control plane using UAV swarms. We compare our work with a frame-by-frame approach and demonstrate its superiority, both in terms of speed and incurred cost, via simulations using some of the largest public mobility datasets with millions of records gathered over the span of months, containing GPS trajectories of thousands of pedestrians and vehicles in large metropolitan areas like San Francisco, US and Beijing, China. Based on these simulations, RCP and RCP+ can be up to 25 times faster than a conventional frame-by-frame method.

[1] The latter is used in the context of wireless telecommunications networks.

RCP+ can be viewed as the culmination of the contributions of this thesis. Interestingly ECP-LL, and RCP can be formulated as restricted versions of RCP+ algorithm. RCP+ allows for node prioritization, sparse subsampling, node trajectory prediction using an underlying Recurrent Neural Network (RNN), computation parallelization, and codebook expansion, making it a viable choice even for large-scale mobility networks, which we explore in this thesis. We benchmark RCP+ against a number of alternatives, and show that for real sized networks, it outperforms the comparable state of the art methods.

To anyone seeking truth.

Acknowledgments

This project would not have been possible without the support of many people. Many thanks to my advisor, Carolyn Beck, who read my numerous revisions and helped make some sense of the confusion. Also thanks to my committee members, Professor Salapaka, Professor Srikant, and Professor Stipanovic, who offered insight and support. And finally, thanks to my family and friends who endured this long process with me, always offering support and love.

Table of contents

Chapter 1

Introduction

A computer network is a complex arrangement of interconnected computers, which grants downstream users the ability to access the entire network through seamless connectivity. These computers communicate through what is known as networking hardware [1] that mediates transmission data, in both wired and wireless networks. These networking devices operate using a Network Operating System (NOS) that is specialized for a given networking device.

Conventionally the routing process or selection of paths for data packets is performed locally by the network hardware in a distributed fashion, without a global view of the network. Due to the proprietary nature of most NOSs, networking hardware is required to be compatible with other network components, which introduces many difficulties in terms of operation and maintenance of the network, for example, requiring vendor-specific training, vendor lock-in, limitations on scalability, and potential constraints on upgrade options. SDN is a paradigm that transforms the distributed intelligence of the network into a centralized entity called a controller. This paradigm essentially disentangles the data and control planes, fully delegating the routing process to a dedicated device. Softwarization paradigms predate SDNs and were aimed at prioritizing software implementation of network functions. Through the 20th century, telecommunications technology was driven by hardware, with most functions of the network implemented in ad-hoc physical equipment, and in the early 2000's when commercial CPU's became cheaper the softwarization trend became more prevalent. Software Defined Radio (SDR), and softswitches are early examples of this softwarization trend that may be used to replace hardware based functions with software. The data and control plane disentanglement in SDNs is attainable through *controller* software, which implements an SDN protocol like OpenFlow, and is mounted on a computer device. The ensemble of controllers constitute a logically centralized but physically distributed intelligence for the network, as opposed to the both logically and physically distributed controllers of legacy networks. Ensemble controllers appear to applications and policy engines as a single, logical switch. For example, OpenFlow, described in [1], enables network controllers to determine the path of network packets across a network of switches. The controllers are distinct from the switches, which allows for more sophisticated traffic management. OpenFlow automatically allows rerouting of network packets when a switch is down or a system is under maintenance; this is not possible in legacy networks. OpenFlow is layered on top of the Transmission Control Protocol (TCP) and allows switches from different vendors–often each with their own proprietary interfaces and scripting languages–to be managed remotely using a single open protocol bringing immense scalability. Essentially this moves network control

[1] Routers, switches, repeaters, etc.

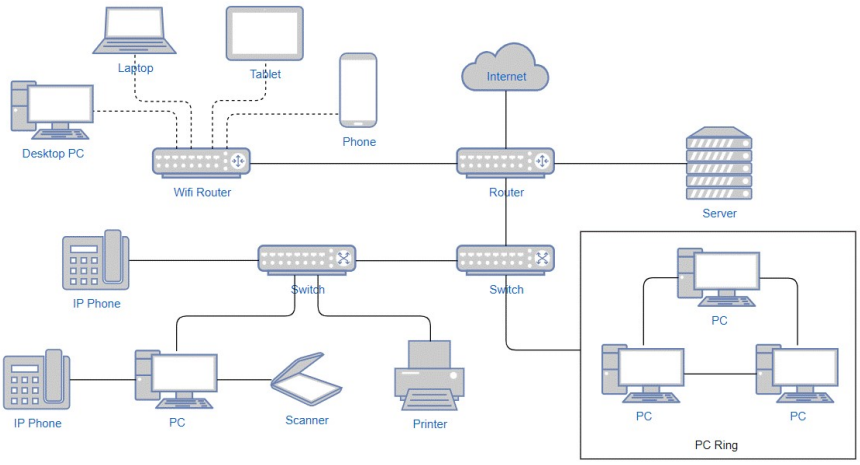


Figure 1.1: A typical computer network with conventional network devices.

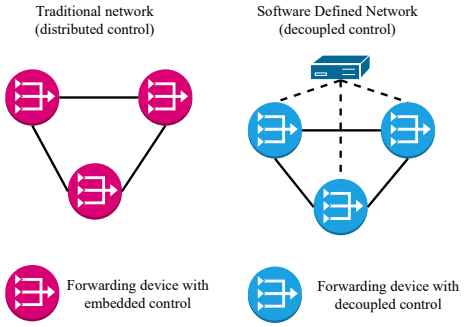


Figure 1.2: Centralized control versus distributed control.

out of proprietary network switches and into open source and locally managed control software, enabling the SDN paradigm, as can be seen in Figure 1.2. Following the introduction of SDNs (approximately 2012), this approach was quickly adopted and incorporated in design of switches by major companies like HP, Google, and Microsoft.

Wireless networks are of high importance in modern telecommunication systems as they are efficient, mobile, responsive, accessible, have enhanced guest access and better support expansion of network. In order to enhance these systems, Software-Defined Networks (SDN) have been introduced as an emerging paradigm whose primary advantage is giving developers greater control over the network traffic and administration [2]. Traditionally wireless networks have played both the role of administration and relay of data within the same infrastructure. One of the limitations of this architecture is that modifying these networks requires manually re-configuring nodes of the network to accommodate the new changes. Softwarization is a new trend in wireless communication networks that helps to automate this type of manual work.

One of the most studied open research problems, on which SDN itself heavily relies, is the so-called edge controller placement problem (ECP) [2]. Controller placement is one of the most important components of software defined networks [4]. This problem was first introduced in [5] and is in general NP-hard (see [6]). Controllers are network nodes which are designated to control other nodes of a network. ECP in a fog/cloud network essentially reduces to determining how many and which nodes in the network need to be

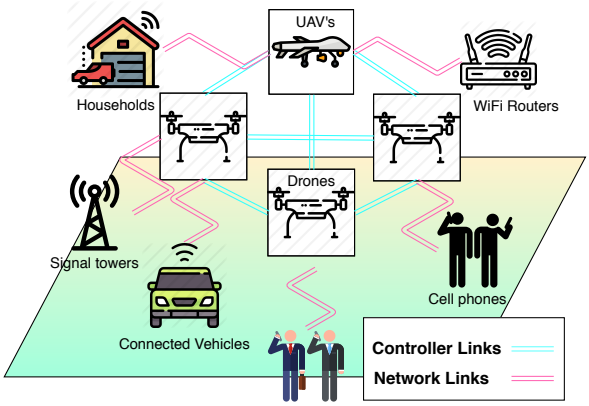


Figure 1.3: Software Defined Network with mobile controller nodes. Image from [3]

designated as the controllers. This placement induces several costs including delays between edge nodes and the controllers they are assigned to, and synchronization delay between the controllers themselves which we refer to as delay and synchronization costs respectively throughout this study.

There are several approaches to address the ECP problem. Our approach is based on viewing this problem in a data clustering sense. Many clustering based approaches in literature are hindered by naive initialization and are thus prone to poor local optima. This leads to multiple optimization attempts with varied initializations that increase total computation time needed to find an optimal placement. These approaches are also restricted to a single objective value which prevents the decision maker from simultaneously considering multiple controller placement criteria. We discuss the use of the deterministic annealing (DA) algorithm, which is tailored to avoid these shortcomings, and introduce algorithms that iteratively minimize the costs associated with ECP. We identify the core competences of our static algorithms as being (1) scalable and fast, due to linear computational complexity in terms of problem size, (2) high quality in terms of near optimal solutions, (3) initialization independent as we always start with one controller in the mass center of data, (4) excellent at avoiding poor local minima due the to use of a Shannon entropy term in the clustering objective function and (5) able to address a multi-objective scheme.

The SDN paradigm has existed for some time for wired networks, but more recently Software Defined Mobile Networks (SDMN) has extended the softwarization paradigm to mobile networks. This paradigm promotes usage of generic and commodity hardware and software and prioritizing software over hardware in implementation of protocol-specific functions. The benefits of SDMN include:

- networks are directly programmable since they are decoupled from the forwarding process;
- upgrades are software-only and hardware is protocol agnostic;
- maintenance can be carried out by a wider range of vendors;
- due to a reliance on general purpose hardware, networks can be scaled up easily by adding more commodity hardware;
- global network visibility leading to minimizing the security breaches.

In recent years, the IT community has been actively discussing SDMN as a principal 5G enabler. The global SDN market value was $9.9 million in 2019, and will appreciate to $72.6 million by 2027 [7], with large

telecommunication companies like AT&T, Vodafone and Rogers already benefiting from it. More recently usage of mobile controllers has been promoted in [8] and [9], and herein we similarly assume the overlay network of controllers is implemented using a UAV swarm. The application potential of UAV swarms is intriguing, and only in its infancy [10]. Do to their high mobility, aerial vehicles are an ideal choice to respond to continuous locale changes of SDMN edge nodes in a real-time fashion. Moreover, deploying this type of SDN controller gives great flexibility to networks in locations where ground stations may not be present [11], [12].

The Controller Placement (CP) problem can be defined as determining the locations of controller devices and the assignment of network nodes to these devices in the SDN network, such that certain criteria are optimized. These criteria can be a cost, such as cumulative network delay, and/or a utility, such as load balance, or a hybrid criterion consisting of multiple cost or utility components. In terms of the Open Systems Interconnection (OSI) model described in [13], the optimization happens at the *physical* layer. This problem was first introduced in [5], and the authors proved this problem, a variant of the celebrated facility location problem, is NP-hard. The CP problem can be viewed as a subclass of the Dynamic Controller Placement (DCP) problem with no dynamicity [2], and in reverse DCP can be seen as a chronologically ordered concatenation of many CP subproblems. This perspective motivates a crude yet effective way to tackle DCP using an existing approach for solving CP, namely viewing each CP subproblem in isolation. We recognize this approach as the *static*, or naive approach, contrary to the *dynamic* approach that uses the temporal relationship between CP subproblems, and prioritizes adaption and prediction over recalculation. The static approach is blind to the evolution of CP subproblems that can be leveraged to make predictions of future states of the network. To the best of our knowledge the static approach has been the de facto approach in the current body of literature, until [14] proposed the first temporal algorithm to tackle DCP. A deeper inspection of the static approach, as we will discuss later, reveals that the static approach is inherently inadequate for the task of Real-time Controller Placement, where smooth transitions between the solutions to subproblems are necessary to guarantee a feasible path for the mobile controllers.

There is a dichotomy in DCP rooted in the source of variability in the network. The dynamicity of the network can be due to either (1) a change in network topology, or (2) the packet flow rate of the network. Our focus is on a specific type of dynamicity that is rooted in the mobility of the network nodes, i.e. the network topology change.

A subclass of DCP which was first introduced by [14] is the so-called *Real-time Controller Placement* [3], emphasizing the sheer speed of placements. The definition of real-time is domain based, which can vary from nano-seconds in finance and high-frequency trading to seconds in log file processor applications. In the context of mobile SDNs we will assume the conventional standard in traditional networks that requires communication between users from end to end with a latency of 20 milliseconds or less [15]. Using this standard, we claim that many of the works in literature, which emphasize real-time aspects of their approaches, are not practically real-time. To see this we show the computational complexity of finding a solution to a CP subproblem is $O(\tau(n)T(n))$ when using any iterative method. Here, $\tau(n)$ is the iteration complexity that depends on the subroutine used for the placement, which could be anything from linear programming, quadratic programming, to clustering and heuristic algorithms, and $T(n)$ is the number of steps required by the approach to reach a solution. RCP, as we will show later runs at $O(n)$, namely $\tau(n) = n$, $T(n) = 1$. Softwarized UAV/Drone systems are a fertile ground for implementation of this type of controller placement

[2]The sources of dynamicity in DCP can be either changes in network topology, packet flow, or a mixture of both.
[3]We refrain from referring to this class of DCP subclass as RCP, to avoid confusion with RCP algorithm.

due to criticality of split-second position updates for aerial vehicles.

Similar to other works in the literature such as [16], RCP adopts a clustering approach to the CP problem, whereby controller placements are matched with the position of the so-called cluster centroids. Clustering is the task of grouping a set of objects in such a way that objects in the same group (cluster) are more similar [4] to each other than to those in other groups [17]. Similarly clustering sequences of unlabeled point sets taken from a common metric space is known as the temporal clustering problem [18] which is applicable to temporally evolving data; this is the approach we adopt in this thesis to address the continuous-time dynamics of network nodes in a typical SDMN. As we will later show, the CP problem under study can be analytically decoupled into placement and assignment subproblems, analogous to a clustering algorithm. Thus a clustering approach is an intuitive way to determine the underlying structure of the problem at hand.

The main distinction between RCP and other algorithms in the literature is RCP's leveraging of the network nodes' trajectory history, which computationally allows removal of the outer loop that exists in iterative optimization procedures for controller placement. Existing placement schemes are agnostic to this temporal information, leading to under-utilization of available information and sub-optimality of performance. These dynamics-agnostic solutions fall under what [19] first called the *frame-by-frame* approach, where at each time interval the system is *frozen* and placements are found as if the problem is static, discarding the previous solution and starting from scratch at each time interval. This inherently inadequate static approach has led to suboptimal methods, yet is common in the DCP literature. Herein we offer a novel, or rather a dynamic, perspective that is more in tandem with the nature of the problem.

To the best of our knowledge, RCP is the only dynamics-aware algorithm in the literature for addressing DCP. Here we offer a linear time algorithm that does not necessarily return the global optimum but yields a locally optimum solution that is seen to be very close to the global optimum. This feat is attainable through using what is known as the Maximum Entropy Principle (MEP), or a closely related optimization concept called convexification, where we start with a well-behaved (convex) but unrelated problem and slowly anneal it back to the original problem, avoiding poor local optima in the process. RCP+ is a more practical replacement for RCP that shows promising results for the real world and large-scale mobility datasets to which we apply it in this study. We identify the contributions of our algorithms for the dynamic setting as (1) yielding high quality solutions using the MEP and convexification techniques, despite the presence of many poor local optima in the objective function; (2) tackling DCP using a temporal approach, whereby SDN network history is recruited to dynamically adapt the solution as opposed to recalculating; (3) computing controller position updates in real-time (20ms or better); (4) trajectory prediction using an auxiliary and lightweight neural network; (5) sparse sampling of the SDN network to improve speed; (6) parallelizing the RCP+ implementation on an Nvidia GPU; and finally (7), we propose to benchmark the RCP+ algorithm against state-of-the-art Dynamic Controller Placement (DCP) algorithms using popular large-scale mobility datasets. These datasets consist of millions of records collected over several months and contain GPS trajectories of thousands of pedestrians and vehicles in large metropolitan areas like San Francisco, US, and Beijing, China.

[4] According to some similarity measure like Euclidean distance as an example.

Chapter 2

Literature Review

In an abstract sense SDN can be seen as an analog of Cloud technology. More precisely Cloud computing is the on-demand availability of computer system resources, especially data storage (Cloud storage) and computing power, without direct active management by the user, which centralizes computing and storage capabilities and provides it to users on demand [20]. Similarly SDN is the on-demand availability of network resources without direct management of network nodes, that centralizes network control and provides this service to switches. Software-Defined Networking brings a paradigm shift to network design, namely an approach to the design of mobile networks where all protocol-specific features are implemented in software. The so-called controller device hosts this software that essentially is an implementation of a SDN protocol like OpenFlow.

Controller placement problem was introduced in 2012 [5] and since then many researchers have focused on this problem. Controller placement problem can accommodate a wide gamut of objectives such as:

- Delay Reduction: Minimize the communication latency between switches and controllers to reduce end-to-end packet transmission delays. This objective is crucial for real-time applications and services that demand low-latency network performance.
- Fault Tolerance: Optimize controller placements to enhance network resilience against failures. By strategically positioning backup controllers, the network can quickly recover from controller failures, ensuring uninterrupted network operations.
- Robustness: Design controller placements that can adapt to changes in network topology and traffic patterns while maintaining optimal performance. Robust controller placement ensures that the network remains efficient and reliable even in dynamic environments.
- Load Balancing: Distribute the control plane traffic evenly across multiple controllers to prevent congestion and bottlenecks. Load-balanced controller placement enhances network scalability and resource utilization.
- Energy Efficiency: Position controllers in a manner that minimizes energy consumption. Energy-efficient controller placement is particularly important in large-scale SDNs to reduce operational costs and environmental impact.
- Security: Optimize controller placements to enhance network security. By strategically locating controllers, the network can better defend against potential attacks and unauthorized access.

- Scalability: Design controller placements that can accommodate network growth and scale to handle increasing traffic and device counts without sacrificing performance.
- Cost Minimization: Place controllers in a way that minimizes infrastructure and operational costs while meeting the network's performance requirements.
- QoS Optimization: Optimize controller placements to ensure that Quality of Service (QoS) requirements, such as bandwidth, latency, and packet loss, are met for different types of network traffic.
- Network Coverage: Ensure that the placement of controllers provides sufficient network coverage to manage and control all switches effectively.

According to [4] controller placement is one of the most important components of software defined networks. [21] is the first to implement the Cuckoo search algorithm for the problem of controller placement in software defined networks. The benchmark their algorithm against a number of other methods under two different network topologies to showcase its superior performance. [22] identifies the main function of Software Defined Networks (SDN) as decoupling the data plane and control plane. They summarize the bulk of the works in controller placement problem into four master categories: latency-oriented, reliability-oriented, cost-based, and multi-objective classes. They also identify controller-placement as one of the hottest topics in SDN.

Focusing on reliability aspects of ECP, authors in [2] address maximizing fault-tolerance aspects of controller placement rather than performance. They show sacrificing latency for reliability is generally not a good trade-off except in special cases. In [23] authors derive the specific position of all network controllers by minimizing a linear function of load balance factor and total flow request cost. [24] studies the wireless controller placement problem using a multi-objective optimization problem and measure the sensitivity of this placement to variant metrics. The authors in [25] model and analyze a realization of the mobile core network as virtualized software instances running in data centers and SDN transport network elements, with respect to time-varying traffic demands. In [26], the authors develop a Quadratic Program (QP) that aims to minimize network switch to controller latency. As network packet flows change and controller overloads occur, they use another QP to perform switch migration to meet the increased load. In [27], the authors introduce the algorithm LiDy+, which has run-time complexity of $\mathcal{O}(n^2)$ (an improvement over predecessors, with run time complexities of $\mathcal{O}(n^2 log n)$), and requires a smaller number of controllers while achieving a higher controller utilization. This method relies on heuristics for placing controller modules and for adjusting the number of controllers needed according to traffic fluctuations.

[28] proposes a network partition based controller placement algorithm based on a mixture of k-means and game theoretic initialization. They benchmark their result on OS3E network topology against vanilla k-means. [29] proposes the Density Based Controller Placement which uses a clustering algorithm to split the network into multiple sub-networks. Their algorithm do not use the iteration-based scheme present in most clustering algorithms like k-means and thus enjoy faster speed.

[30] addresses ECP in the novel context of Software Defined Satellite Networks (SDSN). They address two major categories of ECP in SDN, i.e. dynamic and static. The former assumes we need to decide on switching on or off already existing controllers and the latter assumes we need to place the controllers for the first time in network. [31] also considers ECP in the context of satellite networks and study the use-case scenario of SDN-enabled satellite space segment. They design a Integer Linear Program (ILP) to address this problem. Focusing on reliability aspect of ECP, authors in [2] address maximizing fault-tolerance aspect of controller placement rather than performance. They address fundamental questions like performance-reliability trade-off,

and maximum achievable fault-tolerance for a given SDN. They show that it is generally not a good trade-off to sacrifice latency for reliability except for some special cases. A multi-period approach to controller placement over a finite horizon can be seen in the work of [32]. They use a multi-objective optimization model to derive the multi-period roll-out plan for controller placements. This work is the earliest of its kind in incremental controller placement. They discovered that a cost saving of %70 to %80 can achieved against a latency penalty of %35 to %45 for incremental controller placement.

satellite gateway placement problem is in many ways similar to controller placement problem and is addressed in detail in the work of [33]. They solve this problem using a hybrid, clustering and simulated annealing method and produce near-optimal latency values. They address gateway and controller placement simultaneously to maximize reliability and minimize latency. [34] proposes a novel scheme to minimize measurement overhead. Thy formulate the measurement-aware Distributed Controller Placement (MDCP) as a quadratic integer programming problem. They propose an algorithm with an approximate ratio of 1.61 and show that it can reduce measurement overhead by over %40.

[35] uses the Analytic Hierarchy Process (AHP) to perform multi-criteria controller assignment problem. Apart from latency they also address hop count and link utilization as part of the controller assignment process. They use a hybridized ad-hoc genetic algorithm to solve this controller placement problem. [36] design a multi-objective controller placement scheme that simultaneously address reliability, load balance and low latency. They use the heuristic Adaptive Bacterial Foraging Optimization (ABFO) to solve this problem. Thy show that their algorithm can improve SDN performance and has practical significance on actual networks.

In this thesis we present ECP-LL and ECP-LB as the first maximum entropy based clustering algorithm to address ECP in wireless edge networks. A tutorial on deterministic annealing for the unfamiliar reader may be found in [37]. We distinguish our algorithms from previous clustering approaches in that it is the first multi-objective clustering approach to the ECP problem and it does not require initialization. We found previous algorithms in literature that typically enjoy a fast speed such as Cuckoo search, GA, and other ad-hoc heuristics suffering from susceptibility to poor local optima solutions. On the other hand exact approaches like quadratic integer programming are too slow to be practical for real-case scenarios. Our algorithms address these shortcomings by leveraging their ability to sense and escape poor local minima and at the same time enjoy fast speed due to linear computational complexity in terms of parameters of the problem.

It turns out that the placement and assignment of network nodes to these controllers can have significant impacts on performance of SDNs–network delay being one of the common metrics. In this light DCP came to existence in the context of dynamic SDN systems where network topology or packet flow rate are the two common sources of dynamicity. DCP can be defined as the task of continuously updating the optimal placement and node association of SDN controllers in the network. Historically DCP has been tackled using the aforementioned static approach, until [14] proposed the first temporal approach to DCP that takes into account the dynamics of the network. It has been shown that recruiting this underutilized piece of information can amount to significant computational savings in terms of finding an optimal solution. Table 2.1 summarizes the key works in DCP literature, along with the context of implementation and method used. We acknowledge that there is a fair amount of research that does consider network dynamics but not in the sense of recruiting network history, but more in the line of already being aware of the future and proactively planning for it, as in a deterministic setting. Because this look ahead policy is extended multiple steps in the future it is more prone to errors compared to making predictions for the immediate next step, in the

aforementioned dynamic approach. Because of this, *proactive* methods are only relevant in the context of being aware of flow rates in future; for example a data center with consistent demand patterns.

Paper	Proactive	Context	Objective	Variable	Solution	Real-time
[38]	No	SDVN	Load balancing+latency	Traffic fluctuation	ILP	✗
[26]	No	5G SDN	Latency	Network flows	QP	✗
[31]	Yes	LEO Constellation	Flow setup time	Network flows	ILP	✗
[8]	No	Drone SDN	Link quality	Switch positions	ILP	✗
[39]	No	SDN	Load balancing+latency	Traffic fluctuation	GA	✗
[27]	No	SDN	Maximum latency	Traffic fluctuation	Heuristic	✗
[40]	Yes	SDN	Latency+migration	Traffic fluctuation	SA	✗
[41]	No	ATN	Load balancing	Switch positions	ILP+GA	✗
[42]	No	SDN	Flow setup time	Network flows	ILP	✗
[30]	Yes	SDSN	Load balancing+latency+economic	Network flows	APSO	✗
[43]	No	SDN	Flow setup time	Network flows	ILP	✗
[44]	Yes	Data Centers	Latency+synchronization+migration	Request rate	Game theoretic	✗
[45]	Yes	EON	Latency+controller load	Network flows	heuristic	✗
[46]	No	Edge-core SDN	controller load	Network flows	heuristic	✗
[12]	No	UAV SDN	Latency+synchronization	Network flows	heuristic	✗
[47]	Yes	SDN	Load balancing	Network flows	Clustering	✗
[48]	Yes	SDN	Synchronization+load balancing	Network flows	heuristic	✗
[49]	Yes	SDN	Synchronization+load balancing	Network flows	Deep reinforcement learning	✗
[14]	No	Mobile SDN	Latency+Synchronization	Geo-spatial positions	Temporal clustering	✓
This work	No	SDMN	Latency+Synchronization	Geo-spatial positions	Temporal clustering	✓

Table 2.1: Key research in DCP literature.

They assume the incoming flow at each snapshot is already known and *proactively* solve the DCP problem. They report that this "dynamic" approach outperforms the standard static approach by approximately an average of 20%. In [8], the authors formulate the DCP problem for an SDN enabled drone network as a Mixed Integer Non-linear Program (MINLP), and propose solving this problem using a heuristic that relies on decoupling the placement and assignment tasks.

The authors in [40] cast DCP as a multi-period MINLP with partial information of future traffic flows. The authors consider both operational and migration costs and decompose the problem into smaller online problems, solving them using the Simulated Annealing (SA) algorithm. The authors in [41] design an ILP algorithm to address dynamic controller placement in Aeronautical Telecommunication Networks (ATNs). They propose two heuristic algorithms, DPFA and GA-DPDA, to solve the ILP problem when controller failure happens due to packet flow overload. In [42], the authors introduce an ILP problem that considers both migration time and switch re-assignment time. Each time a new flow profile arrives this ILP is recalculated.

Software Defined Satellite Networking (SDSN) is considered in [30] where authors partition the time horizon into smaller intervals within which the average flow per switch is assumed known. They further assume that back-up controller nodes are placed throughout the network, and by toggling the on-off status of controller nodes they meet the changing network conditions. The solution approach proposed for this problem uses Accelerated Particle Swarm Optimization (APSO). In [43], the authors report up to a 50% improvement over static placement methods using a dynamic controller placement scheme that relies on solving an ILP that re-calculates optimal placements when system change occurs.

In [44], the authors consider the controller assignment aspect of DCP and decompose the problem into a series of stable matching problems with transfers, for which they propose a hierarchical two-phase algorithm that efficiently uses knowledge of future arrival rates. They report a 46% reduction in cost and better load balancing compared to static assignment.

In [31], in which a Low-Earth-Orbit (LEO) constellation setting is considered, the authors partition the time horizon into static snapshots and compute optimal placements and assignments of controllers at each time step using ILP. They assume the incoming flow at each snapshot is already known and *proactively* solve

the DCP problem. They report that this "dynamic" approach outperforms the standard static approach by approximately an average of 20%. In [39], the authors use a multi-objective genetic algorithm (GA) to break SDN networks into domains and sub-graphs assigned to controllers. They use inter-controller latency, load distribution, and controller numbers as the fitness metrics of their GA algorithm. In [27], the authors introduce the algorithm LiDy+, which has run-time complexity of $\mathcal{O}(n^2)$ (an improvement over predecessors, with run time complexities of $\mathcal{O}(n^2 log n)$), and requires a smaller number of controllers while achieving a higher controller utilization. This method relies on heuristics for placing controller modules and for adjusting the number of controllers needed according to traffic fluctuations. The authors in [38] develop an Integer Linear Program (ILP) for a Software Defined Vehicular Network that updates the reallocation of roadside units (RSU) to their corresponding controllers. The dynamics of the network in this work is abstracted as the number of vehicles communicating with a RSU at each time step; here the ILP-based algorithm objective is to minimize a mixed latency and load balancing cost function.

The authors in [25] model and analyze a realization of the mobile core network as virtualized software instances running in data centers and SDN transport network elements, with respect to time-varying traffic demands. In [26], the authors develop a Quadratic Program (QP) that aims to minimize network switch to controller latency. As network packet flows change and controller overloads occur, they use another QP to perform switch migration to meet the increased load.

In [50] authors try to linearize CP into an Mixed Integer Program (MIP) for immediate processing by commercial solvers like CPLEX however because the number of MIP constraints grows linearly with network size, constrained optimization techniques become inherently impractical for real-sized networks. This is due to the fact that typical methods like Branch and Bound have exponential worst case with respect to size of problem [51] , thus the computational complexity can in worst case be exponential with respect to network size.

Tolerable limits to latency for live, real-time processing is a subject of investigation and debate but is estimated to be between 6 and 20 milliseconds [52], [53].

[54] uses an evolutionary game theory approach to DCP and tries to minimize operational cost in terms of power consumption leading to an agile controller placement policy in the context of 5G networks. The SDN in this problem is utilized to the backbone part of the network. [49] take a data-drive approach to DCP considering the delay and flow fluctuations and load balance, they recruit a deep reinforcement learning model to place controllers. The state space in this setting is the flows passing through each switch and action space is a discrete corresponding to association of a switch with a network controller. In [48], the authors point out that flow request nature can be ignored in DCP, and in this light offer a two step method consisting of a greedy set coverage method to find an initial placement and a follow-up game theoretic solution to migrate switches in order to maintain a balance between traffic overhead and controller loads. [55] offer the POCO-PLC toolset that is an all-encompassing system for SDN controller management under dynamic conditions.

[47] offers a bi-level controller architecture where network node migration decisions are dedicated to a certain tier of controllers using the Dynamic Controller Clustering (DCC) algorithm. They report a computational complexity of $\mathcal{O}\left(MN \log N\right)$ with N and M being number of network switches and controllers respectively. [12] Studies DCP in a UAV based communication system were WiFi access point are installed on UAVs to provide internet access to ground nodes. [46] state that static mapping of switches to controller can lead to load imbalance when traffic conditions change. In an edge-core SDN type system they design an algorithm for switch to controller migration to react to controller overload instances.[45] appropriately

describes the CP problem as an optimal matching between switches and controllers. In the context of Elastic Optical Networks (EON) for interconnecting data centers, they proactively optimize a hybrid cost consisting of average latency and controller load. They propose the heuristic DMA that reacts to change in-flow rates of the network.

To summarize, there are two main approaches for the DCP problem in the literature. Either authors assume they proactively know the value of network variables in the future, as in [31], [40], [30], [44], or they rely on what we earlier described as the *frame-by-frame* approach, as in [8], [27] amongst others. Both of these methods fall short of real-world practicality. The former assumes availability of data that is typically not known, and the latter does not exploit the temporal relationship between network states over time. To overcome this gap, our work is aimed at creating a new placement procedure that can work in real-time and exploit given temporal relationships of the system; this has led to the design of the RCP, and RCP+ algorithms.

Chapter 3

Methodology

3.1 Static Setting

The controller placement problem in software-defined networks exhibits an inherent dichotomy that arises from the variability present in the network's constituents. This dichotomy necessitates distinct approaches for two sub-classes of the problem: static and dynamic cases. In the static scenario, the network's topology and traffic patterns remain relatively stable over time. Addressing this case requires a lens focused on optimizing the placement of controllers based on a static snapshot of the network, seeking to minimize latency, maximize efficiency, and ensure smooth data flow. On the other hand, the dynamic case involves networks with constantly changing topologies and fluctuating traffic patterns. In this context, placing controllers in real-time becomes crucial to meet the network's dynamic demands. Solving the static case serves as a fundamental stepping stone towards efficiently placing controllers in the dynamic scenario, enabling adaptability and responsiveness to the network's variability. By approaching each sub-class separately, the controller placement problem can be effectively tackled, ensuring the optimal operation of software-defined networks under different conditions and providing a foundation for intelligent and adaptive network management.

3.1.1 Problem Statement

Wireless networks can be illustrated by a graph as shown in Figure 3.1, in which the vertices are the network nodes and the edges represent the communication between them. One or multiple numbers of these vertices can be designated as a controller, where the optimal number and placement of these controllers depends on how far and how close graph vertices are from each other in terms of communication delay associated with graph edges. ECP reduces to finding this optimal assignment of controllers. In this scheme both the nodes and controllers they are assigned to and the controllers themselves constantly communicate data. This means that a scattered placement of controllers may reduce the delay cost but increase the synchronization cost. On the contrary a more compact placement of edge controllers can reduce the synchronization cost while increasing the delay cost between nodes and the controllers. Two dominant schemes for placement of controllers typically considered are leader-less and leader-based [56]. The distinction between the two is that in the former all pairs of controllers in the network directly communicate with each other while in the latter controllers only communicate with a leader controller.

To cast this problem as a mathematical program we define $\mathcal{N}$ as the set of all edge nodes with $\mathbf{Card}(\mathcal{N}) = N$ and $\mathcal{N}_h$ as the set of edge nodes that can serve as controllers with $\mathcal{N}_h \subseteq \mathcal{N}$. Additionally, $x_i \in \mathbb{R}^d$ such that

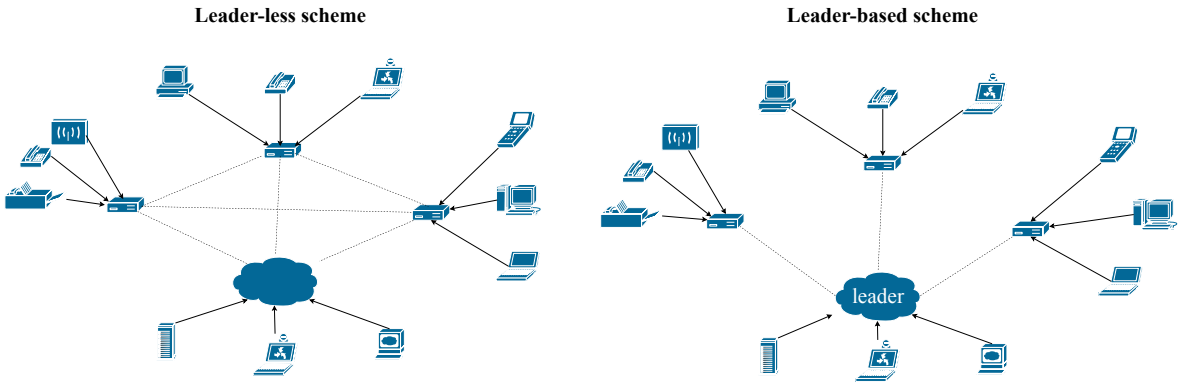


Figure 3.1: Leader-based versus Leader-less edge controller placement scheme

$i \in \mathcal{N}$ determines the position of edge nodes in the wireless network. We use $\mathcal{X} = (\hat{x}_i \in \{0,1\}, i \in \mathcal{N}_h)$ to represent the controller placement policy. If we choose node i to play the role of a controller then $\hat{x}_i = 1$ otherwise $\hat{x}_i = 0$. Similarly $\mathcal{Q} = (q_{ij} \in \{0,1\}, i \in \mathcal{N}, j \in \mathcal{N}_h)$ determines the controller assignment policy where $q_{ij} = 1$ if node i is assigned to controller j otherwise $q_{ij} = 0$. $\mathcal{Z} = (z_j \in \{0,1\}, j \in \mathcal{N}_h)$ determines the leader assignment policy in the leader-based scheme. $z_j = 1$ if controller j is the leader and $z_j = 0$ otherwise. $d_{ij} = d(x_i, x_j)$ encodes the communication delay between nodes i and j which we assume to be proportional to the squared Euclidean distance, i.e. $d_{ij} = \|x_i - x_j\|_2^2$.

Leader-less Case

In this setting all controllers communicate not only with edge nodes but also with each other. Thus we incur a controller synchronization cost between *all* pairs of controllers. We can express the optimal assignment as the solution of the following integer program:

$$\min_{\mathcal{Q},\mathcal{X}} \quad \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_h} q_{ij} d_{ij} + \gamma \sum_{i,j \in \mathcal{N}_h} \hat{x}_i \hat{x}_j d_{ij} \sum_{k \in \mathcal{N}} q_{kj} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{N}_h} q_{ij} = 1 \quad \forall i \in \mathcal{N} \tag{3.2}$$

$$q_{ij} \leq \hat{x}_j \qquad \forall i, j \in \mathcal{N} \tag{3.3}$$

$$\hat{x}_i \in \{0,1\}, \quad i \in \mathcal{N}_h \tag{3.4}$$

$$q_{ij} \in \{0,1\}, \quad i \in \mathcal{N}, j \in \mathcal{N}_h, \tag{3.5}$$

The first term in the objective function corresponds to communication delay across all node-controller pairs. The second term shows the synchronization delay between controllers. Note that synchronization delay also depends on how many nodes are assigned to a certain controller. Constraint (3.2) ensures that each edge node is only assigned to one controller; constraint (3.3) ensures node assignments to a controller are only made to designated controller nodes. Parameter $\gamma \geq 0$ shows the relative importance of controller synchronization delay compared to controller node delay.

Leader-based Case

The leader-based case is similar to the leader-less case except that controllers synchronize only with the leader. We can express the optimal assignment in this setting as the solution to the following integer program:

$$\min_{(\mathcal{Q},\mathcal{X},\mathcal{Z})} \quad \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_h} q_{ij} d_{ij} + \gamma \sum_{i \in \mathcal{N}_h} \sum_{j \in \mathcal{N}_h} \hat{x}_i z_j \left(N d_{ij}\right) \tag{3.6}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{N}_h} q_{ij} = 1 \quad \forall i \in \mathcal{N} \tag{3.7}$$

$$q_{ij} \leq \hat{x}_j \quad \forall i, j \in \mathcal{N} \tag{3.8}$$

$$\sum_{j \in \mathcal{N}_h} z_j = 1 \tag{3.9}$$

$$\hat{x}_i \in \{0,1\}, \quad i \in \mathcal{N}_h \tag{3.10}$$

$$q_{ij} \in \{0,1\}, \quad i \in \mathcal{N}, j \in \mathcal{N}_h. \tag{3.11}$$

Constraint (3.9) ensures that there is always exactly one leader controller in the leader-based setting. Both leader-less and leader-based cases are NP-hard nonlinear combinatorial problems with no guarantees for finding a global optimum solution [6].

3.1.2 Solution

We assume that the delay and synchronization costs are proportional to the squared Euclidean distance between the network nodes. This is not an unreasonable assumption since according to [56] these costs are proportional to if not determined by the Euclidean distances. We further assume that geospatial coordinates[1] of the nodes are provided to us instead of the mutual delays between network nodes.

In the deterministic annealing clustering setting [37], the expected *distortion* [2] can be defined as

$$D = \sum_{i=1}^{N} p(x_i) \sum_{j=1}^{M} p(y_j \mid x_i) D(x_i, y_j).$$

$X = \{x_i\}_{i=1}^{N}$ are the data points and $Y = \{y_j\}_{j=1}^{M}$ are cluster centroids, or edge controller locations, to be determined. $p(y_j \mid x_i)$ is called the association probability[3] of point x_i with centroid y_j and $D(x_i, y_j)$ is the distortion measure which is typically chosen to be the squared Euclidean distance. We interpret $p(x_i)$ as the relative importance given to ith node and assume, if not otherwise indicated that $p(x_i) = \frac{1}{N}$. System entropy can be defined as $H = -\sum_{i=1}^{N} p(x_i) \sum_{j=1}^{M} p(y_j \mid x_i) \log p(y_j \mid x_i)$. We also define the system free energy as $F = D - TH$ where T is the system's so-called temperature.[4] Note that F can be viewed as the Lagrangian for the primary objective of minimizing D, with T being the Lagrange multiplier. The central iteration of DA can be summarized as sequentially optimizing F with respect to the free parameters, i.e. association probabilities and centroid locations.

[1] Here, assumed to be a two dimensional or three dimensional vector representing the location of each edge node.

[2] Distortion is an average weighted distance term, between nodes and centroids, that serves as our basic cost function.

[3] The weighting indicating that a node belongs to a particular centroid. For each node the sum of these associations over all centroids must equal one.

[4] A coefficient scaling the entropy term which indicates how important the entropy term is compared to the distortion term. We typically reduce this coefficient from a high value to a value close to zero.

Leader-less Case

For the purpose of adapting the DA clustering to the leader-less ECP problem we define the distortion measure as $D(x_i, y_j) = d(x_i, y_j) + \gamma \sum_{j'=1}^{M} d(y_j, y_{j'})$. This means the distortion between edge node x_i and controller y_j not only depends on the communication delay between these two nodes but also depends on how far y_j is placed from other controllers denoted as $y_{j'}$.

In order to observe the relation to integer program (3.1)-(3.5) notice we can write total distortion as

$$D = \sum_{i=1}^{N} \sum_{j=1}^{M} p(y_j \mid x_i) d(x_i, y_j) + \gamma \sum_{j'=1}^{M} \sum_{j=1}^{M} \left(d(y_j, y_{j'}) \sum_{i \in \mathcal{N}} p(y_j \mid x_i) \right)$$

This is objective function (3.1) with hard assignments q_{ij} replaced by the soft association probabilities. As noted earlier we define the system's free energy as $F = D - TH$. Setting partial derivatives of the free energy term with respect to association probabilities to zero and solving, yields the solution:

$$p(y_j \mid x_i) = \frac{\exp\left(-\frac{D(x_i, y_j)}{T}\right)}{Z_i}, \qquad Z_i = \sum_{j=1}^{M} \exp\left(-\frac{D(x_i, y_j)}{T}\right)$$

Thus the association probabilities have the celebrated Boltzmann distribution. Similarly, setting the derivatives of F with respect to the centroids y_j to zero leads to the following linear systems of equations:

$$\eta y_j - \gamma \sum_{j' \neq j} y_{j'} = C_j, \quad j = 1, \dots m \tag{3.12}$$

where $\eta = \gamma(m-1) + 1$ and $C_j = \sum_{i \in \mathcal{N}} p(x_i \mid y_j) x_i$. We can compute $p(x_i \mid y_j)$ using Bayes' rule. This gives us a linear system of md variables and md equations with m and d being respectively the number of centroids and the dimensionality of data. It is essential for the convergence of our clustering algorithm that this linear system of equations always has a solution.

Proposition 3.1. *Given the linear system of equations in (3.12) with η and C_j defined as above, if $\gamma \neq \frac{1}{n-m}, \frac{1}{n-2m}$ then there always exists a unique solution $\{y_j\}_{j=1}^{m}$, where the coefficient matrix associated with the system of the equations is non-degenerate with determinant $\left(\frac{(\gamma m+1)^m(\gamma(n-m)-1))}{\gamma(n-2m)-1}\right)^d$.*

Proof. We can write the coefficient matrix associated with (3.12) as the block matrix $\Theta \in \mathbb{R}^{md \times md}$ with diagonal blocks equal to ηI and non-diagonal blocks equal to $-\gamma I$ such that $I \in \mathbb{R}^{d \times d}$.

$$\Theta = \begin{bmatrix} \eta I & -\gamma I & \dots & -\gamma I \\ -\gamma I & \eta I & \dots & -\gamma I \\ \vdots & \vdots & \ddots & \vdots \\ -\gamma I & -\gamma I & \dots & \eta I \end{bmatrix} \tag{3.13}$$

Dividing all rows by constant $-\gamma$ we get $\det(\Theta) = (-\gamma)^{md} \det(\bar{\Theta})$. $\bar{\Theta}$ is a block diagonal matrix with diagonal elements equal to αI and non-diagonal blocks equal to I with $\alpha = -\frac{\eta}{\gamma}$. Using straightforward linear algebra

we can transform $\bar{\Theta}$ to an upper triangular matrix:

$$\bar{\Theta}\times\begin{bmatrix} I & 0 & \dots & 0 \\ \frac{-1}{\alpha+n-2}I & I & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{-1}{\alpha+n-2}I & \frac{-1}{\alpha+n-3}I & \dots & I \end{bmatrix} = \begin{bmatrix} \beta_1 I & \times & \dots & \times \\ 0 & \beta_2 I & \dots & \times \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \beta_m I \end{bmatrix} = \Phi$$

Where $\beta_i = \alpha - \frac{n-i}{\alpha+n-i-1}$ and $\det(\bar{\Theta}) = \det(\Phi) = \prod_{i=1}^{m} (\beta_i)^d$. We can use simple telescoping to further simplify the product to $\left(\frac{(\alpha-1)^m(\alpha+n-1)}{\alpha+n-(m+1)}\right)^d$. This will give $\det\Theta = \left(\frac{(\gamma m+1)^m(\gamma(n-m)-1))}{\gamma(n-2m)-1}\right)^d$ which is well defined for $\gamma \neq \frac{1}{n-m}, \frac{1}{n-2m}$. □

Algorithm 1: ECP-LL

Set max # of clusters K_{max} and min temperature T_{min};
Initialize: $T \to \inf, K = 1, y_1 = \sum_{i\in\mathcal{N}} x_i p(x_i)$;
while *Convergence test* **do**
  Update:

$$p(y_j \mid x_i) \longleftarrow \exp\left(-\frac{d(x_i, y_j) + \gamma \sum_{j'=1}^{M} d(y_j, y_{j'})}{T}\right) / Z_i$$

  Solve:

$$\eta y_j^{new} - \gamma \sum_{j'\neq j^*} y_{j'}^{new} = \sum_{i\in\mathcal{N}} p(x_i \mid y_j) x_i, \quad j = 1, \dots m$$

  Update: $y_j \longleftarrow y_j^{new}$ $j = 1, \dots m$;
  if $T \leq T_{min}$ **then**
    break;
  else
    Cooling Step: $T \longleftarrow \alpha T (\alpha < 1)$;
    Generate small random vector ϵ
    Replace y_j with $y_j + \epsilon$ and $y_j - \epsilon$;
  end
end
Perform last step iteration for $T = 0$;
$y_j \longleftarrow \arg\min_{x_i \in \mathcal{N}_h} d(x_i, y_j)$;

The resulting DA clustering algorithm for the this case is given in Algorithm 1.

For the convergence test we stop at iteration τ if $\|F_\tau - F_{\tau-1}\| < \delta$ for some predetermined tolerance level δ. In the last line of Algorithm 1 we designate the closest valid node to each centroid as a controller.

The iteration complexity for this algorithm depends on (a) calculation of mutual squared Euclidean distances between x_i, y_j for $i \in \{1, \dots, N\}, j \in \{1, \dots, m\}$, (b) similar calculation of mutual distances between centroids, (c) calculation of association probabilities and (d) solving the linear system of equations. The complexities for these operations are respectively, $O(NK_{max}d)$, $O(K_{max}^2 d)$, $O(K_{max}N)$ and $O(K_{max}^3 d^3)$. For large N these terms are dominated by $O(NK_{max}d)$, thus for a maximum number of iterations τ the algorithmic computational complexity for the leader-less case is $O(\tau NK_{max}d)$ which is linear in data size and dimensionality of data.

Leader-based case

In order to adapt DA to the leader-based ECP problem we define an appropriate distortion measure by:

$$D(x_i, y_j) = d(x_i, y_j) + \gamma \min_{j \in \{1,\dots,m\}} \sum_{j'=1}^{M} d(y_j, y_{j'}) \tag{3.14}$$

Similarly we can consider the weighted total distortion as:

$$D = \sum_{i \in \mathcal{N}} \sum_{j=1}^{M} p(y_j \mid x_i) d(x_i, y_j) + \gamma \min_{j \in \{1,\dots,m\}} \sum_{j'=1}^{M} N d(y_j, y_{j'}) \tag{3.15}$$

In order to observe its relation to MINLP objective function, notice (3.6) is equivalent to the following objective function:

$$\min_{(\mathcal{Q},\mathcal{X})} \quad \sum_{i=1}^{N} \sum_{j \in \mathcal{N}_h} q_{ij} d_{ij} + \gamma \min_{j \in \{1,\dots,m\}} \sum_{i \in \mathcal{N}_h} \hat{x}_i \left(N d_{ij}\right)$$

To establish this equivalence, we used the relationship that for $W = \{w_i\}_{i=1}^{m}$ and $S = \{W \in \mathbb{R}_+^n \mid \sum_{i=1}^{m} w_i = 1\}$ then $\min_{\mathcal{Z} \in S} \sum_{j=1}^{M} z_j \alpha_j = \min_{j \in \{1,\dots,m\}} \alpha_j$.

We define the system's free energy similar to the previous case. Setting the gradient with respect to the association probabilities to zero, yields solution:

$$p(y_j \mid x_i) = \frac{\exp\left(-\frac{D(x_i, y_j)}{T}\right)}{Z_i}, \qquad Z_i = \sum_{j=1}^{M} \exp\left(-\frac{D(x_i, y_j)}{T}\right)$$

Denote $j^* = \arg\min_{\mathrm{j} \in \{1,\dots,\mathrm{m}\}} \sum_{j'=1}^{M} D(y_j, y_{j'})$ as the index of the leader centroid and set gradient with respect to y_j to zero to yield the centroid update rules:

$$y_j = \frac{\gamma N y_{j^*} + \sum_{i \in \mathcal{N}} p(y_j \mid x_i) x_i}{\gamma N + \sum_{i \in \mathcal{N}} p(y_j \mid x_i)} \quad y_j \neq y_{j^*} \tag{3.16}$$

$$y_{j^*} = \frac{\gamma N \sum_{j' \neq j^*} y_{j'} + \sum_{i \in \mathcal{N}} p(y_{j^*} \mid x_i) x_i}{(m-1)\gamma N + \sum_{i \in \mathcal{N}} p(y_{j^*} \mid x_i)} \tag{3.17}$$

We can compute values of y_j and y_{j^*} by substituting (3.17) in (3.16). The resulting DA clustering algorithm for the leader-based case can be found in Algorithm 2.

The computational complexity for the leader-based algorithm is similar to the previous one, except for the centroid calculation step in which we no longer have to compute a linear system of equations. The computational complexity is $O(NK_{max}d) + O(K_{max}^2 d) + O((N + K_{max})d) + O(NK_{max})$. For a maximum of τ iterations and large N this is again dominated by $O(\tau N K_{max} d)$.

3.2 Dynamic Setting

In the context of controller placement in software-defined networks, the dynamic case can be regarded as an extension of the static case, wherein the problem is essentially a temporal concatenation of static problems. As the network topology and traffic patterns evolve over time, addressing the dynamic case

Algorithm 2: ECP-LB

Set limits: max # of clusters K_{max} and minimum temperature T_{min};
Initialize: $T \rightarrow \inf, K = 1, y_1 = \sum_{i \in \mathcal{N}} x_i p(x_i)$;
while *Convergence test* **do**

Update: $P_{y|x} \longleftarrow \exp\left(-\frac{d(x_i, y_j) + \gamma \min\limits_{j \in \{1,\ldots,m\}} \sum\limits_{j'=1}^{M} d(y_j, y_{j'})}{T} \right) / Z_i$;

Solve: $j^* = \arg\min_{\mathrm{j} \in \{1,\ldots,\mathrm{m}\}} \sum_{j'=1}^{M} D(y_j, y_{j'})$;

Solve:

$$y_j^{new} = \frac{\gamma N y_{j^*} + \sum_{i \in \mathcal{N}} P_{y|x} x_i}{\gamma N + \sum_{i \in \mathcal{N}} P_{y|x}} \quad j \neq j^*$$

$$y_{j^*}^{new} = \frac{\gamma N \sum_{j' \neq j^*} y_{j'} + \sum_{i \in \mathcal{N}} p(y_{j^*} \mid x_i) x_i}{(m-1)\gamma N + \sum_{i \in \mathcal{N}} p(y_{j^*} \mid x_i)}$$

Update: $y_j \longleftarrow y_j^{new}$ $j = 1, \ldots m$;
if $T \leq T_{min}$ **then**
  break;
else
  Cooling Step: $T \longleftarrow \alpha T (\alpha < 1)$;
  Generate small random vector ϵ
  Replace y_j with $y_j + \epsilon$ and $y_j - \epsilon$;
end
end
Perform last step iteration for $T = 0$;
$y_j \longleftarrow \arg\min_{x_i \in \mathcal{N}_h} d(x_i, y_j)$;

necessitates incorporating time-dependent information to optimize controller placement. A key piece of valuable information in this regard is the history of network node trajectories. By analyzing the past behavior of nodes, we can gain insights into their movement patterns and trends, enabling us to make informed predictions about the future state of the network. Leveraging historical trajectory data empowers us to develop predictive models that anticipate network changes, facilitating proactive controller placement strategies that adapt to the evolving network dynamics.

Moreover, in solving the extended dynamic problem efficiently, it is essential to consider the utilization of previously derived solutions. It is observed that the consecutive solutions to controller placement within a small time interval tend to exhibit proximity to each other due to the incremental nature of network changes. By re-using previous solutions as initial approximations for subsequent time steps, we can significantly reduce computation time and converge faster towards optimal solutions. This approach not only enhances computational efficiency but also ensures continuity and consistency in the controller placement process across time intervals. As a result, the network management system gains the ability to make intelligent and coordinated decisions, efficiently adjusting the controller's locations to cope with the temporal variations in the network and maintain optimal performance over time. Integrating historical trajectory analysis and leveraging past solutions, the dynamic controller placement problem can be effectively tackled, creating a dynamic network management framework that remains agile, adaptable, and resilient to the changing demands of the software-defined network environment.

3.2.1 RCP

The Real-time Controller Placement (RCP) algorithm brings a novel approach to controller node positioning in software-defined networks by prioritizing sheer speed and delivering simultaneously real-time and near-optimal solutions. One of the distinctive aspects that set RCP apart from traditional approaches is its utilization of two key pieces of information that are often overlooked in the SDN literature. Firstly, RCP leverages the history of network nodes, incorporating trajectory data, and past behavior to make informed predictions about the network's future state. By analyzing historical patterns and trends, RCP gains valuable insights into how network nodes are likely to evolve over time. This predictive capability empowers the algorithm to proactively adjust the positions of controller nodes in anticipation of forthcoming changes in network topology and traffic patterns. As a result, the network remains agile and adaptive, dynamically aligning its controller placements to the evolving demands of the environment.

Secondly, RCP adopts a solution adaptation approach instead of recalculating placements from scratch. When considering consecutive solutions within short time intervals, RCP capitalizes on the observation that the network's changes often exhibit incremental variations. As a result, the solutions for consecutive time steps tend to be close to each other. RCP leverages this inherent proximity between consecutive solutions, reusing the previously derived placement as a starting approximation for the subsequent time step. This adaptation strategy significantly reduces computational overhead and accelerates convergence towards optimal or near-optimal solutions. By avoiding redundant calculations, RCP achieves remarkable computational savings without compromising on the quality of controller placements. The efficiency gained through solution adaptation allows RCP to operate in real-time, providing rapid decision-making capabilities that match the dynamic nature of mobile software-defined networks.

Problem Statement

We make the same assumptions on the communication protocol, controller type, and cost function as in [56]. Specifically we assume network drones/UAV's are SDN-enabled and programmable via an API such as OpenFlow, similar to that described in [8]. In study paper, we will maintain a high-level focus on the topology of the network and placement of controllers

For simplicity, our analyses are given in $\mathbb{R}^2$ equipped with the Euclidean norm; however our results can be extended directly to $\mathbb{R}^n$ (namely, to $\mathbb{R}^3$ in which the basic problem lies). We further assume that the domain of the problem $\Omega \subset \mathbb{R}^2$ is a compact set that serves as the space within which the network operates. Throughout the paper we use the shorthand notation $[A]_{ij} = a_{ij}$ to represent the matrix A that is constructed by equating its ijth element to the scalar a_{ij}, and $A = \text{diag}(a_j)$ to represent the diagonal matrix A that has a_j as the jth diagonal element. We also use I_n to denote the identity matrix of size $n \times n$. We may occasionally drop the time index $t \in \mathbb{R}^+$ from the dynamical equations for the sake of readability and in such circumstances the reader can infer this from the context of the problem. We represent a mobile network as an undirected graph $G(\{\mathcal{N}, \mathcal{M}\}, \mathcal{E})$ with $\mathcal{N}$ as the set of network nodes of the graph, $\mathcal{M}$ as the set of controller nodes of the graph, and $\mathcal{E}$ as the set of edges connecting these two types of nodes. We will use $N = |\mathcal{N}|$ and $M = |\mathcal{M}|$ to denote the number of regular network nodes and controller nodes in the network, respectively. In the controller placement topology under study [5], there is communication between controllers and their assigned network nodes, and between controller nodes themselves, however network nodes do not directly communicate with each other. In fact, each node is *assigned* to a controller node that serves as a gate between the node and the rest of the network [56]. In this work, we further assume that the delay in the network is proportional to the squared Euclidean distance between source and origin of connection due to inverse-square law, which states that intensity is inversely proportional to the square of the distance from a source. Let $x_i(t), y_j(t) \in \mathbb{R}^2$ represent the location of the network node $i \in \mathcal{N}$ and controller node $j \in \mathcal{M}$, respectively, at time $t \in \mathbb{R}^+$. The dynamics of the network node $i \in \mathcal{N}$ is determined by the function $\phi_i(t, x, y) : \mathbb{R}^+ \times \mathbb{R}^{2N} \times \mathbb{R}^{2M} \to \mathbb{R}^2$ which we assume is continuously differentiable. The velocity of the controller node $j \in \mathcal{M}$ is determined by the vector $u_j(t) \in \mathbb{R}^2$; it is this velocity function we aim to design to achieve real-time controller placement by solving a cluster tracking control problem. We thus represent a mobile SDN system by the following dynamical system:

$$\begin{cases} \dot{x} = & \phi(t, x, y) \\ \dot{y} = & u \end{cases} \tag{3.18}$$

where $x \in \mathbb{R}^{2N}$, $y \in \mathbb{R}^{2M}$, $\phi(t, x, y) : \mathbb{R}^+ \times \mathbb{R}^{2N} \times \mathbb{R}^{2M} \to \mathbb{R}^{2N}$, and $u \in \mathbb{R}^{2M}$ are vectors that are constructed by concatenation of network and controller node location and velocity vectors. Denoting $\zeta = [x^T y^T]^T \in \mathbb{R}^{2(N+M)}$ as the vector containing all positional information of the network nodes and controller nodes, and letting $f(t, x, y) = [\phi(t, x, y)^T u(t)^T]^T \in \mathbb{R}^{2(N+M)}$ denote the vector containing the velocities of these nodes; then we compactly refer to this first order possibly nonlinear state-space system as

$$\dot{\zeta} = f(\zeta). \tag{3.19}$$

Temporal clustering refers to separation of a time-indexed set of objects into disjoint partitions known as *clusters* that satisfy a certain degree of similarity. The most representative point within each cluster is

[5] This topology is known as leaderless if there is no hierarchy amongst the controller nodes.

typically called the centroid, which in this work is equivalent to the location of the controller node. Following the approach introduced by Rose (see [37]), we leverage the Maximum Entropy Principle for our solution, letting $p(y_j \mid x_i) \in [0,1]$ denote the intensity of association of network node $i \in \mathcal{N}$ with controller node (and cluster) $j \in \mathcal{M}$ such that $\sum_{j\in\mathcal{M}} p(y_j \mid x_i) = 1$; this quantity is also referred to as an association weight. We represent the objective function $F : \mathbb{R}^{N\times M} \times \mathbb{R}^{2N} \times \mathbb{R}^{2M} \to \mathbb{R}$ associated with this clustering problem by [6]

$$
\begin{aligned}
F(P_{y|x}, x, y) = & \underbrace{\sum_{\substack{i\in\mathcal{N}\\ j\in\mathcal{M}}} p(y_j \mid x_i)\|x_i - y_j\|^2}_{D_1\text{:delay cost}} \\
& + \gamma \underbrace{\sum_{\substack{j\in\mathcal{M}\\ j'\in\mathcal{M}}} \|y_j - y_{j'}\|^2 \sum_{i\in\mathcal{N}} p(y_j \mid x_i)}_{D_2\text{:synchronization cost}} \\
& - T \underbrace{\left(- \sum_{\substack{i\in\mathcal{N}\\ j\in\mathcal{M}}} p(y_j \mid x_i) \log p(y_j \mid x_i) \right)}_{\text{H: entropy}}
\end{aligned}
\tag{3.20}
$$

Here we aim to find a set of trajectories for y_j which minimizes $F(P_{y|x}, x, y)$, thereby minimizing latency and synchronization times by using a maximum entropy function to help convexify the original problem. Note that so-called migration [7] cost is outside the of scope of the present paper and so our objective function does not reflect this cost, as in [38], [30], and [8] among others.

Solution

Based on the results of [57] and [19], in which the authors extend the results of Rose to show that for a given set of trajectories, $\{y_j(t)\}$, a Gibbs distribution will minimize equation (3.20), we have

$$
p(y_j \mid x_i) = \exp\left(-\frac{d(x_i, y_j)}{T}\right)/Z_i \quad \forall i \in \mathcal{N}, j \in \mathcal{M}, \tag{3.21}
$$

where $d(x_i, y_j) = \|x_i - y_j\|^2 + \sum_{j'\in\mathcal{M}} \|y_j - y_{j'}\|^2$ is the distance function, or so-called distortion measure [8] between node i and controller node j, and Z_i can be seen as the usual normalizing partition function. These association weights will become "hard" if we let T go to zero, and uniform, as T approaches a very high value. Formulation (3.20) and the "annealing" or "temperature" parameter T ensures that the total system delay attains a good local minimum in theory, while the nodes are initially maximally noncommittal towards the controllers. This latter point is essential since according to the maximum entropy principle in information theory, among all candidate distributions, the one with highest entropy best describes the current state of the system. This approach also has advantages for optimization over the surface of the cost function (3.20) where local optima abound [37].

Using the terminology of [19], let $[P_{y|x}]_{ij} = p(y_j|x_i) \in \mathbb{R}^{N\times M}$ be the matrix that contains information on the relative *shape* of the clusters. Similarly define $[P_{x|y}]_{ij} = p(x_i|y_j) \in \mathbb{R}^{N\times M}$ as the matrix containing

[6] This is a relaxed version of the cost function in [56]. For details refer to [57].
[7] re-positioning controllers and re-assigning network nodes.
[8] Distortion is a term in information theory that signifies the dissimilarity or distance between two points.

posterior associations $p(x_i|y_j)$, which we calculate using Bayes' rule, with $p(x_i) = \frac{1}{N}$, for all $i \in \mathcal{N}$. Moreover, define $P_y = \text{diag}(p(y_j)) \in \mathbb{R}^{M\times M}$ as the matrix containing information on the *mass* of the clusters, where $p(y_j) = \sum_{i\in\mathcal{N}} p(y_j|x_i)p(x_i)$. Let $\check{P}_{y|x}$, $\check{P}_{x|y} \in \mathbb{R}^{2N\times 2M}$ and $\check{P}_y \in R^{2M\times 2M}$, such that $\check{P}_{y|x} = P_{y|x} \otimes I_2$, $\check{P}_{x|y} = P_{x|y} \otimes I_2$, and $\check{P}_y = P_y \otimes I_2$. In prior work, we've shown that the optimal placement of controller y with respect to energy function F follows as:

$$y = \Theta^{-1}\check{P}_{x|y}^T x \tag{3.22}$$

where $\Theta \in \mathbb{R}^{2M\times 2M}$ is a block matrix with M^2 blocks of size 2×2. The diagonal blocks are equal to ηI_2 where $\eta = \gamma(M-1)+1$ and the non-diagonal blocks are equal to $-\gamma I_2$. We show that for $\gamma \neq \frac{1}{N-M}$ and $\gamma \neq \frac{1}{N-2M}$, Θ^{-1} is well defined (see [57]). Ideally we want the function F to serve similar to a control Lyapunov function, requiring the time derivative of F along the trajectory of network nodes and controllers to be non-positive which then ensures a non-increasing value for (3.20), our objective function. Following the development in [57] and [58], we can show the following.

Lemma 3.2. *Given the control Lyapunov function (3.20), for the system defined in (3.18) the time derivative of F has the following structure.*

$$\dot{F} = 2\zeta^T\Gamma(\zeta)f(\zeta),\ \Gamma(\zeta) = \begin{bmatrix} I_{2N\times 2N} & -\check{P}_{y|x} \\ -\check{P}_{y|x}^T & N\Theta P_y \end{bmatrix} \tag{3.23}$$

Proof. Using basic calculus and taking partial derivatives of F with respect to its constituents we can show that $\forall i \in \mathcal{N}$ and $j \in \mathcal{M}$,

$$\frac{\partial F}{\partial x_i} = 2\left(x_i \sum_{j\in\mathcal{M}} p(y_j \mid x_i) - \sum_{j\in\mathcal{M}} y_j p(y_j \mid x_i)\right)$$

and

$$\frac{\partial F}{\partial y_j} = 2\eta N p(y_j)y_j - N\gamma p(y_j)\sum_{j'\neq j} y_{j'} - \sum_{i\in\mathcal{N}} p(y_j \mid x_i)x_i.$$

Thus so far we have shown that

$$\frac{\partial F}{\partial \zeta} = 2\zeta^T\Gamma(\zeta).$$

The next step is to apply the chain rule

$$\frac{dF}{dt} = \frac{\partial F}{\partial \zeta}\frac{\partial \zeta}{\partial t} = \frac{\partial F}{\partial \zeta}f(\zeta),$$

from which the desired result follows. □

Our goal here is to design a control law $u \in \mathbb{R}^{2M}$ such that the output of system (3.18) asymptotically tracks the optimal placement of controllers in the 2D plane based on the energy function (3.32). Following the results of [59], [60], and [19] we propose the following control law, and show it results in a non-increasing function $F(t)$.

Theorem 3.3. *For the nonlinear system given in Equation (3.19) if*

$$u = -\left[k_0 + \frac{(x^T - y^T\check{P}_{y|x}^T)\phi}{\overline{y}^T\check{P}_y\overline{y}}\right]\overline{y} \tag{3.24}$$

where $K_0 > 0$ is a positive scalar and $\overline{y} = N\Theta\left(y - \Theta^{-1}\check{P}_{x|y}^T x\right)$, then $\dot{F}(t) \leq 0$ for all $t \in \mathbb{R}^+$.

Proof. We can expand equation (3.23) to get

$$\dot{F} = 2\left(\left(x^T - y^T\check{P}_{y|x}^T\right)\phi + \left(-x^T\check{P}_{y|x} + Ny^T\Theta\check{P}_y\right)u\right).$$

Now using the definition of $\overline{y}$ and $\frac{1}{N}\check{P}_{y|x} = \check{P}_{x|y}\check{P}_y$ we can show that

$$\dot{F} = 2\left(\left(x^T - y^T\check{P}_{y|x}^T\right)\phi + \overline{y}^T\check{P}_y u\right). \tag{3.25}$$

Substituting the control law (3.24) into (3.25) gives us $\dot{F} = -2K_0\overline{y}^T P_y\overline{y} \leq 0$ for all $\overline{y} \in \mathbb{R}^{2M}$, since $\check{P}_y$ is assumed to be a positive definite matrix [9] and $K_0 > 0$. □

To make explicit the result that y asymptotically tracks the optimal placement of controllers we state Theorem 3.5, which follows. We first introduce Lemma 3.4 without proof which is useful in the proof of Theorem 3.5. For details please see Lemma E.1. in [19].

Lemma 3.4. *For a non-negative function $f : \mathbb{R} \to \mathbb{R}$ of bounded variation, if $\int_0^\infty f(t)dt < \infty$ then $\lim_{t\to\infty} f(t) = 0$.*

Theorem 3.5. *For the dynamics given by system (3.18) using the control law in (3.24), y asymptotically tracks the optimal placement of controller nodes. That is:*

$$\lim_{t\to\infty} y(t) = \Theta^{-1}P_{x|y}x(t)$$

Proof. Because the system is constrained within the compact set Ω, the continuous real-valued function F is bounded from below. Since $\dot{F} \leq 0$ we must have that $F(t) \to F_\infty$ as $t \to \infty$. This implies that $\int_0^\infty -\dot{F}(\tau)d\tau = F(0) - F(\infty) < \infty$. Because $-\dot{F}$ is non-negative, using Lemma 3.4 we deduce that $\lim_{t\to\infty} -\dot{F}(t) = 0$. Since $-\dot{F}(t) = 2K_0\overline{y}^T\check{P}_y\overline{y}$ and $\check{P}_y$ is positive definite then we must have $\overline{y}(t) \to 0$ as $t \to 0$. Using the definition of $\overline{y}$ and because Θ is invertible by design, $y - \Theta^{-1}\check{P}_{x|y}x \in \text{Null}(\Theta) = \{0\}$. This yields that $y(t) \to \Theta^{-1}\check{P}_{x|y}x$ as $t \to \infty$ analogous to Equation (3.22). □

Computational Complexity The computational complexity of RCP can be determined as follows: (1) calculating mutual distances between all x_i, y_j pairs ($i \in \mathcal{N}$, $j \in \mathcal{M}$); (2) completing similar calculations for mutual distances between controllers; (3) calculating the distortions $d(x_i, y_j)$ for all $i \in \mathcal{N}$, $j \in \mathcal{M}$; (4) calculating association probabilities; and (5) updating the y_j trajectories. The complexities for these operations are respectively: (1) $\mathcal{O}(NMd)$, (2) $\mathcal{O}(M^2d)$, (3) $\mathcal{O}(Md)$, (4) $\mathcal{O}(NM)$ and (5) $\mathcal{O}(MNd^2)$. For terms (1) to (4) the calculations are analogous to those in the study completed in [57]. For term (5) the result comes from the fact that updating the y_j requires calculating the numerator $\mathcal{O}(MNd^2 + Nd)$ and denominator $\mathcal{O}(M^2d^2)$, plus the final multiplication by $\overline{y}$, which is dominated by the term $\mathcal{O}(MNd^2)$ for $N >> M$. For a fixed time horizon τ we can express the overall computational complexity of RCP as $\mathcal{O}(\tau NMd^2)$. This is a significant gain over other DCP algorithms like LiDy and LiDy+ discussed in [27], which respectively report $\mathcal{O}(N^2)$ and $\mathcal{O}(N^3)$ complexities in terms of network size. Using the result in Theorem 3.5 it can be seen that in the limit RCP becomes a variation of the ECP-LL algorithm in [57] that is computed for only one iteration per each system snapshot. This observations roughly means that, in the limit, RCP is $\frac{T}{d}$ times

[9] Note that degenerate (zero mass) clusters are not allowed in this formulation and diagonal elements of this diagonal matrix are always positive.

Algorithm 3: RCP

Initialize $y \in \mathbb{R}^{2M}$, starting temperature $T \approx \infty$, current time $t = 0$, time horizon $\tau \in \mathbb{R}^{+}$, time steps $n \in \mathbb{N}$, and decay rate $0 < \alpha < 1$;

$\Delta t \leftarrow \frac{\tau}{n}$;

for $i = 1$ **to** n **do**

update time: $t \leftarrow t + (i-1)\Delta t$;

update association weights $P_{y|x}$ using (3.21): $p(y_j \mid x_i) = \exp\left(-\frac{d(x_i, y_j)}{T}\right)/Z_i \quad \forall i \in \mathcal{N}, j \in \mathcal{M}$;

update y using control law (3.24):

$$u = -\left[k_0 + \frac{(x^T - y^T \check{P}_{y|x}^T)\phi}{\overline{y}^T \check{P}_y \overline{y}}\right] \overline{y}$$

$$y(t + \Delta t) \leftarrow y(t) + u(t)\Delta t$$

update system temperature: $T \leftarrow \alpha T$

end

faster than ECP-LL if on average ECP-LL takes T iterations to converge for each system snapshot. This is because running ECP-LL over the time horizon will on average require $\mathcal{O}(\tau TNMd)$ flops.

3.2.2 RCP+

RCP+ is an enhanced version of RCP that overcomes several challenges and achieves improved performance. It utilizes innovative techniques like MEP and convexification to produce high-quality solutions, even in the presence of many suboptimal local optima. The algorithm adopts a temporal approach to handle DCP, dynamically adapting solutions using SDN network history instead of recalculating. This results in quicker and more adaptive decision-making.

A notable feature of RCP+ is its real-time controller position updates, with response times as fast as 20ms. This real-time capability allows the network to rapidly respond to changing conditions. The algorithm also employs a lightweight neural network for trajectory prediction, enhancing future network planning.

RCP+ optimizes speed by employing sparse sampling of the SDN network and parallelizing its implementation on an Nvidia GPU.

To validate its performance, RCP+ is rigorously benchmarked against state-of-the-art DCP algorithms using extensive large-scale mobility datasets. These datasets contain millions of records, representing GPS trajectories of pedestrians and vehicles in major urban areas like San Francisco, US, and Beijing, China. This thorough evaluation establishes RCP+'s effectiveness in addressing dynamic network controller placement challenges.

Problem Statement

In [50] the authors show that the CP problem can be transformed to maximization of a submodular function, and use a randomized greedy algorithm that provides a guaranteed bound of 0.5 (1 being the optimal) from the optimal solution. We take a different approach and transform the CP problem into a clustering problem which better exploits underlying structure, namely segmentation of the network into disjoint sets assigned to

each controller [10]. This is illustrated by Figure 3.2 where each node cluster consists of many network nodes with varying dynamics and is assigned to a controller. Nodes can be mobile, like smartphones or stationary access switches, cellular base stations, and set-top boxes. We consider only edge nodes i.e., nodes that are closer to the user, unlike core switches and routers of ISP backbone networks or data centers, giving a specific type of SDMN where the SDN paradigm is applied to edge nodes (see [61]). Similar to [62], and [63] we assume that SDN soft-switches, like Open vSwitch (OvS), are deployed to handheld mobile devices, turning the smartphones into virtual switches that can receive control messages and forwarding rules from an SDN controller. Both controllers and switches in the proposed network can be non-stationary. In a realistic scenario there would be some stationary ground controllers to share the load with a flying control plane bringing more consistency to the network. Because some nodes can be of critical importance like busy cell sites, in the formulation we allow for a priori node weights that can alter controller placements in favor of strategic nodes. [8] and [9] study the use of SDN-enabled aerial devices such as drones as controllers, enabling a flying network infrastructure, as illustrated in Figure 3.2. The controllers will continuously exchange messages with the data plane nodes they manage for statistics collection and forwarding table updates. We adopt a similar approach and assume network controllers are SDN-enabled UAVs.

We perform our analyses in $\mathbb{R}^2$ equipped with the Euclidean norm and delay metric $d : R^2 \times R^2 \to [0, \infty)$ induced by squared Euclidean norm, i.e. $d(x, y) = \|x - y\|^2$, $\forall x, y \in R^2$. The delay metric has a connection to the well-known inverse square law in signal propagation strength. According to this law, the intensity or strength of a signal, such as light or gravity, decreases with the square of the distance from its source. In our case, the delay metric $d(x, y)$ expresses a similar concept. As the points x and y move further apart in $\mathbb{R}^2$, the squared Euclidean distance between them increases, leading to a higher delay in signal propagation or communication between these points. results, however can be readily extended to $\mathbb{R}^n$ [11]. We also assume that the domain of the problem $\Omega \subset \mathbb{R}^2$ is a compact set that serves as the space within which the network operates. Throughout the rest of this study we use the shorthand notation $[A]_{ij} = a_{ij}$ to represent the matrix A that is constructed by equating its ijth element to the scalar a_{ij}, and $A = \text{diag}(a_j)$ to represent the diagonal matrix A that has a_j as the jth diagonal element. A_{ij} also represents the ijth element of matrix A. For any matrix A, we denote $\breve{A} = A \otimes I_2$, i.e., the matrix expanded by Kronecker product. We also use I_n to denote the identity matrix of size $n \times n$. Matrix $\mathbb{1}_{n\times m}$ denotes the matrix of all one's with size $n \times m$. With a slight abuse of notation, here we define $\exp(A)$, and $\log(A)$ as, respectively the element-wise exponential and logarithm of a matrix A contrary to conventional generalizations of scalar exponential and logarithm functions.

We may occasionally drop the time index $t \in \mathbb{R}^+$ from the dynamical equations for the sake of readability and in such circumstances the reader can infer this from the context of the problem. We represent a SDMN as an undirected graph $G(\{\mathcal{N}, \mathcal{M}\}, \mathcal{E})$ with $\mathcal{N}$ as the set of nodes of the graph, $\mathcal{M}$ as the set of controller nodes of the graph, and $\mathcal{E}$ as the set of edges connecting these nodes. We will use $N = |\mathcal{N}|$ and $M = |\mathcal{M}|$ as the number of network nodes and controllers in the network, respectively. In the controller placement topology under study, there is communication between controllers and their assigned network nodes and between controllers themselves [12].

In this work we assume that the delay in the network is proportional to the squared Euclidean distance between source and origin of connection. Let $x_i(t), y_j(t) \in \mathbb{R}^2$ represent the location of the network node

[10] The assignee of a segment is called the centroid or exemplar in clustering terminology, which is the most representative point of that segment. Here the centroids indicate the position of controllers in the network

[11] e.g. instead of restraining network nodes to a 2D plane, it is entirely possible to work with three dimensional positions of nodes in space.

[12] This topology is known as *leaderless*, where it is assumed there is no hierarchy between the controllers [56]

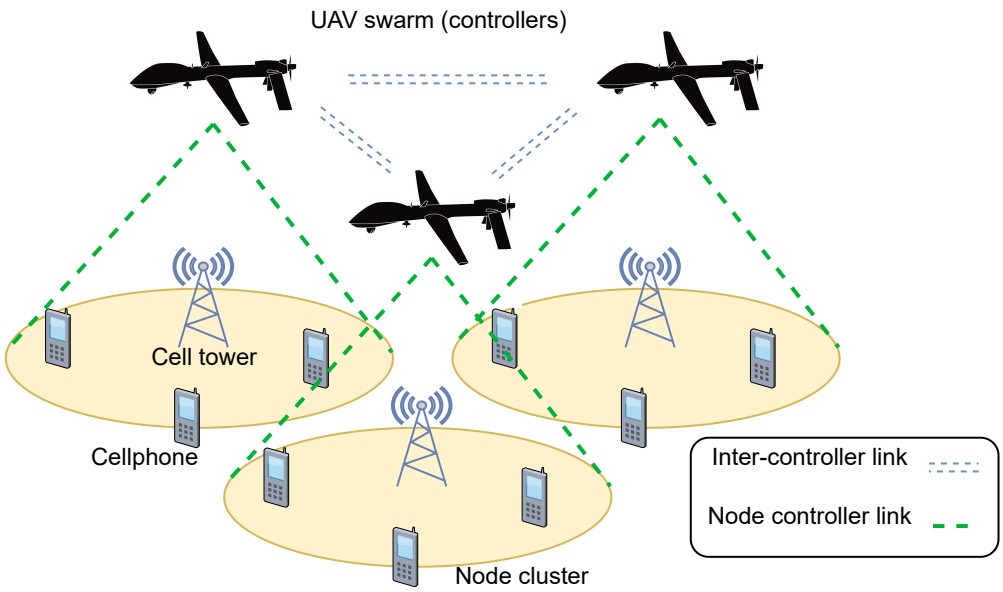


Figure 3.2: Software defined mobile network.

$i \in \mathcal{N}$ and controller $j \in \mathcal{M}$, respectively, at time $t \in \mathbb{R}^+$. The dynamics of the network node $i \in \mathcal{N}$ is determined by the function $\varphi_i(t) : \mathbb{R}^+ \to \mathbb{R}^2$ and we assume it is continuously differentiable, Lipschitz and bounded. Velocity of the controller $j \in \mathcal{M}$ is determined by the vector $u_j(t) \in \mathbb{R}^2$. We can represent a mobile SDN system by the following dynamical system:

$$\begin{cases} \dot{x} = & \varphi \\ \dot{y} = & u \end{cases} \tag{3.26}$$

where $x \in \mathbb{R}^{2N}$, $y \in \mathbb{R}^{2M}$, $\varphi(t) : \mathbb{R}^+ \to \mathbb{R}^{2N}$, and $u \in \mathbb{R}^{2M}$ are vectors that are constructed by concatenation of network and control node location and velocity vectors. Denote $\zeta = [x^T y^T]^T \in \mathbb{R}^{2(N+M)}$ as the vector containing all positional information of the network nodes and controllers and let $f(t) = [\varphi(t)^T u(t)^T]^T \in \mathbb{R}^{2(N+M)}$ denote the vector containing the velocities of these nodes; then we can compactly refer to this first order system as

$$\dot{\zeta} = f(\zeta). \tag{3.27}$$

In this setup we view DCP as the problem of finding the optimal trajectory of controllers $y(t)$, and optimal association weights of network nodes $P_{y|x}(t) \in \mathbb{R}^{N \times M}$ with these controllers, for $t \in \mathbb{R}^+$ such that a certain cumulative cost is optimized. Throughout this work we consider a hybrid cost that is comprised of two components, *synchronization* and *delay*. Delay cost is equivalent to the cumulative network delay resulting from communication between network nodes and controllers.

The synchronization cost is the cost of controllers transmitting data to each other and is dependent on the distance of controllers from each other as well as the number of nodes assigned to each controller. The more nodes that are assigned to a controller the more messages it exchanges with its peers to communicate state of network, which comprises the overhead property of inter-controller transmissions [50]. The relative importance of synchronization cost versus delay cost is tuned using a scalar. For further motivations for this

cost function view [50], and [34].

Two consecutive solutions can exhibit large spatial variations and in such cases temporal information is not preserved. That is, two solutions might provide similar objective values, but spatially might be very dissimilar. Finding the temporal relationship between controllers for two solutions is not always possible. Simply connecting points makes trajectories infeasible and unrealistic. In this light a solution adaption strategy is necessary to preserve the temporal information and a solution recalculation strategy by definition fails to satisfy this constraint.

Let $p(x_i)$ be the prior weight (or relative importance) of the network node $i \in \mathcal{N}$ and $p(y_j \mid x_i) \in [0,1]$, referred to as an association weight, denote the strength of the association of network node $i \in \mathcal{N}$ with controller $j \in \mathcal{M}$, with $\sum_{j\in\mathcal{M}} p(y_j \mid x_i) = 1$. We define matrix $[P_{y|x}]_{ij} = p(y_j \mid x_i)$ as the matrix containing these association weights. We define cluster $C_j \in \mathcal{M}$ as $\{i \in \mathcal{M} \mid \arg\max_{j\in\mathcal{M}} p(y_j \mid x_i) = j\}$, i.e., the set of nodes whose strongest associations are with controller $j \in \mathcal{M}$. Let $P_y = \text{diag}(p(y_j)) \in \mathbb{R}^{M\times M}$ be the matrix containing information on the *mass* of the clusters, where $p(y_j) = \sum_{i\in\mathcal{N}} p(y_j|x_i)p(x_i)$. Note that we assume clusters are nondegenerate, namely zero mass clusters are not allowed ($P_y \succ 0$). Let $p(x_i, y_j) = p(y_j \mid x_i)p(x_i)$ and $[P_{xy}]_{ij} = p(x_i, y_j)$. Moreover let $[P_{x|y}]_{ij} = p(x_i \mid y_j)$, where $p(x_i \mid y_j)$ is calculated using Bayes' rule. Let $[\Delta_{xy}]_{ij} = d(x_i, y_j) \in \mathbb{R}^{N\times M}$ and $[\Delta_{yy}]_{jj'} = d(y_j, y_{j'}) \in \mathbb{R}^{N\times M}$ define the node-controller and inter-controller distance matrices, respectively.

Given these definitions, we now define our objective function $D : \mathbb{R}^{N\times M} \times \mathbb{R}^{2M} \to \mathbb{R}$ by

$$D(P_{y|x}, y) = \underbrace{\mathbb{1}_N^T P_x \left(P_{y|x} \circ \Delta_{xy}\right) \mathbb{1}_M}_{\text{delay cost}} + \gamma \underbrace{\mathbb{1}_M^T \left(P_y \circ \Delta_{yy}\right) \mathbb{1}_M^T}_{\text{synchronization cost}}. \tag{3.28}$$

This objective function consists of the two competing metrics, one vying for inter-controller latency reduction and the other for controller-node latency reduction. The geometric counterpart of this competition would be a highly compact placement versus a highly dispersed placement.

More precisely this hybrid cost is cubic in terms of its arguments, and consists of two parts: the first term is the sum of distances of nodes $i \in \mathcal{N}$ from all controllers $j \in \mathcal{M}$ weighted by association weights and scaled by each node's importance; the second term, or synchronization cost is the sum of pairwise inter-controller delays amplified [13] by the mass of, or rather ratio of network nodes assigned to, each controller. The scalar $\gamma > 0$ determines the relative importance of the delay cost compared to the synchronization cost. If we set γ to zero it is possible to recover a classic facility location problem (see [64]), thus CP can be viewed as a generalization of this problem, augmented to consider distance between facilities.

Letting $\Delta = \Delta_{xy} + \gamma \mathbb{1}_{N\times M} \Delta_{yy}$ denote the distortion matrix [14] containing pairwise distortions between nodes $i \in \mathcal{N}$ and controllers $j \in \mathcal{M}$, i.e. $[\Delta]_{ij} = d(x_i, y_j) + \gamma \sum_{j'\in\mathcal{M}} d(y_j, y_{j'})$, we have

$$D = \mathbb{1}_N^T \left(P_{xy} \circ \Delta\right) \mathbb{1}_M, \tag{3.29}$$

thus delay and synchronization costs can be collectively viewed as the weighted sum of distortions in the network. Temporal clustering refers to separation of a time-indexed set of objects into disjoint segments known as *clusters* that satisfy a certain degree of similarity. The most representative point within each cluster is typically called the centroid. With this, the DCP problem herein is essentially a clustering problem where the controllers are the centroids and the network nodes assigned to them comprise the clusters, and cost

[13] This encodes the overhead property of inter-controller communication. For more information on this refer to [50]

[14] Distortion is a concept in information theory is the cost of approximating information. Here we use it as the degree of dissimilarity between two points.

function (3.28) is the objective function of this clustering problem. Time index $t \in \mathbb{R}^+$, which is dropped from expression (3.28) for readability, emphasizes the temporal aspect of our clustering problem. To be more precise $P_{y|x}$, P_y, P_{xy}, Δ_{xy}, and Δ_{yy} are all functions of time, through dependence on x, and y in the dynamical system (3.26).

Analogous to the procedure in [37] for deterministic annealing based clustering, we invoke the maximum entropy principle (MEP) from [65] for optimization of the cost function D. The MEP is a statistical inference method that provides the least biased estimate–given some information–that is *maximally noncommittal* towards missing information. More relevant to this context, the MEP states that among all possible distributions, the one with highest *entropy* best explains the current state of the system. Similar to [37] and using the Shannon's entropy in [66] defined as

$$H(y \mid x) = -\mathbb{1}_N^T P_x \left(P_{y|x} \circ \log P_{y|x}\right) \mathbb{1}_M, \tag{3.30}$$

we configure the MEP based problem to maximize the entropy, subject to a certain distortion level $D_0 \in \mathbb{R}$, namely

$$\begin{aligned} &\max\ H(y \mid x) \\ &\text{s.t.} \quad D = D_0,\ P_{y|x}\mathbb{1}_M = 1,\ P_{y|x} \geq 0. \end{aligned} \tag{3.31}$$

Using a Lagrangian relaxation, we relax this problem into that of maximizing the Lagrangian function $L = H - \beta\,(D - D_0)$ parametrized by a Lagrange multiplier β, and subject to probability (or association weight) constraints. This maximization is equivalent to minimization of a so-called energy function F, given in equation (3.32) where $T = \frac{1}{\beta}$, that is, minimization of

$$F = D - TH. \tag{3.32}$$

This energy function is analogous to Helmholtz free energy in statistical mechanics and interesting relations between the two can are discussed in [37] and [66]. Thus the MEP problem can be viewed as a distortion minimization problem subject to probability constraints, which we refer to as the *master problem*:

$$\begin{aligned} &\min\ F(P_{y|x}, y) \\ &\text{s.t.} \quad P_{y|x}\mathbb{1}_M = 1,\ P_{y|x} \geq 0. \end{aligned} \tag{3.33}$$

By proper annealing of the parameter T, also known as temperature, it is possible to start with highly noncommittal solutions and slowly gravitate towards a local optimum of the cost function D. Addition of the entropy term H to the original objective function (3.28) has implications beyond just unbiasedness and is far important than a mere mathematical juxtaposition. This is actually the underlying mechanism that allows RCP+ to avoid poor local optimal that abound in the cost surface of D, and yield higher quality solutions compared to generic optimization methods that rely on multiple random initializations to discover a desirable solution.

More specifically, addition of an entropy term also can be viewed as a convexification procedure similar to those proposed by [67], where a convex term with a large multiplier is added to the objective function allowing reaping the well-behaved results of convex optimization. To gain intuition as to why this is the case we provide following example.

Example 3.1. *Consider the real-valued function* $f : \mathbb{R} \to \mathbb{R}$ *such that* $f(x) = x^4 - 13x^3 + 62x^2 - 129x + 99$. *The local minima of this function are respectively* $x_1 = \frac{27-\sqrt{41}}{8}$ *and* $x_2 = \frac{27+\sqrt{41}}{8}$ *as can be seen in Figure 3.3*

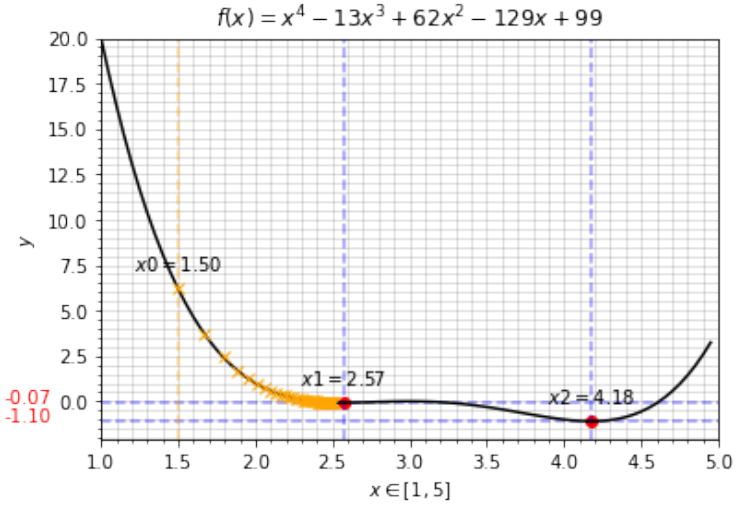


Figure 3.3: Optimization landscape.

Starting at $x_0 = 1.5$ *and using the conventional gradient descent we end up in the local minimum* x_1. *It can also be seen starting anywhere to the left of* $x = 3$ *we always get stuck in* x_1 *before reaching the global minimum* x_2. *With a minor tweak consider the function* $\hat{f}(x) = f(x) - T\log(x)$, *where* $T \in \mathbb{R}^+$ *and* $-T\log(x)$ *can be viewed as a convex term augmenting the original function. Instead of performing the usual gradient descent, at each descent iteration we also update the value of* T *such that* $T \leftarrow \alpha T$ *and* $\alpha = 0.8$. *Starting with* $T = 100$ *the result can be seen in Figure 3.4 starting at the same point, after third iteration gradient descent successfully "escapes" the shallower minimum and makes its way towards the global minimum. This is in fact the same mechanism behind maximum entropy principle that helps reaching minima that are closer to the global minimum.*

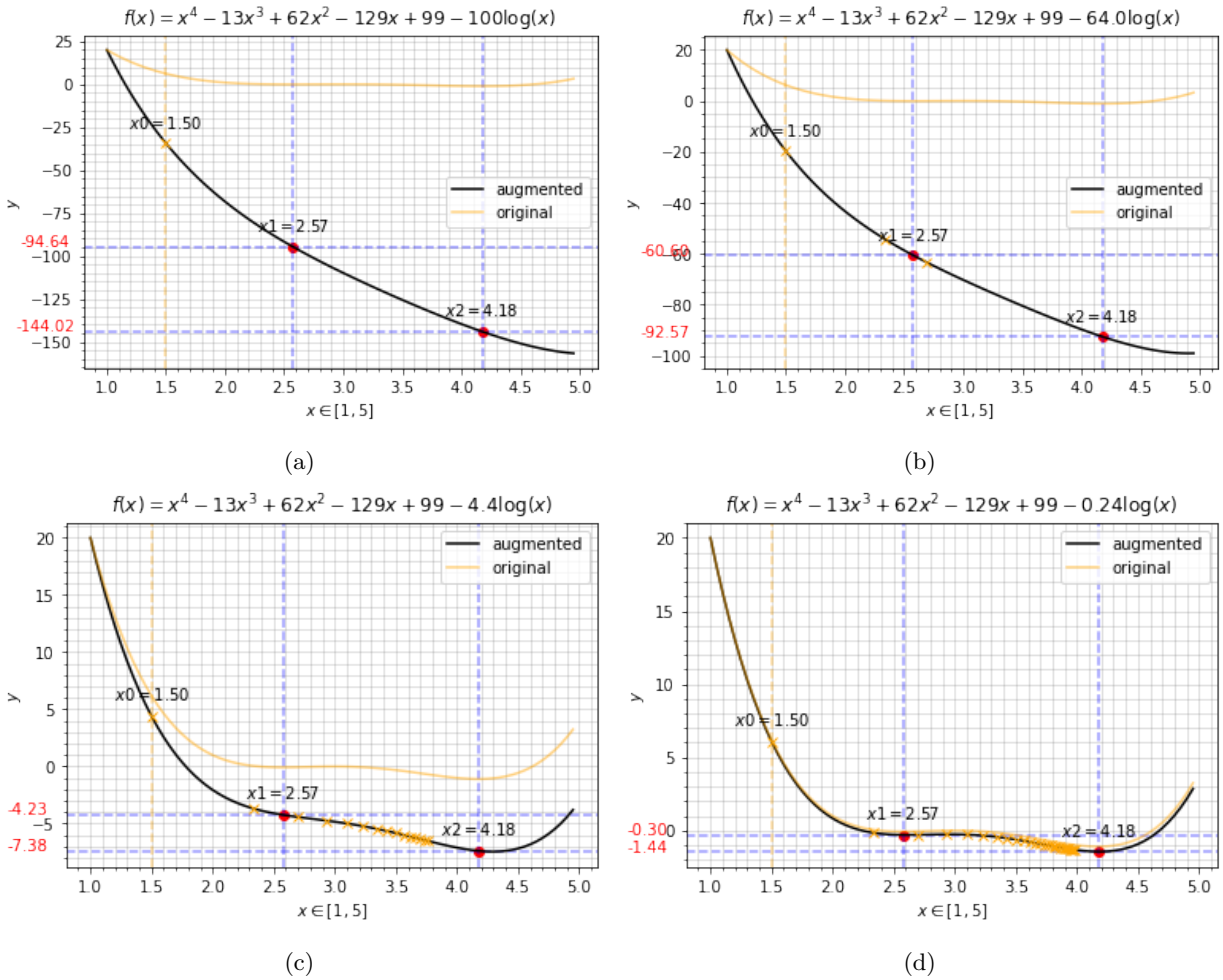


Figure 3.4: Convexification progress

At $T = \infty$ one may think of the convexified function as a convex envelope [68] around the original objective that deflates with depreciation of T (as in Figure 3.4). During this process the deeper optima "stick out" faster as the convexified function is morphing back into it's original shape. Due to this, conventional descent based algorithms gravitate towards deeper optima earlier than they do towards shallower ones. In [37], Rose appropriately names a similar process Deterministic Annealing (DA) analogous to its physical interpretation of *annealing* in metallurgy.

Theorem 3.6. *Convexification: Assume $f(x) : \mathbb{R}^n \to \mathbb{R}$ twice differentiable and has bounded gradient, then for any strongly convex $c(x) : \mathbb{R}^n \to \mathbb{R}$ there exists $t_0 > 0$ such that $\forall t \geq t_0$, $\tilde{f}(x) = f(x) + tc(x)$ is strictly convex.*

Proof. Because f has bounded gradient then it is Lipschitz continuous and consequently, $\forall x \in E$ we have $-LI \preceq \nabla^2 f(x) \preceq LI$, such that $L > 0$ is a Lipschitz constant of $f(x)$. Suppose $c(x)$ is α-strongly convex such that $\alpha I \preceq \nabla^2 c(x)$. We note that for any $x \in \mathbb{R}^n$ we have $\lambda_{min}(\nabla^2 \tilde{f}(x)) \geq \lambda_{min}(\nabla^2 f(x)) + t\lambda_{min}(\nabla^2 c(x))$, and for $t \geq \dfrac{L}{\alpha}$, we have $\lambda_{min}(\nabla^2 \tilde{f}(x)) > 0$ indicating the strict convexity of $\tilde{f}(x)$. □

If we restrict the controller placements to coincide with that of network nodes [15] then at each time interval

[15] See reference [50] for a motivation of this restriction.

$t \in \mathbb{R}^+$ we can find the optimal placement of controllers as the solution of a constrained optimization problem. To cast this auxiliary problem as a mathematical program (at time $t \in \mathbb{R}^+$) we define $\mathcal{X} = (\hat{x}_i \in \{0,1\}, i \in \mathcal{N}_h)$ where $\mathcal{N}_h \subseteq N$ is the set of nodes that can accommodate a controller. $\mathcal{X}$ represents the controller placement policy such that if we choose node i to be a controller then $\hat{x}_i = 1$ otherwise $\hat{x}_i = 0$. Similarly $\mathcal{Q} = (q_{ij} \in \{0,1\}, i \in \mathcal{N}, j \in \mathcal{N}_h)$ determines the controller assignment policy where $q_{ij} = 1$ if node i is assigned to controller j otherwise $q_{ij} = 0$. $d_{ij} = d(x_i, x_j)$ encodes the communication delay between nodes i and j which we assume to be proportional to the squared Euclidean distance.

We can express the optimal placements and assignments as the solution of the following mathematical program [16]:

$$\min_{\mathcal{Q},\mathcal{X}} \quad \sum_{i=1}^{N} p(x_i) \sum_{j \in \mathcal{N}_h} q_{ij} d_{ij} + \gamma \sum_{i,j \in \mathcal{N}_h} \hat{x}_i \hat{x}_j d_{ij} \sum_{k \in \mathcal{N}} p(x_k) q_{kj} \tag{3.34}$$

$$\text{s.t.} \quad \sum_{j \in \mathcal{N}_h} q_{ij} = 1 \quad \forall i \in \mathcal{N} \tag{3.35}$$

$$q_{ij} \leq \hat{x}_j \qquad \forall i, j \in \mathcal{N} \tag{3.36}$$

$$\hat{x}_i \in \{0,1\}, \quad i \in \mathcal{N}_h \tag{3.37}$$

$$q_{ij} \in \{0,1\}, \quad i \in \mathcal{N}, j \in \mathcal{N}_h, \tag{3.38}$$

This SDN setup is similar to the the works in [56], [34], and [57] with the addition of network nodes prior weights. This yields a Mixed Integer Nonlinear Program (MINLP) with no guarantees for the global optimal solution, and poor local minima abound in its optimization landscape. According to our most recent experiments, finding a *local optimum* of optimization problem (3.34)-(3.38) using the state of art MINLP solver BARON on a conventional PC, with network size of 50 nodes can take up to several hours. As such, solving this problem for each time step $t \in \mathbb{R}^+$ is not practical for real-time purposes.

Solution

In [57] we have shown that equation (3.28) can be used as a proxy [17] for objective function (3.34). This allows design of a clustering approach to this problem that essentially relaxes this MINLP and serves as a heuristic for it. This relaxation significantly reduces the computational complexity and has been shown to find better solutions than the state of art MINLP solver, BARON. However this solution is designed for a static system where nodes of the network are stationary and cannot be directly applied to a mobile SDN setting. A naive approach to leverage this static solution in a dynamic setting is to *freeze* the system and find the placement of controllers for each frozen snapshot of the system at some $t \in \mathbb{R}^+$. This static approach, as discussed in [19], is known as the *frame-by-frame* approach, and is computationally suboptimal due to starting from scratch at each time frame where information from previous system snapshots is not utilized.

Computability is an issue for all DCP methods in the literature as they fall under the frame-by-frame category and fall short of real-world practicality for real-time purposes, such as when SDN controllers are UAV's and need split-second positional updates. In this light the RCP algorithm in [14] was introduced as the first real-time and dynamics aware method that exploited the temporal information in the mobile SDN system. RCP can also be viewed as a temporal version of the ECP-LL algorithm in [57].

Lemma 3.7 below studies the structure of F and its first order information with respect to its constituents.

[16] Here we imply that $\mathcal{Q}, \mathcal{X}, d_{ij}, i \in \mathcal{N}, j \in \mathcal{N}_h$ are functions of $t \in \mathbb{R}^+$ and we drop the time index for readability.
[17] In this earlier work a uniform distribution is assumed for network prior weights.

Lemma 3.7. *First order information: For the energy function F defined as in equation (3.32) the partial derivatives with respect to network nodes x, controller nodes y, and association weights $P_{y|x}$ are given as follows:*

$$\begin{aligned}
\frac{\partial F}{\partial x} &= 2(x^T \check{P}_x - y^T \check{P}_{xy}^T), \\
\frac{\partial F}{\partial y} &= y^T \check{P}_y \check{\Theta} - x^T \check{P}_{xy}, \\
\frac{\partial F}{\partial P_{y|x}} &= P_x \left(\Delta + T\left(\log P_{y|x} + \mathbb{1}_{N\times M}\right)\right),
\end{aligned} \qquad \Theta \in R^{N\times M} = \begin{bmatrix} \eta & -\gamma & \dots & -\gamma \\ -\gamma & \eta & \dots & -\gamma \\ \vdots & \vdots & \ddots & \vdots \\ -\gamma & -\gamma & \dots & \eta \end{bmatrix}, \quad \eta = \gamma(M-1)+1. \tag{3.39}$$

Proof. We will study the component-wise derivative of F with respect to x_i, $i \in \mathcal{N}$ and y_j, $j \in \mathcal{M}$ and extend the result to the vectors x and y. By expanding (3.32) and using straightforward calculus we can show that:

$$\frac{\partial F}{\partial x_i} = 2\left(p(x_i)x_i - \sum_{j\in\mathcal{M}} y_j p(x_i, y_j)\right) \tag{3.40}$$

$$\frac{\partial F}{\partial y_j} = p(y_j)\left(\eta y_j - \gamma \sum_{j'\neq j} y_{j'}\right) - \sum_{i\in\mathcal{N}} x_i p(x_i, y_j) \tag{3.41}$$

$$\frac{\partial F}{\partial p(y_j \mid x_i)} = p(x_i)\left(d(x_i, y_j) - \gamma \sum_{j'\in\mathcal{M}} d(y_j, y_{j'}) + T\left(\log p(y_j \mid x_i) + 1\right)\right) \tag{3.42}$$

Equations (3.40), (3.41) now readily yield the matrix form given in Equation (3.39). □

The absence of the variable y in constraints of the master problem (3.33) signals its decomposability. A natural way to obtain this decomposition is to break the problem into placement (optimization over y) and assignment (optimization over $P_{y|x}$) subproblems as shown below:

$$\begin{aligned}
\min_{y} \min_{P_{y|x}} \quad & F(y, P_{y|x}) \\
\text{s.t.} \quad & P_{y|x}\mathbb{1}_M = \mathbb{1}_N \\
& P_{y|x} \geq 0
\end{aligned} \tag{3.43}$$

where we recognize the inner problem as the *assignment* subproblem, i.e.,

$$F^*(y) = \operatorname*{arg\,min}_{P_{y|x}: P_{y|x}\mathbb{1}_M = \mathbb{1}_N, P_{y|x}\geq 0} F(P_{y|x}, y).$$

Consequently, the outer or the master problem can be viewed as $\min_y F^*(y)$. We call the closely related partial minimization problem $\min_y F(y, P_{y|x})$ the *placement* subproblem.

A deeper inspection of this decomposition yields that the complexity of the problem lies in the placement subproblem as opposed to the assignment subproblem. Theorem 3.8 below sheds more light on this observation, which we will exploit to tackle the master problem efficiently. It also reinforces the idea of decomposition into placement and assignment subproblems as a natural way to exploit underlying structure of the problem. As an aside, henceforth we use the term *optimal* or *minimizer* throughout the paper to refer to a locally optimal solution, unless we explicitly refer to the global optimum.

Theorem 3.8. *Coupled solution: For any $P^*_{y|x} \in \mathbb{R}^{N\times M}$, and $y^* \in \mathbb{R}^{2M}$ such that $(P^*_{y|x}, y^*)$ is a minimizer of master problem (3.33), then $P^*_{y|x}$ follows a Gibbs distribution and there exists maps $\nu : \mathbb{R}^{N\times M} \to \mathbb{R}^{2M}$, and $\rho : \mathbb{R}^{2M} \to \mathbb{R}^{N\times M}$ such that $\nu(P^*_{y|x}) = y^*$ and $\rho(y^*) = P^*_{y|x}$.*

Proof. Assume we know the optimal placements y^*, then master problem (3.33) reduces to

$$\begin{aligned} \min_{P_{y|x}} \; & F(P_{y|x}, y^*) \\ \text{s.t. } & P_{y|x}\mathbb{1}_M = \mathbb{1}_N \\ & P_{y|x} \geq 0 \end{aligned} \tag{3.44}$$

We note that this is a linearly constrained convex optimization problem that we can solve analytically using KKT conditions. The associated Lagrangian $L : \mathbb{R}^{N\times M} \times \mathbb{R}^N \times \mathbb{R}^{N\times M} \to \mathbb{R}$ can be written as $L(P_{y|x}, \lambda, \mu) = D - TH + \lambda^T \left(P_{y|x}\mathbb{1}_M - 1\right) - \mathbb{1}_N^T \left(\mu \circ P_{y|x}\right)\mathbb{1}_M$, and the KKT conditions are

$$\nabla_{p(y_j|x_i)} L(P_{y|x}, \lambda, \mu) = p(x_i)\Delta_{ij} + Tp(x_i)\left(\log p(y_j \mid x_i) + 1\right) + \lambda_i - \mu_{ij} = 0 \quad \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{3.45}$$

$$\lambda_i \left(\sum_{j\in\mathcal{M}} p(y_j \mid x_i) - 1 \right) = 0 \; \forall i \in \mathcal{N}, \quad \mu_{ij} p(y_j \mid x_i) = 0 \; \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{3.46}$$

$$\sum_{j\in\mathcal{M}} p(y_j \mid x_i) = 1 \; \forall i \in \mathcal{N}, \quad p(y_j \mid x_i) \geq 0 \; \forall i \in \mathcal{N}, j \in \mathcal{M}, \quad \mu_{ij} \geq 0 \; \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{3.47}$$

The stationarity condition in (3.45) implies that $p(y_j \mid x_i) = \exp\left(-\frac{\Delta_{ij}}{T}\right)\exp\left(-\frac{\lambda_i+\mu_{ij}}{Tp(x_i)}\right)$, indicating the inequality conditions $p(y_j \mid x_i) > 0$ strictly hold, thus due to complimentary slackness conditions in (3.46), $\mu_{ij} = 0$, $\forall i \in \mathcal{N}, j \in \mathcal{M}$. Furthermore, substituting the value of $p(y_j \mid x_i)$ in the feasibility conditions in (3.47), we get $\lambda_i = Tp(x_i)\log\frac{1}{\sum_{j\in\mathcal{M}}\exp(-\Delta_{ij}/T)}$, $\forall i \in \mathcal{N}$, which yields the optimal solution to problem (3.44) as $p(y_j \mid x_i) = \frac{\exp(-\Delta_{ij}/T)}{\sum_{j\in\mathcal{M}}\exp(-\Delta_{ij}/T)}$, $\forall i \in \mathcal{N}, j \in \mathcal{M}$; this is the celebrated Gibbs or Boltzmann distribution. Finally because the assignment subproblem satisfies the well-known Slater condition [18] the KKT solution is also the global minimum of the assignment subproblem. Let $Z = diag(z_i)$, and z_i for $i \in \mathcal{N}$ be a normalizing factor that ensures $\sum_{j\in\mathcal{M}} p(y_j \mid x_i) = 1$, i.e. $Z_i = \sum_{j\in\mathcal{M}} \exp\left(-\frac{\Delta_{ij}}{T}\right)$ thus $\rho(y^*) = Z^{-1}\exp\left(-\frac{\Delta}{T}\right) = P^*_{y|x}$.

To show the converse, by substituting $P^*_{y|x}$ in the master problem, (3.44) reduces to $\arg\min_y F(P^*_{y|x}, y) = y^*$. We note this is an unconstrained convex optimization problem and solving for a stationary point yields the global minimum $y^* = \check{\Theta}^{-1}\check{P}^{*^T}_{x|y}x = \nu(P^*_{y|x})$ which is readily available from equating (3.41) to zero. $\check{P}^{*^T}_{x|y}$ is the optimal posterior distribution calculated using Bayes' rule, namely $\check{P}^{*^T}_{x|y} = \check{P}_x\check{P}^*_{y|x}\check{P}_y^{-1}$. □

Finding y^* if $P^*_{y|x}$ is known is not as analytically demanding but is computationally more expensive, thus the complexity of the master problem is dominated by the placement subproblem. Despite availability of y^* in closed form it is beneficial to solve the system of equations $\frac{\partial F}{\partial y} = 0$ directly, as it can be faster than the matrix inversion given below in (3.49).

Theorem 3.8 shows the optimal solution to the master problem inherently has a coupled structure, where optimal assignments and placements can be found in terms of each other, in closed form. However this does not immediately provide a way to optimize the master problem. Corollary 3.1 provides a procedure to narrow down the search to solutions that satisfy this coupled property.

[18] Namely, there exists $P_{y|x} > 0$, strictly satisfying inequality conditions.

Corollary 3.1. *Convergence to minimizer: Define maps* $\nu : \mathbb{R}^{N\times M} \to \mathbb{R}^{2M}$, *and* $\rho : \mathbb{R}^{2M} \to \mathbb{R}^{N\times M}$ *such that*

$$\rho(y) = \underset{P_{y|x}:P_{y|x}\mathbb{1}_M=\mathbb{1}_N, P_{y|x}\geq 0}{\arg\min} F(P_{y|x}, y) = Z^{-1}\exp\left(-\frac{\Delta}{T}\right) \tag{3.48}$$

and

$$\nu(P_{y|x}) = \arg\min_y F(P_{y|x}, y) = \check{\Theta}^{-1}\check{P}^T_{x|y}x \tag{3.49}$$

and let $y^\infty \in \mathbb{R}^{2M}$ *be a fixed point of the composition* $\nu \circ \rho : \mathbb{R}^{2M} \to \mathbb{R}^{2M}$. *Then* $\rho(y^\infty)$ *is a fixed point of* $\rho \circ \nu : R^{N\times M} \to R^{N\times M}$ *and* $(P^\infty_{y|x}, y^\infty)$ *is a minimizer of the master problem* (3.33).

Proof. Using the definition of fixed point we have $y^\infty = \nu \circ \rho(y^\infty)$, giving $\rho \circ \nu \circ \rho(y^\infty) = \rho(y^\infty)$, thus $P^\infty_{y|x} = \rho(y^\infty)$ is a fixed point of $\rho \circ \nu$. This implies that $P^\infty_{y|x} = \underset{P_{y|x}:P_{y|x}\geq 0, P_{y|x}\mathbb{1}_M=1}{\arg\min} F(P_{y|x}, y^\infty)$ and $y^\infty = \arg\min_y F(P^\infty_{y|x}, y)$. Due to convexity of subproblems these fixed points satisfy the KKT conditions. It can be trivially checked that the KKT condition for the master problem is the union of KKT conditions for the subproblems, implying that fixed points are also minimizers of the master problem. □

Alternating between solutions to (3.48) and (3.49) generates the recursions $P^{(n+1)}_{y|x} = \rho(y^{(n)})$, $y^{(n+1)} = \nu(P^{(n+1)}_{y|x})$ and non-increasing sequences [19] $F(y^{(n)})$, and $F(P^{(n)}_{y|x})$. Due to monotonicity of these series and existence of a lower bound for F according to Lemma 3.9, convergence of this procedure is guaranteed. This procedure in conjunction with annealing of parameter T is the basis of the ECP-LL algorithm in [57]. Essentially this algorithm alternates between the global optimum solutions of the two convex subproblems, to iteratively decrease the value of F.

Remark 1. *We note that a minimizer of the unconstrained assignment problem* $\min_{P_{y|x}\in\mathbb{R}^{N\times M}} F(P_{y|x}, y)$, *after normalization, is a minimizer of the assignment subproblem. The proof of this trivially follows by inspection of the KKT conditions in Theorem 3.8. That is, for implementation purposes, we can circumvent having to deal with constraints in the assignment subproblem, and view this as an unconstrained problem, completely eliminating having to handle constraints in the decomposition approach we propose.*

Apart from strict division of constraints, this problem partition serves other motives. Using basic convex calculus, for example as given in [68], it can be seen that F is component-wise convex with respect to $P_{y|x}$ and y. This motivates the idea of working with two convex subproblems in iteration as opposed to directly handling the master problem. Breaking the problem into convex partitions is a well-known optimization approach that can be seen in the Expectation Maximization algorithm in [69]. In a broader sense this procedure is widely known as *grouped coordinate descent*, where at each step optimization is performed only on a group of variables.

Lemma 3.9. *Existence of Lower bound: The energy function* F *in Equation* (3.32) *is a bounded smooth function. More explicitly we have that* $F(P_{y|x}, y) > -T\log M, \quad \forall P_{y|x} \in \mathbb{R}^{N\times M}, y \in \mathbb{R}^{2M}$

Proof. Smoothness trivially follows from continuity of its constituents. Using Equation (3.48) and plugging in the optimal value of $P_{y|x}$ in F we will get

$$F^*(y) = \sum_{i\in\mathcal{N}} p(x_i)\log\sum_{j\in\mathcal{M}}\exp\left(-\frac{\Delta_{ij}}{T}\right) \tag{3.50}$$

[19] With abuse of notation by $F(y)$, and $F(P_{y|x})$ we mean the series generated by altering only one of the arguments of F.

thus we have that

$$F^*(y) + \frac{1}{\beta}\log M = \frac{1}{\beta}\sum_{i\in\mathcal{N}} p(x_i)\log\frac{M}{z_i} > 0$$

The last inequality holds because $z_i < M$, $\forall i \in N$, by design. Noting that $F^*(y) \leq F(P_{y|x}, y)$, $\forall P_{y|x} \in \mathbb{R}^{N\times M}$ the desired result follows. □

With abuse of terminology we refer to minimizer of placement subproblem as "optimal", from here on. This is not haphazard as [50] has shown the complexity of CP lies in the placement part, because the optimal assignement is trivially found given optimal placement. This property aligns with the computational complexity analysis that we present later. Our end goal here is to design a control law u that continuously reduces value of F through tracking the *optimal* placements of controllers in the system. The first step in this direction is understanding the time derivative of the energy function F.

Lemma 3.10. *Given the energy function defined in (3.32), and the dynamical system (3.26), the time derivative of F has the following structure:*

$$\dot{F} = 2\zeta^T\Gamma(\zeta)f(\zeta), \quad \Gamma = \begin{bmatrix} \check{P}_x, & -\check{P}_{xy} \\ -\check{P}_{xy}^T & \check{P}_y\check{\Theta} \end{bmatrix} \tag{3.51}$$

Proof. Combining Equations (3.40), and (3.41) we can show that $\frac{\partial F}{\partial \zeta} = 2\zeta^T\Gamma$. Now by using the chain rule we have that $\dot{F} = \frac{\partial F}{\partial \zeta}\dot{\zeta}$ and the desired result follows. □

Lemma 3.11. *Positive definiteness of $\check{\Theta}$: Matrix $\check{\Theta} \in \mathbb{R}^{2M+2M}$ as defined in Theorem 3.7, is a positive definite matrix, namely $\check{\Theta} \succ 0$. Furthermore eigenvalues of $\check{\Theta}$, are $\lambda_1 = \gamma M + 1$, and $\lambda_2 = 2\gamma M + 1$, with their algebraic multiplicities being respectively of $2M-2$, and 2.*

.

Proof. We analyze the eigenvalues of $\check{\Theta}$ by equating its characteristic polynomial to zero.

$$p(\lambda) = det\left(\check{\Theta} - \lambda I\right) = \begin{vmatrix} (\eta-\lambda)I & -\gamma I & \dots & -\gamma I \\ -\gamma I & (\eta-\lambda)I & \dots & -\gamma I \\ \vdots & \vdots & \ddots & \vdots \\ -\gamma I & -\gamma I & \dots & (\eta-\lambda)I \end{vmatrix} = \gamma^{2M}\begin{vmatrix} \alpha I & I & \dots & I \\ I & \alpha I & \dots & I \\ \vdots & \vdots & \ddots & \vdots \\ I & I & \dots & \alpha I \end{vmatrix} \tag{3.52}$$

Where $\alpha = -\frac{\eta-\lambda}{\gamma}$. Using a similar triangularization procedure in [57] we have

$$\begin{bmatrix} \alpha I & I & \dots & I \\ I & \alpha I & \dots & I \\ \vdots & \vdots & \ddots & \vdots \\ I & I & \dots & \alpha I \end{bmatrix}\begin{bmatrix} I & 0 & \dots & 0 \\ \frac{-1}{\alpha+M-2}I & I & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{-1}{\alpha+M-2}I & \frac{-1}{\alpha+M-3}I & \dots & I \end{bmatrix} = \begin{bmatrix} \beta_1 I & \times & \dots & \times \\ 0 & \beta_2 I & \dots & \times \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \beta_m I \end{bmatrix}$$

Where $\beta_i = \alpha - \dfrac{M-i}{\alpha+M-i-1}$ thus $p(\lambda) = \gamma^{2M}\left(\prod_{i=1}^{M}\beta_i\right)^2 = \gamma^{2M}(\alpha-1)^{2M-2}(\alpha-M-1)^2$. Equating $p(\lambda)$ to zero yields the desired result. Note that despite having repeated eigenvalues, $\check{\Theta}$ is still orthogonally diagonalizable due to spectral theorem. □

Lemma 3.12. *Positive definiteness preservation: Pre or post multiplication by a diagonal positive definite matrix, preserves positive definiteness.*

Proof. Let $A, \Lambda \in \mathbb{R}^{n\times n}$ be positive definite matrices such that Λ is also a diagonal matrix. Due to positive definiteness of Λ, its square root and its inverse exists thus $\Lambda^{-1/2}$ is well defined. Let $A_\Lambda = \Lambda^{1/2} A \Lambda^{1/2}$, and we note that $x^T A_\Lambda x = (\Lambda^{1/2}x)^T A(\Lambda^{1/2}x) > 0$, due to positive definiteness of A, rendering A_Λ positive definite. since $A_\Lambda = \Lambda^{-1/2}(\Lambda A)\Lambda^{1/2}$, it is similar to $A\Lambda$ and shares its eigenvalues. The case ΛA trivially follows from transposition of $A\Lambda$. □

Here we revisit some of the previously defined expressions, albeit in an average sense, as they are going to be useful in the design of the control, for propelling network controllers. Let $\overline{x} = x - \check{P}_{y|x}\check{\Theta}^{-1}\check{P}_{x|y}^T x$ as the weighted average distance of network nodes from optimal placement of controllers. Similarly define $\overline{y} = y - \check{\Theta}^{-1}\check{P}_{x|y}^T x$ as the relative controller placements [20]. $\overline{\zeta}^T = [\overline{x}^T\ \overline{y}^T]$ denotes the stacked average positions of network and controller nodes. $\overline{u} = u - \check{\Theta}^{-1}\check{P}_{x|y}^T\varphi$, and $\bar{\varphi} = \check{P}_x\varphi$, will similarly, and respectively denote the relative acceleration of controllers and network nodes in the weighted average sense.

Lemma 3.13. *For a non-negative function $f : \mathbb{R} \to \mathbb{R}$ if improper integral $\int_0^\infty f(t)dt$ is convergent then* $\lim_{t\to\infty} f(t) = 0$

Proof. By way of contradiction assume $\lim_{t\to\infty} f(t) = L$ and $L \neq 0$. Then for the case where $L > 0$, using definition of limit, for some $M > 0$ we have $f(x) > L/2$ for $x \geq M$, thus we have $\int_0^\infty f(t)dt \geq \int_0^\infty Ldt$, contradicting convergence of the improper integral. The case $L < 0$ follows similarly. □

Theorem 3.14. *Control: With proper choice of control $\overline{u} \in \mathbb{R}^{2M+2N} \to \mathbb{R}^{2M}$, it is possible to make time derivative of energy function F negative, along the trajectory of System (3.26). More specifically choose $\overline{u}$ from the sets of the form*

$$\overline{\mathcal{U}}(\alpha) = \left\{ \overline{u}(\overline{\zeta}) = -\left[K_0 + \frac{\alpha(\overline{\zeta}) + \theta(\overline{\zeta})}{\overline{y}^T\check{P}_y\check{\Theta}\overline{y}}\right]\overline{y},\ \theta\left(\overline{\zeta}\right) > 0, K_0 > 0 \right\} \tag{3.53}$$

Parametrized by $\alpha : \mathbb{R}^{2M+2N} \to \mathbb{R}^{2M}$, then we have that $\dot{F}(t) < 0$, $\forall t \in \mathbb{R}^+$.

Proof. Using expansion of Equation (3.51) and algebraic manipulation we can show that

$$\dot{F} = \overline{x}^T\bar{\varphi} + \overline{y}^T\check{P}_y\check{\Theta}\overline{u} \tag{3.54}$$

Now by choosing $\overline{u} \in \overline{\mathcal{U}}(\alpha)$ and plugging into above equation we have that

$$\dot{F} = \overline{x}^T\bar{\varphi} - \alpha(\overline{\zeta}) - \theta(\overline{\zeta}) - K_0\overline{y}^T\check{P}_y\check{\Theta}\overline{y} \tag{3.55}$$

Letting $\alpha(\overline{\zeta}) = \overline{x}^T\bar{\varphi}$ we have $\dot{F} = -K_0\overline{y}^T\check{P}_y\check{\Theta}\overline{y} - \theta(\overline{\zeta}) < 0$, due to Lemmas (3.11), and (3.12) and assumption on non-degenerateness of the clusters. □

Although F does not satisfy the conditions for a conventional control Lyapunov function, we can use it to derive interesting stability results for the auxiliary system $\dot{\overline{y}} = \overline{u} - \check{\Theta}^{-1}\dot{\check{P}}_{x|y}^T x$.

[20] Relative to optimal placements $\check{\Theta}^{-1}\check{P}_{x|y}^T x$

Corollary 3.2. *Asymptotic stability of relative placements: The equilibrium point $\overline{y} = 0$, for the dynamical system (3.56), such that $\overline{u} \in \overline{\mathcal{U}}(\overline{x}^T \bar{\varphi})$ is globally asymptotically stable, i.e. $\forall \overline{y}(0) \in \mathbb{R}^{2M}$ we have $\lim_{t\to\infty} \overline{y}(t) = 0$*

$$\dot{\overline{y}} = \overline{u} - \check{\Theta}^{-1} \dot{\check{P}}_{x|y}^T x \tag{3.56}$$

Proof. System (3.26) is constrained within the compact set Ω by assumption, and the continuous real-valued function F is bounded from below, due to Lemma (3.9). Letting $\theta(\overline{\zeta}) = 0$, and $\alpha(\overline{\zeta}) = \overline{x}^T \bar{\varphi}$ in control law (3.53) we have $\dot{F} = -K_0 \overline{y}^T \check{P}_y \check{\Theta} \overline{y} < 0$. combined with boundedness of F we can deduce that $\int_0^\infty \dot{F}(\tau) d\tau = \lim_{t\to\infty} F(t) - F(0) < \infty$, and because this improper integral converges, then according to Lemma 3.13 we must have that $\lim_{t\to\infty} \dot{F}(t) = 0$. Since $\check{P}_y \check{\Theta}$ is positive definite, due to Lemmas (3.11), and (3.12) and assumption on non-degenerateness of the clusters, then we must have $\lim_{t\to\infty} \overline{y}(t) = 0$. □

Stability of the system $\dot{\overline{y}} = \overline{u} - \check{\Theta}^{-1} \dot{\check{P}}_{x|y}^T x$ in Corollary (3.2) implies that $\lim_{t\to\infty} y(t) = \check{\Theta}^{-1} \check{P}_{x|y} x$, analogous to Equation (3.49). This corollary is the crux of the RCP+ algorithm that is leveraged to track the optimal placement of controllers.

In the interim period when $\overline{y} \neq 0$ the placements are suboptimal and it is desirable to have a control that decreases this time period as much as possible. The following theorem states that the equilibrium point of the system (3.56) is exponentially stable.

Theorem 3.15. Exponential stability of the relative placements*: The equilibrium point $\overline{y} = 0$ is exponentially stable for the dynamical system (3.56), where $\overline{u} \in \overline{\mathcal{U}}(\overline{x}^T \bar{\varphi})$, with $\theta(\overline{\zeta}) = \left|\overline{x}^T \bar{\varphi}\right| + M_1 \|\overline{y}\| (\|\overline{x}\| + \eta \|\overline{y}\|)$ and $M_1 > 0$ is a bound on $\left\|\dot{\check{P}}_{x|y}\right\|_\infty$, and $\left\|\dot{\check{P}}_y\right\|_\infty$.*

Proof. Note that existence of M_1 is guaranteed due to Lipschitz continuity of $\check{P}_y$, and $\check{P}_{x|y}$. We claim that $V(\overline{y}) = \frac{1}{2} \overline{y}^T \check{P}_y \check{\Theta} \overline{y}$ is a Lyapunov function for the system (3.56). V is positive definite due to positive definiteness of $\check{P}_y \check{\Theta}$. Also

$$\dot{V} = \frac{1}{2} \left(\overline{y}^T \check{P}_y \check{\Theta} \dot{\overline{y}} + \dot{\overline{y}}^T \check{P}_y \check{\Theta} \overline{y} \right) + \frac{1}{2} \overline{y}^T \dot{\check{P}}_y \check{\Theta} \overline{y} \tag{3.57}$$

$$= -K_0 \overline{y}^T \check{\Theta} \check{P}_y \overline{y} - \left(\overline{x}^T \bar{\varphi} + \left|\overline{x}^T \bar{\varphi}\right| + M_1 \|\overline{y}\| (\|x\| + \eta \|\overline{y}\|) \right) - x^T \dot{\check{P}}_{x|y} \check{P}_y \overline{y} + \frac{1}{2} \overline{y}^T \dot{\check{P}}_y \check{\Theta} \overline{y} < 0 \tag{3.58}$$

Thus far we have shown that V is a Lyapunov function. Additionally

$$\dot{V} < -K_0 V \implies V(t) \leq \exp(-K_0 t) V(0) \tag{3.59}$$

Using Lemmas (3.18), (3.19), and (3.11) this implies that

$$\|\overline{y}\| \leq \sqrt{\frac{2\gamma M + 1}{\omega (\gamma M + 1)}} \exp\left(-\frac{K_0 t}{2} \right) \|\overline{y}(0)\| \tag{3.60}$$

where $\omega > 0$ is a lower bound on $\lambda_{min}(\check{P}_y)$. □

Theorem 3.15 indicates that time needed to reduce $\overline{y}$ to $\delta \|\overline{y}(0)\|$ is $t = 2 \left| \log \delta \sqrt{\frac{\omega (\gamma M + 1)}{2\gamma M + 1}} \right| / K_0$, for any $0 < \delta < 1$. It also indicates the sensitivity of settling time to the parameter K_0 in Expression (3.53), where if this time needs to be halved, then the value of K_0 needs to be doubled, showing that the interim period to settle in optimal controller placements can be made arbitrarily small. To gain even more control on the

convergence time it is entirely possible to recruit an annealing scheme for this parameter perhaps based on the value of $\overline{y}$, for example $K_0(t) = K_1 \left\|\overline{y}(t)\right\| + K_2$, such that $0 < K_1 < 1$, and $0 < K_2$, which precipitates convergence to optimal placements when $\|\overline{y}\|$ has large values and plateaus at K_2 as $\overline{y}$ converges to 0.

For the purpose of implementation, RCP works by starting with high values of temperature T, in Equation (3.32), and slow annealing of this parameter to $T = 0$. This ensures that the starting placements are highly non-committal towards any controller [21] and is slowly biased towards minimization of cost function D as temperature decreases. An interesting phenomenon known as *phase transition* ([37]) occurs as temperature is decreased which is going to be useful for determination of the optimal number of controllers.

Theorem 3.16. *Phase transition: Codebook $y \in \mathbb{R}^{2M}$ is no longer optimal when $\det\left[I - \frac{2}{T} C_{x|y_j}\right] = 0$, for some $j \in \mathcal{M}$, where $C_{x|y_j}$ is the covariance matrix of posterior distribution $P_{x|y}$ such that*

$$C_{x|y_j} = \sum_{i \in \mathcal{N}} p(x_i \mid y_j)(y_j^c - x_i)(y_j^c - x_i)^T \tag{3.61}$$

and $y_j^c = \sum_{i \in \mathcal{N}} p(x_i \mid y_j) x_i,\ \forall j \in \mathcal{M}$ is the weighted geometric center of jth cluster. This occurs when temperature is lowered to twice the variance along the principal axis of a cluster.

Proof. Let F^* equal to Expression (3.50) the necessary and sufficient condition for optimality of F^* with respect to y can be defined as

$$\frac{d}{d\epsilon} F^*(y + \epsilon\Psi) \mid_{\epsilon=0} = 0 \quad \frac{d^2}{d\epsilon^2} F^*(y + \epsilon\Psi) \mid_{\epsilon=0} \geq 0 \tag{3.62}$$

Where $\Psi \in \mathbb{R}^{2M} = [\psi_1^T, \psi_2^T, \ldots, \psi_M^T]$ is a perturbation vector with $\psi_j \in \mathbb{R}^2,\ \forall j \in \mathcal{M}$. The first derivative condition will lead to Expression (3.49), and the second derivative [22] condition with straightforward differentiation produces the following equation.

$$\sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} p(x_i, y_j) \psi_j^T \left[I - \frac{2}{T} (\eta y_j - \gamma \sum_{\substack{j' \in \mathcal{M} \\ j' \neq j}} y_{j'} - x_i)(\eta y_j - \gamma \sum_{\substack{j' \in \mathcal{M} \\ j' \neq j}} y_{j'} - x_i)^T \right] \psi_j$$
$$+ \sum_{i \in \mathcal{N}} p(x_i) \left[\sum_{j \in \mathcal{M}} p(y_j \mid x_i)(y - \gamma \sum_{\substack{j' \in \mathcal{M} \\ j' \neq j}} y_{j'} - x_i)^T \psi_j \right]^2 = 0$$

which can be further simplified to

$$\sum_{j \in \mathcal{M}} p(y_j) \psi_j^T \left[I - \frac{2}{T} C_{x|y_j} \right] \psi_j + \sum_{i \in \mathcal{N}} p(x_i) \left[\sum_{j \in \mathcal{M}} p(y_j \mid x_i)(\check{\Theta}_j^T y - x_i)^T \psi_j \right]^2 = 0 \tag{3.63}$$

□

Where $\check{\Theta}_j$ is the jth row of $\check{\Theta}$. By way of contradiction assume left hand side of Equation (3.63) is positive and its first term is not. This means $\exists j \in \mathcal{M}$, such that $I - \frac{2}{T} C_{x|y_j}$ is not positive definite, and we

[21] Note distribution of $p(y_j \mid x_i)$ will be uniform for $T \to \infty$, in Equation (3.48).
[22] Bifurcation occurs when second derivative is equal to zero, where Hessian straddles optimality and non-optimality.

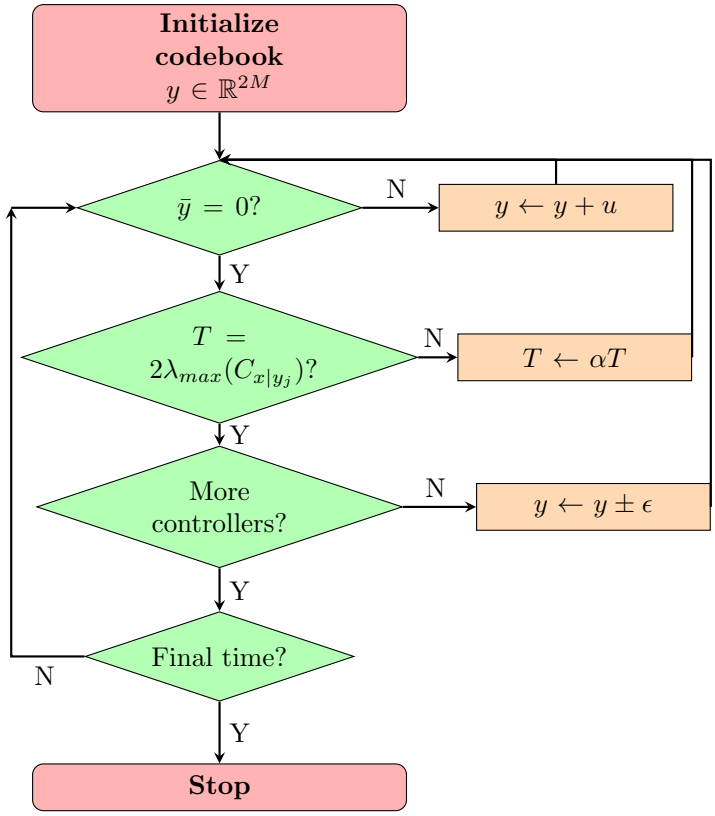


Figure 3.5: Codebook expansion procedure.

can construct Ψ with $\psi_{j'} = 0 \; \forall j' \in \mathcal{M}$, such that $y_{j'} \neq y_j$, and $\sum_{\substack{j' \in \mathcal{M} \\ y_{j'} = y_j}} \psi_{j'} = 0$. This perturbation makes the second term vanish, and rendering the left hand side non-positive. Thus left hand side is positive if and only if the first term is positive, and phase transition occurs when for some $j \in \mathcal{M}$, $det\left[I - \frac{2}{T} C_{x|y_j}\right] = 0$, and the critical temperature lowered to $T_c = 2\lambda_{max}(C_{x|y_j})$, or the largest eigenvalue of covariance matrix.

Theorem 3.16 is useful in determining when to expand to codebook when current number of controllers are not optimal. Finding $\lambda_{max}(C_{x|y_j})$ could be computationally demanding for larger sized networks and instead it is possible to follow the procedure in [37] and *perturb* the centroids by generating the the perturbation vector $\epsilon \in \mathbb{R}^M$ and replacing each centroid y with $y \pm \epsilon$ at each iteration. It can be shown that when codebook size is optimal, the newly generated centroids will collapse in one point, and if not $y \pm \epsilon$ values diverge, and a new centroid is automatically added to codebook.

In order to establish Lipschitz continuity of the control law (3.53), we will impose a mild assumption on the consistency of clusters, similar to [19]. This assumption implies *average* distance between network nodes and controller positions do not increase with time, namely $\overline{x}^T \overline{\varphi} \leq 0$. This also implies that network nodes, *on average* are "pulled" towards the cluster and not away from it.

Theorem 3.17. *Lipschitz continuity of control: If the assumption on the consistency of clusters holds ($\overline{x}^T \overline{\varphi} \leq 0$), with choice of $\overline{u}(\overline{\zeta}) \in \overline{\mathcal{U}}(\overline{x}^T \overline{\varphi})$, and $\theta(\overline{\zeta}) = \left|\overline{y}^T \check{P}_y \check{\Theta} \overline{y}\right|^2$ in Expression (3.53) then the control $u = \overline{u} - \check{\Theta}^{-1} \check{P}_{x|y}^T \varphi$ for propelling the network controllers becomes Lipschitz Continuous, and bounded.*

$$\overline{u}(\overline{\zeta}) = -\left[K_0 + \frac{\overline{x}^T\bar{\varphi} + \sqrt{\left|\overline{x}^T\bar{\varphi}\right|^2 + \left|\overline{y}^T\check{P}_y\check{\Theta}\overline{y}\right|^2}}{\overline{y}^T\check{P}_y\check{\Theta}\overline{y}}\right]\overline{y}$$

More precisely for $\overline{u}(\overline{\zeta})$ as defined above, we have $\left\|\overline{u}(\overline{\zeta})\right\| \leq (1+K_0)\left\|\overline{\zeta}\right\|$, $\forall\overline{\zeta} \in \mathbb{R}^{2M+2N}$

Proof. The proof herein is analogous to Proposition 3.43 in [59], and Theorem 4 in [19]. Note that $\check{\Theta}$ has bounded norm and Lipschitz continuity of $\check{P}_{x|y}$ trivially follows from that of softmax function. Also note that φ is Lipschitz, and bounded by assumption. Thus we just need to prove Lipschitz continuity, and boundedness of $\overline{u}_S$ to achieve the same properties for $u = \overline{u} + \check{\Theta}^{-1}\check{P}_{x|y}^T\varphi$. For any $a, b \in \mathbb{R}$ such that $a \leq 0$, and $b > 0$, we have $0 \leq a + \sqrt{a^2+b^2} \leq b$, implying that

$$K_0 + \frac{\overline{x}^T\bar{\varphi} + \sqrt{\left|\overline{x}^T\bar{\varphi}\right|^2 + \left|\overline{y}^T\check{P}_y\check{\Theta}\overline{y}\right|^2}}{\overline{y}^T\check{P}_y\check{\Theta}\overline{y}} \leq 1 + K_0$$

Which means

$$\left\|\overline{u}(\overline{\zeta})\right\| \leq \left(K_0 + \frac{\overline{x}^T\bar{\varphi} + \sqrt{\left|\overline{x}^T\bar{\varphi}\right|^2 + \left|\overline{y}^T\check{P}_y\check{\Theta}\overline{y}\right|^2}}{\overline{y}^T\check{P}_y\check{\Theta}\overline{y}}\right)\|\overline{y}\| \leq (1+K_0)\left\|\overline{\zeta}\right\|$$

establishing Lipschitz continuity of $\overline{u}$. Let $r_\Omega = \min\{r > 0 \mid \Omega \subset \mathcal{B}_r\}$, where $\mathcal{B}_r = \{p \in \mathbb{R}^2 \mid \|p\| \leq r, r > 0\}$ as the minimum radius of the smallest enclosing ball around Ω, which is well-defined due to compactness of Ω. Note that $\|\overline{y}\| \leq r_\Omega\left(1 + \sqrt{M}\left\|\check{\Theta}^{-1}\right\|\right)$, by definition thus $\|\overline{u}\| \leq r_\Omega(1+K_0)\left(1+\sqrt{M}\left\|\check{\Theta}^{-1}\right\|\right)$ □

Lemma 3.18. Bounds on quadratic form: *For quadratic form x^TAx, where $x \in \mathbb{R}^n$, and $A \in \mathbb{R}^{n\times n}$ is a symmetric matrix, the inequality $\lambda_{min}(A)\|x\|^2 \leq x^TAx \leq \lambda_{max}(A)\|x\|^2$ holds where $\lambda_{min}(A)$, and $\lambda_{max}(A)$ are respectively the smallest and largest eigenvalues of matrix A.*

Proof. Because A is symmetric, therefore it is orthogonally diagonalizable due to spectral theorem. Thus $\exists\, Q, \Lambda \in \mathbb{R}^{n\times n}$, where Q is an orthogonal, and Λ is a diagonal matrix, such that $A = Q^T\Lambda Q$. Note that

$$\frac{x^TAx}{\|x\|^2} = \|\Lambda^{1/2}Q\frac{x}{\|x\|}\|^2 = \sum_{i=1}^{N}\lambda_i\left|z_i\right|^2$$

Where z_i, λ_i are elements of respectively $Q\frac{x}{\|x\|}$, and Λ, and $\sum_{i=1}^n\left|z_i\right|^2 = 1$, due to orthonormality of Q. Thus the right hand side achieves its maximum value for $\lambda_i = \lambda_{max}(A)$, and its minimum for $\lambda_i = \lambda_{min}(A)$ for $i = 1, 2, \ldots, n$, yielding the desired result. □

Lemma 3.19. *Bound on eigenvalues: For $A, B \in \mathbb{R}^{n\times n}$, such that $A, B \succ 0$ we have $\lambda_{min}(A)\lambda_{min}(B) \leq \lambda_{min}(AB)$ and $\lambda_{max}(AB) \leq \lambda_{max}(A)\lambda_{max}(B)$.*

Proof. For symmetric matrices, matrix norm equals the spectral radius thus

$$\lambda_{max}(AB) = \|AB\| \leq \|A\|\,\|B\| = \lambda_{max}(A)\lambda_{max}(B)$$

Applying the above inequality to inverse of AB

$$\lambda_{max}((AB)^{-1}) \leq \lambda_{max}(A^{-1})\lambda_{max}(B^{-1})$$

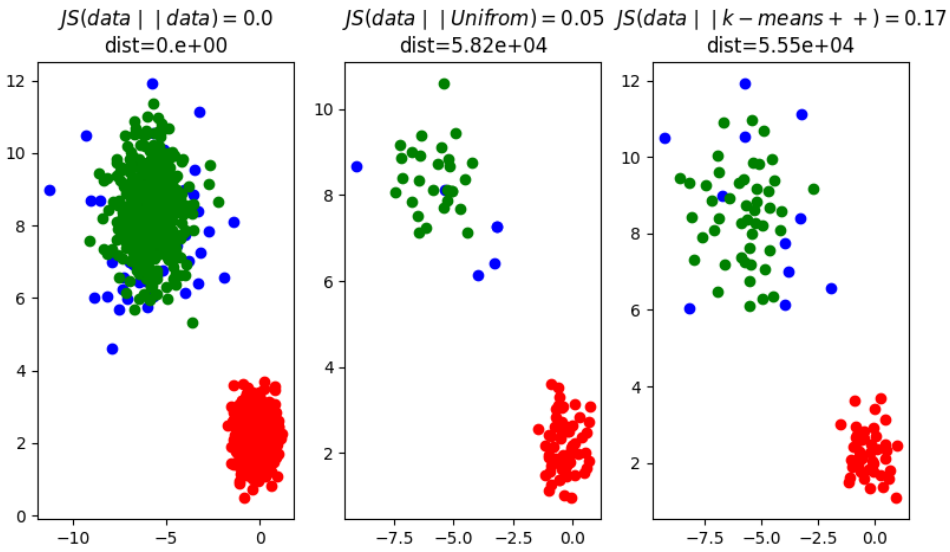


Figure 3.6: Sparse sampling versus uniform sampling.

Noting that eigenvalues of matrix inverse are inverse of eigenvalues

$$\frac{1}{\lambda_{min}(AB)} \leq \frac{1}{\lambda_{min}(A)} \frac{1}{\lambda_{min}(B)} \implies \lambda_{min}(A)\lambda_{min}(B) \leq \lambda_{min}(AB)$$

□

In order to expedite RCP+ for extremely large networks we propose a randomized algorithm that returns a subset of the network of smaller size that is yet representative of the original network. This sampling can be done at regular intervals to ensure the adherence to current state of the network. More specifically we recruit the K-means++ algorithm from [70] that is originally used to seed the celebrated K-means algorithm. The main logic behind this algorithm is to iteratively pick random nodes from the network such that at each iteration nodes furthest away from already chosen nodes are more likely to be chosen, ensuring the subset is maximally sprawling across the original network, capturing a sparse, yet faithful picture of the dataset. In order to efficiently implement the multidimensional proximity search we propose k-d trees from [71] that is essentially a multidimensional binary search tree to store node positions, with an average lookup complexity of $\mathcal{O}(\log n)$, for a k-d tree of size n. Sampling also should be done frequently enough to ensure adherence to current state of network and infrequently enough so that the computational overhead of sampling does not dominate the overall complexity of RCP+ as this procedure can be computationally demanding.

Figure 3.6 shows the result of sampling from a synthetic dataset of three color coded clusters with two features. The result for uniform sampling adheres better to the original dataset in terms of preserving ratio of cluster sizes as can be seen from comparing Jensen-Shannon divergence for uniform sampling versus sparse sampling. However even though uniform sampling represents the original dataset in terms of cluster size ratios, it misrepresents it spatially. The right diagram shows that sparse sampling adheres better to the "shape" or spatial distribution of the clusters with the elongated shape of the green cluster, and the fanned out shape of the blue cluster better reconstructed. Using the distance metric [23] $dist(A, B) = \|A^T A - B^T B\|^2$

[23] See [72], for a motivation of this.

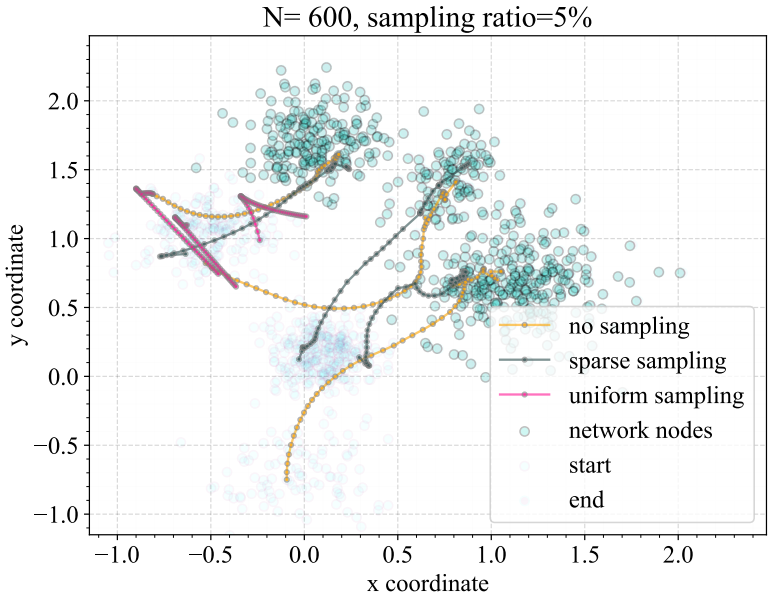


Figure 3.7: Sparse versus unifrom sampling trajectories

for two matrices of equal columns but different rows, we can validate the aforementioned fact by observing that the the sparse sampling has a smaller distance to the original dataset. In the context of our problem, for any sampling method, adherence to spatial distribution of the original dataset is critical, because the pair-wise distances between nodes and controllers propagate into overall network delay.

This evidently can be seen in Figure 3.7 where the uniform sampling leads to a subset that is not representative of the full data, and initialization within that subset leads to completely derailed trajectories. On the contrary, and despite the very low sampling ratio the trajectories generated using sparse sampling slowly, but surely merge with the full data path.

Algorithm 4: SparseSampling

Initialize:

network $G(\{\mathcal{N}, \mathcal{M}\}, \mathcal{E})$, network subset $\bar{\mathcal{N}} = \emptyset$ and sample size $\bar{N} \in \mathbb{N}$, and empty k-d tree

for $n = 1$ **to** $\bar{N}$ **do**

  if $n == 1$ **then**

    Select random node $i \in \mathcal{N}$, using uniform distribution

  else

    $d_i = \arg\min_{j \in \bar{\mathcal{N}}} d(x_i, x_j)$, $\forall i \in \mathcal{N} - \bar{\mathcal{N}}$

    $p_i = \frac{d_i}{\sum_{j \in \mathcal{N}} d_j}$, $\forall i \in \mathcal{N} - \bar{\mathcal{N}}$

    Select random node $i \in \mathcal{N} - \bar{\mathcal{N}}$, using discrete distribution p_i.

  end

  Update network subset:

  $\bar{\mathcal{N}} \leftarrow \bar{\mathcal{N}} \cup \{i\}$

  Add $x_i \in \mathbb{R}^2$ to k-d tree of $\bar{\mathcal{N}}$

end

Algorithm 5: RCP+

Initialize:

network $G(\{\mathcal{N}, \mathcal{M}\}, \mathcal{E})$, sampling interval $K \in \mathbb{N}$, velocity parameter $K_0 \in \mathbb{R}^+$ codebook $y \in \mathbb{R}^{2M}$, starting temperature $T \approx \infty$, minimum temperature T_{min}, time horizon $\tau \in \mathbb{N}$, and decay rate $\alpha \in (0, 1)$;

for $t = 1$ **to** τ **do**

Sample network every K steps:

if $t\%K == 0$ **then**

$G(\{\bar{\mathcal{N}}, M\}, \bar{\mathcal{E}}) \leftarrow SparseSampling\left(G(\{\mathcal{N}, \mathcal{M}\}, \mathcal{E})\right)$;

Compute pairwise distance and distortions:

Δ_{xy}, Δ_{yy}, Δ, $\check{\Theta}$, $\check{\Theta}^{-1}$.

update association weights using Gibbs distribution:

$P_{y|x} \leftarrow Z^{-1} \exp\left(-\frac{\Delta}{T}\right)$;

Update $\overline{y}$:

$\overline{y} \leftarrow y - \check{\Theta}^{-1} \check{P}_{x|y}^T x$

Compute $\overline{u}$ using control law (3.53):

$\overline{u}(\bar{\zeta}) = -\left[K_0 + \frac{\alpha(\bar{\zeta}) + \theta(\bar{\zeta})}{\overline{y}^T \check{P}_y \check{\Theta} \overline{y}}\right] \overline{y}$

compute velocity estimate $\widetilde{\varphi}$:

$\widetilde{\varphi} = PredictNetwork\left(G(\{\bar{\mathcal{N}}, \mathcal{M}\}, \bar{\mathcal{E}})\right)$

update φ:

$\varphi \leftarrow \widetilde{\varphi}$

Update control u:

$u \leftarrow \overline{u} + \check{\Theta}^{-1} \check{P}_{x|y}^T \varphi$

Update controller positions y:

$y \leftarrow y + u$

if $T \geq T_{min}$ **then**

update system temperature:

$T \leftarrow \alpha T$;

end

We assume that for real world networks $N >> M$, namely network nodes by far outnumber the controller nodes. Given this we analyze steps in RCP+ to compute the cumulative iteration complexity. We do not take into account the sampling step as it is carried out sporadically. We assume the computational complexity of the prediction network [24] is negligible compared to other steps, due to low dimensionality of data that results in a lightweight prediction model. Moreover sampling and prediction can be also done asynchronously, although the overall effect on complexity is already nominal. At every iteration t we have to we perform the following operations. In line (4) basic floating point operation counting shows Δ_{xy} can be computed in $\mathcal{O}(MN)$, Δ_{yy} in $\mathcal{O}(M^2)$, Δ in $\mathcal{O}(MN)$, $\check{\Theta}$ in $\mathcal{O}(M^2)$, and its inverse in $\mathcal{O}(M^3)$. Note that Δ can be naively computed in $\mathcal{O}(M^2N)$ by directly computing $\mathbb{1}_{N\times M}\Delta_{yy}$, however using the fact that it has repeated rows it can be computed more efficiently. Thus the total will be dominated by $\mathcal{O}(MN + M^3)$. At line (5) computing the Gibbs distribution requires $\mathcal{O}(MN)$ floating point operations. In line (6), $\overline{y}$ can be computed naively in $\mathcal{O}(M^2N)$ corresponding to left to right multiplication in term $\check{\Theta}^{-1}P_{x|y}^T x$ or in $\mathcal{O}(M^2 + MN)$, for the right to left multiplication. In line (7) $\overline{u}$ can be computed in $\mathcal{O}(M^2)$, and complexity of line (9)

[24] LSTM recurrent neural netowrk in this case.

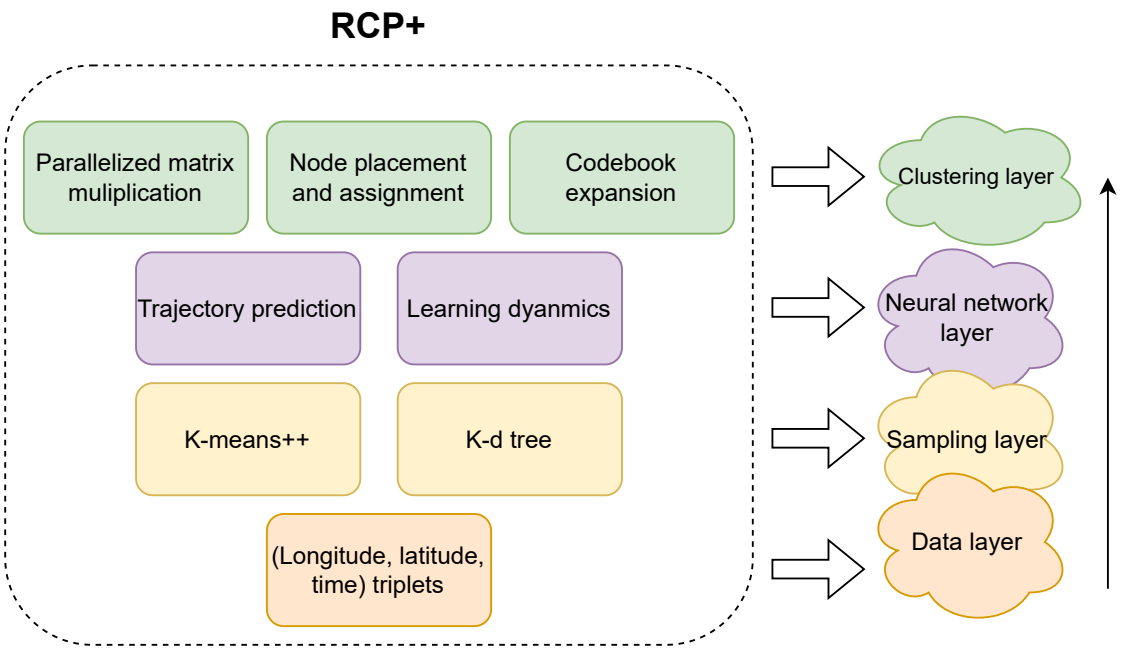


Figure 3.8: Layers of RCP+ algorithm.

is $\mathcal{O}(MN)$ identical to line (6) and finally updating y at line (10) can be done in $\mathcal{O}(M)$. With this we report the iteration computational complexity as $\mathcal{O}(M^2N + M^3)$ which is dominated by the term $\mathcal{O}(M^2N)$ that is linear in terms of network size N. On a separate matter, for step (6) namely computation of the Gibbs distribution, it is entirely possible to run into numeric underflow, especially at low T leading to all probabilities being set to zero, which is the natural behavior of programming implementation of $exp(x)$ for $x \to -\infty$. To avoid this we use the fact that row-wise shifting of values in distortion matrix Δ won't affect the Gibbs distribution. This can be trivially checked from insensitivity of softmax function to shifting of inputs, namely for $\sigma(z)_i = \frac{\exp z_i}{\sum_{j \in \mathcal{M}} \exp z_j}$, $z = \{z_i\}_{i=1}^M$, $\sigma(z) = \{\sigma_i\}_{i=1}^M$ we have $\sigma(z + c\mathbb{1}_M) = \sigma(z)$ for any $c \in \mathbb{R}$. Exploiting this fact, we subtract the maximum value of each row from the entire row making sure the row always has at least one 0 element (corresponding to maximum value prior to shifting), thus circumventing the numeric underflow issue.

The computations in RCP+ can be decomposed into independent partitions, amenable to parallel computing. This includes all the assignement steps, and also matrix multiplications. In addition to that, inference using a Recurrent Neural Network (RNN) if used for network prediction, is also parallelizable. Given this, we are able to use the power of modern GPU's to dramatically enhance the speed of RCP+. We acknowledge that the speed difference is nominal for smaller sized networks, and computation gain from parallelization is significant for large-scale networks.

Trajectory prediction

One major drawback of RCP was to assume there is prior information [25] available on the dynamics of the mobile network nodes. In this work we introduce a trajectory estimation procedure that we incorporate as a subroutine in the RCP+ algorithm and alleviate this shortcoming. To provide a bound on controller

[25] Namely φ in equation (3.26) is known.

acceleration error, assume $\widetilde{\varphi}$ is an approximation of the function φ, such that $\forall t \in \mathbb{R}^+$, $\|\widetilde{\varphi}(t) - \varphi(t)\| \leq \delta$, for some $\delta > 0$. Following the control law (3.53), choose $\overline{u} \in \overline{\mathcal{U}}(\overline{x}^T \overline{\varphi})$ and $\bar{\tilde{u}} \in \overline{\mathcal{U}}(\overline{x}^T \bar{\tilde{\varphi}})$, such that $\bar{\tilde{\varphi}} = \check{P}_x \widetilde{\varphi}$, and set $\theta(\overline{\zeta}) = 0$, for both of these controls. We have that

$$\left\|\overline{u} - \bar{\tilde{u}}\right\| \leq \frac{\left\|\overline{x}^T(\bar{\varphi} - \bar{\tilde{\varphi}})\right\|}{\left|\overline{y}^T \check{P}_y \check{\Theta} \overline{y}\right|} \|\overline{y}\| \leq \frac{\|\overline{x}\|\,\delta}{\lambda_{min}(\check{P}_y \check{\Theta})\,\|\overline{y}\|} \leq \frac{\sqrt{N}\delta}{\nu\,(\gamma M + 1)} \tag{3.64}$$

Where $\nu > 0$ is a bound on $\lambda_{min}(\check{P}_y)$, indicating the controller acceleration error can grow linearly with square root of network size. While this is not the tightest possible bound on relative acceleration error, it turns out, and we practically verify that final controller placements are not highly sensitive to errors in predictions of node velocities. Thus far we have assumed that the vector field $\varphi(t)$, is known, however for real world purposes it is necessary to approximate this function based on network nodes trajectory history.

Popular methods for trajectory prediction include system identification, Kalman filtering, linear projection, and most recently recurrent neural networks, among others. Besides being a modern approach neural networks are a natural choice since inference can be parallelized consistent with other steps of the RCP+. In order to design a trajectory prediction model we use a multilayer LSTM cell implemented in PyTorch 2.0 [73]. For convenience, we may overload certain symbols that were previously used in problem statement to describe the architecture of the neural network. Consider input sequence $\{x^{\langle t \rangle}\}_{t=1}^{T}$ for each element $x^{\langle t \rangle} \in \mathbb{R}^n$ in input layer following recursions are performed.

$$i^{\langle t \rangle} = \sigma\left(W_{ii} x^{\langle t \rangle} + b_{ii} + W_{hi} h^{\langle t-1 \rangle} + b_{hi}\right) \tag{3.65}$$

$$f^{\langle t \rangle} = \sigma\left(W_{if} x^{\langle t \rangle} + b_{if} + W_{hf} h^{\langle t-1 \rangle} + b_{hf}\right) \tag{3.66}$$

$$g^{\langle t \rangle} = \tanh\left(W_{ig} x^{\langle t \rangle} + b_{ig} + W_{hg} h^{\langle t-1 \rangle} + b_{hg}\right) \tag{3.67}$$

$$o^{\langle t \rangle} = \sigma\left(W_{io} x^{\langle t \rangle} + b_{io} + W_{ho} h^{\langle t-1 \rangle} + b_{ho}\right) \tag{3.68}$$

$$c^{\langle t \rangle} = f^{\langle t \rangle} \circ c^{\langle t-1 \rangle} + i^{\langle t \rangle} \circ g^{\langle t \rangle} \tag{3.69}$$

$$h^{\langle t \rangle} = o^{\langle t \rangle} \circ \tanh\left(c^{\langle t \rangle}\right) \tag{3.70}$$

$h^{\langle t \rangle} \in \mathbb{R}^h$ is the hidden state, $c^{\langle t \rangle} \in \mathbb{R}^h$ is the cell state, at step t. $i^{\langle t \rangle}$, $f^{\langle t \rangle}$, $g^{\langle t \rangle}$, $o^{\langle t \rangle} \in \mathbb{R}^h$ are respectively input, forget, cell and output gates. $\sigma : \mathbb{R}^h \to \mathbb{R}^h$ is the Sigmoid function and, similar to $\tanh : \mathbb{R}^h \to \mathbb{R}^h$ is applied component-wise. In a multilayer LSTM input at step t for layer l is $x_l^{\langle t \rangle}$ and is equal to hidden layer at step t for previous layer, namely $h_{l-1}^{\langle t \rangle}$. This is equivalent to stacking cells on top of each other.

We define input element $x^{\langle t \rangle} \in \mathbb{R}^2$ as the latitude and longitude of a node at step t. In order to predict velocity φ, at step T we use $y^{\langle T \rangle} = x^{\langle T+1 \rangle} - x^{\langle T \rangle}$ as the label, for the training sequence $\{x^{\langle t \rangle}\}_{t=1}^{T}$ of length T. Furthermore we used a multilayer LSTM model stacked with a fully connected network as shown in Figure 3.10 to return predictions of velocity $\hat{y}^{\langle t \rangle} \in \mathbb{R}^2$. We used Mean Squared Error (MSE), namely $l\left(y^{\langle t \rangle}, \hat{y}^{\langle t \rangle}\right) = \|y^{\langle t \rangle} - \hat{y}^{\langle t \rangle}\|^2$ as the loss function to learn dynamics of the network. Because the dynamics are dataset dependent we train a separate model for each dataset. This is equivalent to network learning the lay of the land and streets in each dataset. Provided with enough examples the network can reasonably well predict the next position of a node along the learned path of a street.

We used a ratio of 64% for training, 16% for validation and 20% for the test dataset. We used the training dataset to train the model, validation to tune the hyperparameters such as sequence length, number of LSTM

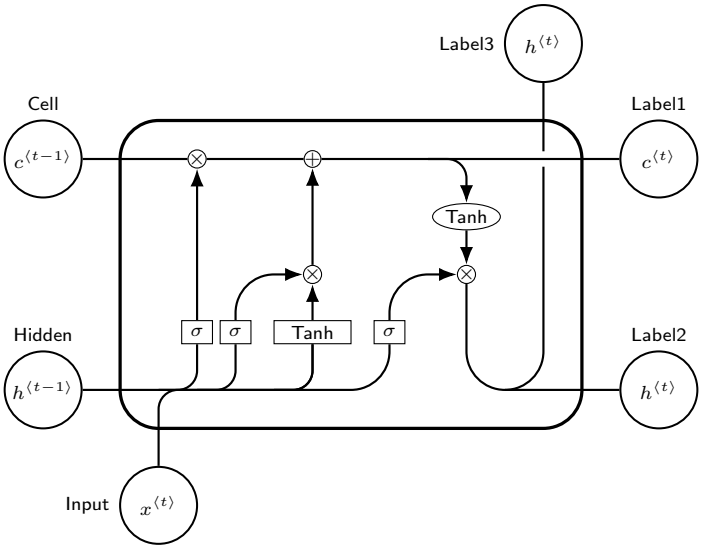


Figure 3.9: Conventional LSTM cell.

Dataset	Before preprocessing	After preprocessing	Train	Validation	Test	Sequence length	LSTM layers	Fully connected layers
Cabspotting [74]	11.2 millions	5.3 millions	3.39 millions	848,000	1.06 millions	5	3	(3,2)
T-Drive [75]	17.7 millions	4.12 millions	2.63 millions	672,000	824,000	4	4	(5,3,2)
UCI [76]	18,107	10,203	6,529	1,632	2,040	2	N/A	(4,2)

Table 3.1: Neural network architectures for node trajectory prediction.

layers and architecture of the fully connected part. Finally we used the test dataset that the model is blind to, to evaluate the best model. Training the model can be done asynchronously, carried out at planned junctures or more expensively and for smaller sized networks, in an online fashion.

The node positions in mobility datasets are collected asynchronously and the time between position updates are not necessarily always uniform. This can prevent RCP+ from directly being applied to these datasets since firstly RCP+ requires synchronous update on positions of the network nodes. Moreover if a trajectory has highly non-homogenous time intervals between position updates, this can lead to "confusion" of the LSTM model. To better understand this consider a trajectory with three points, where the time interval between the first two positions are in milliseconds and between second and third, an hour; clearly the positions of the first two points are much closer to each other than would be for the third point. Because this temporal information is not supplied to the model and in some sense is embedded in the order of element in a sequence, uniformity of time intervals thus is necessary for the LSTM model to work properly. In order to select trajectories with this property, for each trajectory we compute coefficient of variation $CV = \frac{\sigma}{\mu}$, where μ is the average of time intervals, and σ is the standard deviation. Based on this metric we discard any trajectory with $CV \geq \beta$, where β is a dataset dependent constant, to ensure some level of uniformity in the training dataset. β values for Cabspotting, UCI, and T-drive datasets are respectively, 0.1, 0.1, and 0.3. From practice a lower threshold would be more strict but rejects more trajectories that could otherwise be still useful for the purpose of training.

Figure 3.10 shows the general architecture of the model that we used to learn the dynamics of the datasets. This essentially is a multilayered LSTM attached to a fully connected network. From prior knowledge of low dimensionality of input and comparison with similar tasks, we know a smaller network will fit the dataset

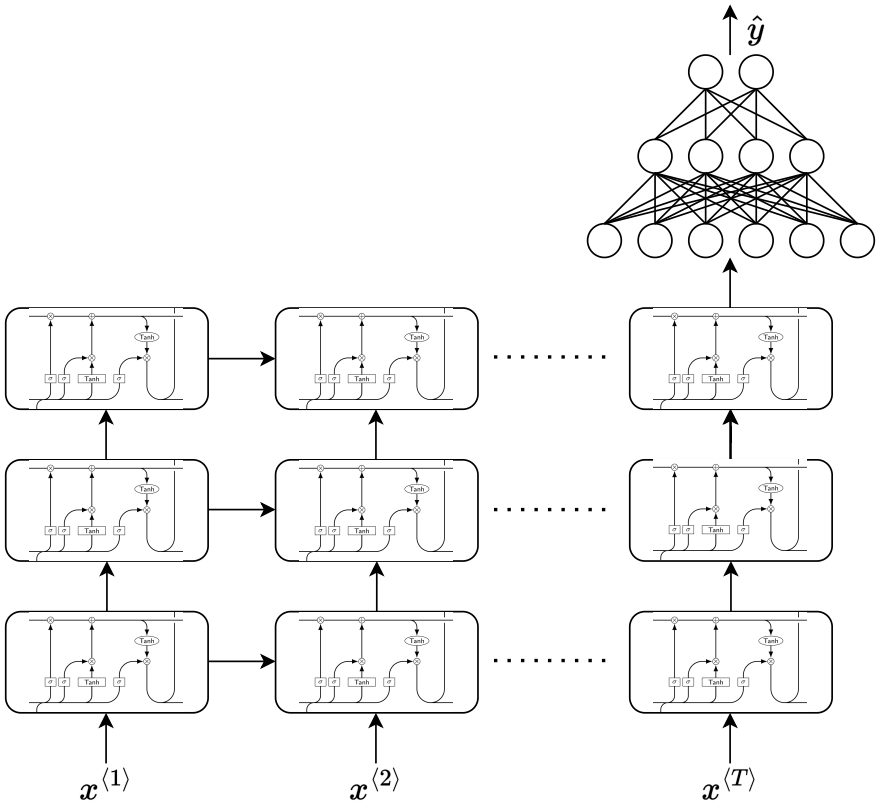


Figure 3.10: Three layer LSTM network attached to a three layer fully connected network.

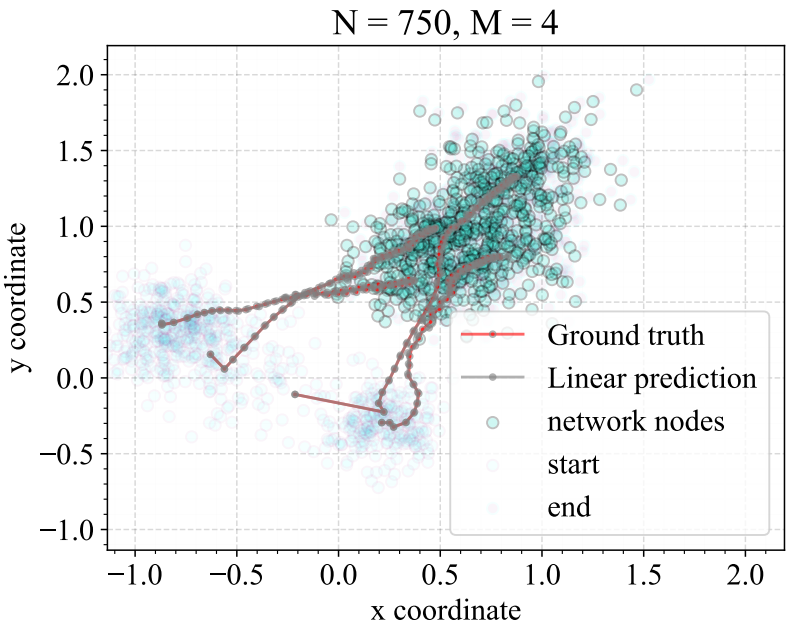


Figure 3.11: Overlapping trajectories of RCP+ for using ground truth velocities versus linear prediction of velocities.

correctly and the hyperparameters such as number of recurrent layers and number of layers and neurons in the attached fully connected network is selected using a randomized search, combined with cross-validation [77]. Unlike Cabspotting and T-drive datasets we recruit a linear model with a sequence length of 2 to learn the dynamics of the UCI dataset. This is expected as this dataset has a noticeably smaller size, and is prone to more complicated models overfitting this dataset.

It turns out that highly accurate estimates of node velocities are not necessary for the RCP+ to work properly. Our observation shows that as long as the individual predictions sum up to the correct direction of movement of clusters, then the placements stay reasonably close to that of ground truth. This can be evidently seen in Figure 3.11 where the dynamics [26] are simulated using a first order linear system with random noise. Due to addition of noise, nodes show chaotic behavior on the micro level, however they collectively move in a common direction as part of the clusters. We observe that even at this level of unpredictability using a crude linear prediction with a sequence length of one in lieu of the ground truth value of node velocities, results in nearly identical controller trajectories. This verifies the fact that even though more accurate velocity predictions lead to better placements, the trajectories prediction however is not a performance bottleneck for RCP+. More concretely controller placements $y \in \mathbb{R}^{2M}$ computed using RCP+ are not highly sensitive to node velocities $\varphi \in \mathbb{R}^{2N}$.

[26] Refer to Chapter 4 for more information

Chapter 4

Results

In this section, we evaluate the performance of our introduced controller placement algorithms. Through a rigorous bench-marking process, we compare our algorithms against competing alternatives using synthetic and real-world network datasets.

4.1 Static Setting

In order to evaluate the performance of ECP-LL and ECP-LB algorithms we compare their final costs with the integer programs (3.1)-(3.5) and (3.6)-(3.11). We use the state-of-the-art MINLP solver BARON to draw this comparison. We used Gaussian distribution to generate our data with K as the number of Gaussian clusters within the data. During the implementation we perform a grid search over the hyper-parameter space of K_{max} to find its optimum value.

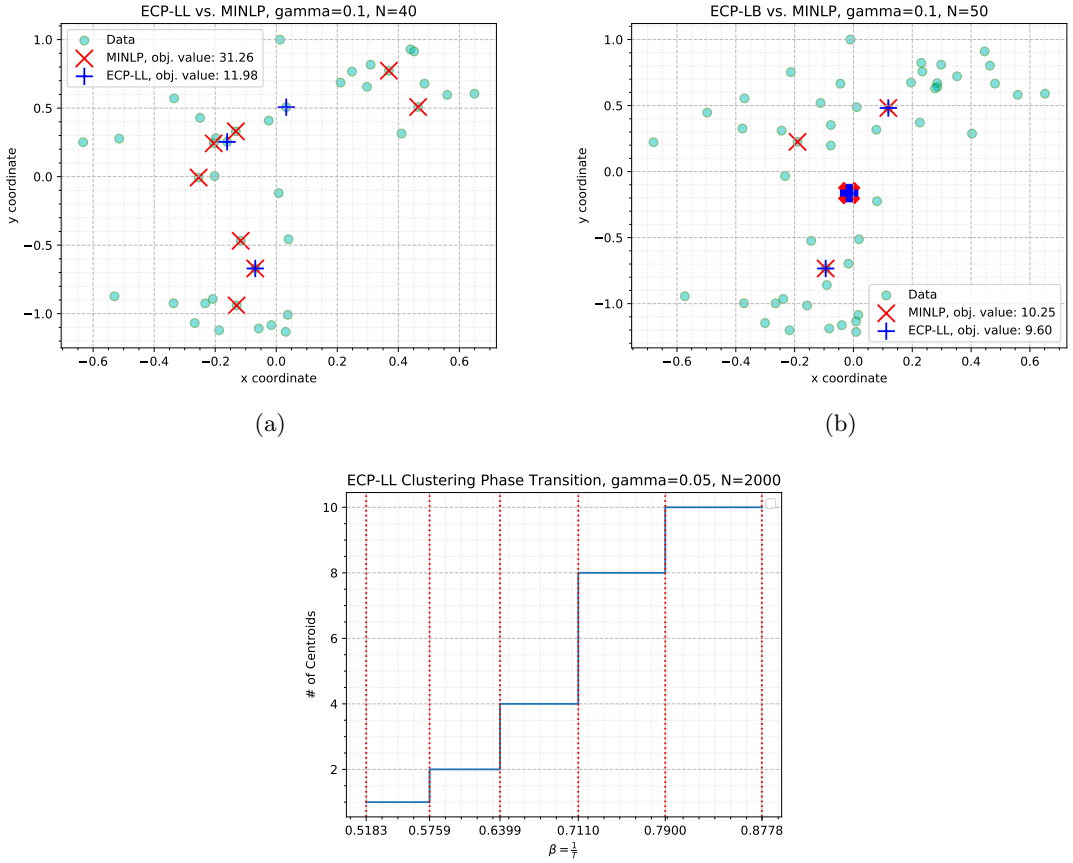


Figure 4.1: (a) ECP-LL vs. MINLP (b) ECP-LB vs. MINLP (c) Phase transition phenomenon

Superior performance of ECP DA-based clustering algorithms can be observed even in small problem instances like in Figure 4.1 (a) and (b). While BARON is stuck in a poor local optimum with an excessive number of controllers, ECP-LL has managed to achieve a considerably lower objective value with fewer controller placements.

In Figure 4.2, as a result of avoiding controller synchronization cost as γ increases it can be observed that the controller placements become more and more compact. At the limit when $\gamma \to \infty$ we can see that the codebook will collapse into a single controller for both leader-less and leader-based cases.

Figure 4.3 shows the trade-off between different hyper-parameters for ECP-LL algorithm. (a) shows as γ increases the optimal objective value also increases and stays relatively constant for very large values of γ. This is due to the fact that for large γ, controller placement becomes more packed and at its extreme we would have only one controller to cancel out synchronization cost. Figure 4.3 (b) also shows the same pattern that as γ increases ECP-LL places fewer controllers in edge network. Figure 4.3 (c) shows the optimal value for hyper-parameter K_{max} in ECP-LL algorithm. We validate that the optimal value of K_{max} is the number of inherent clusters in the dataset. Figure 4.3 (d) Shows the the values of non-projected and projected [1] solutions versus the number of iterations. The projected solution is obtained by setting association probabilities to either zero or one and then projecting the solution centroids onto the data set.

[1] At the last step of algorithm we mapped centroids onto the closest edge node available. We call such a solution projected, otherwise we call the solution non-projected.

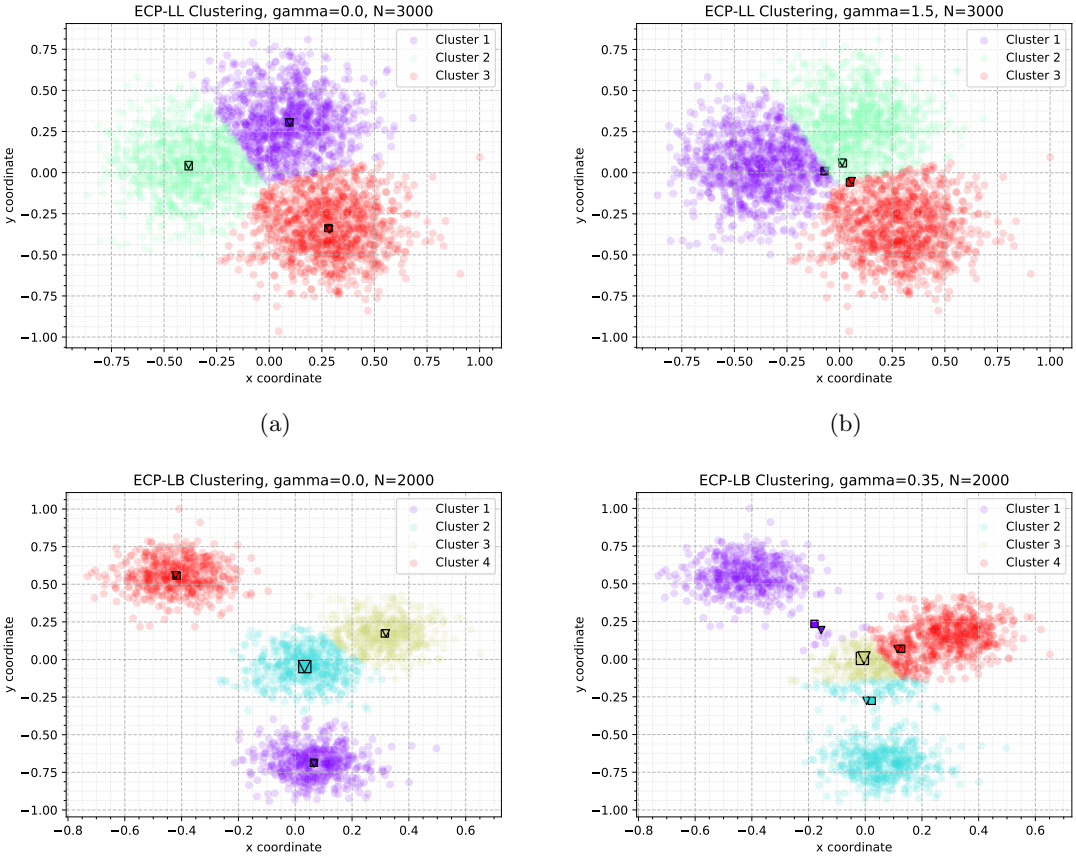


Figure 4.2: Controller placement sensitivity to parameter γ

We observed that the two converge to the same value across most scenarios. Since the non-projected optimal objective value serves as a lower bound for that of the projected, we can assume ECP-LL has reached, in worst case, a near-optimal solution.

Table 4.1 compares performance of ECP-LL against MINLP. While ECP-LL by far outperforms MINLP in terms of total run time, the difference in accuracy is emphasized as problem size increases. ECP-LL and ECP-LB provide consistent performance both in terms of accuracy and speed across different data sizes and varying data clusters as it is by design resilient to local minima that riddle the cost function surface. Figure 4.4 illustrates how run time grows linearly as a function of data size and number of clusters.

4.2 Dynamic Setting

We performed simulations using Python 3.9.0 on a Razer Blade 15 laptop with Intel Core i7-10750H @ 2.60 GHz CPU, 16.0 GB RAM, Nvidia RTX 2060 GPU (1920 cores and 6GB memory) and Windows 10 Operating System. For the initial problem, data is generated as Gaussian distributions with randomized mean and standard deviations. Each starting cluster (distribution) is assigned to a destination cluster of equal size and each point in the initial cluster is assigned to a random point in the respective destination cluster.

The synthetic datasets herein are generated as Gaussian distributions with randomized mean and standard

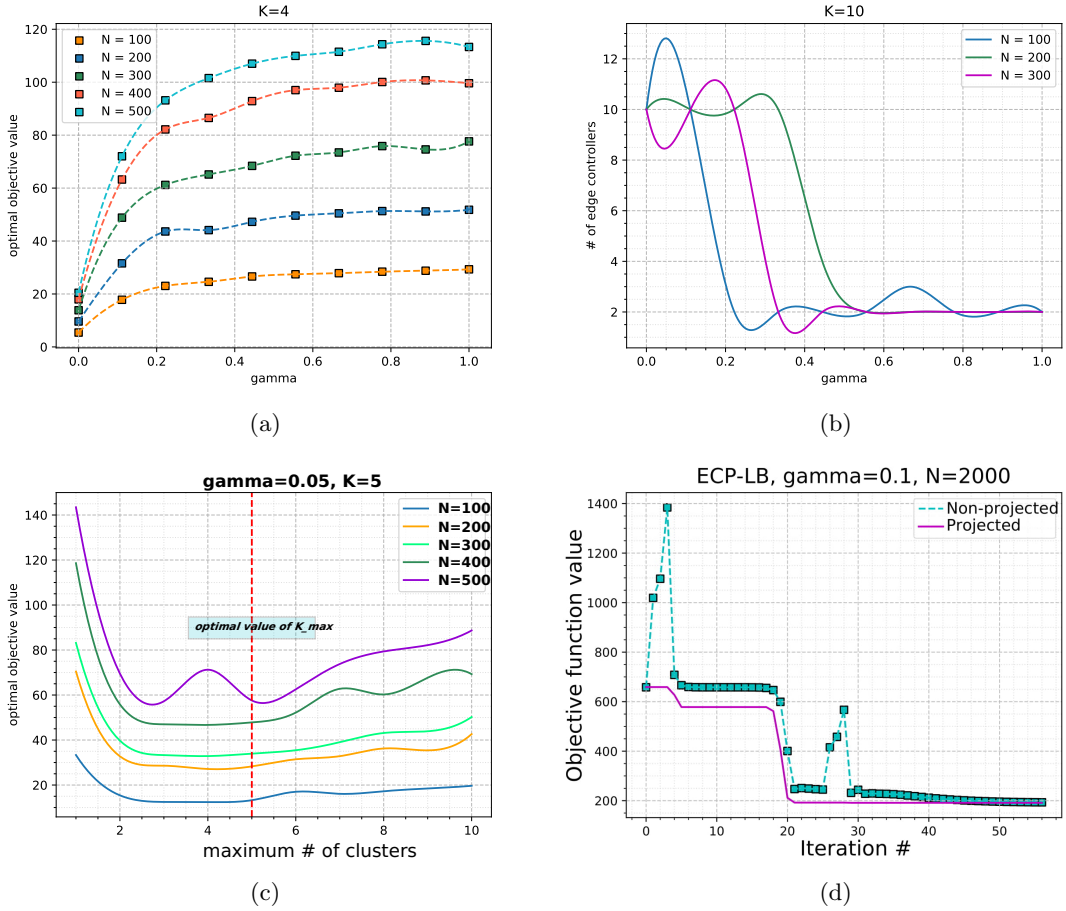


Figure 4.3: (a) γ vs. optimal obj. value (b) γ vs. optimal # of controllers (c) K_{max} vs. optimal obj. val. (d) Iteration # vs. projected and non-projected solutions obj. function values

deviations. Each starting cluster (distribution) is assigned to a destination cluster of equal size and each point in the initial cluster is assigned to a random point in the respective destination cluster. The network is modeled as a first-order linear dynamical system similar to (3.26) with node positions given by $x(t) : \mathbb{R}^+ \to \mathbb{R}^2$, $x(t) = (x_{start} - x_{end})\exp(-kt) + x_{end}$. At $t = 0$ we start at $x_{start} \in \mathbb{R}^2$ and as $t \to \infty$ the system converges to $x_{end} \in \mathbb{R}^2$. The value k for each point is randomized and is generated using the Rayleigh distribution with parameter $\sigma = 0.5$, and the probability distribution function $f(x;\sigma) = \frac{x}{\sigma^2}\exp(\frac{-x^2}{2\sigma^2})$. Rayleigh distributions are typically used for simulation of particle trajectories, which have coordinate-wise normally-distributed velocities [78].

Due to arithmetic underflow that can occur in the floating point division in the update rule (3.21) we recommend the following normalization approach for any point $x \in \Omega$:

$$x_{new} = \frac{x - \mu}{\sigma_{max} - \sigma_{min}} \tag{4.1}$$

where $\mu = \frac{\int_\Omega x dx}{\int_\Omega dx}$ is the mass center of the domain space Ω. $\sigma_{min} = \min_{x\in\Omega} \|x\|_\infty$ and $\sigma_{max} = \max_{x\in\Omega} \|x\|_\infty$ are the minimal and maximal values the coordinates of points in Ω can take, with $\|.\|_\infty$ being the infinity or max norm. After this normalization the network is restrained within the box $[-1, 1] \times [-1, 1]$. Table 4.3

Table 4.1: Duration and total communication delay as a function of size of dataset N and number of clusters K with $\gamma = 0.1$. Tuples show completion time (sec), objective value and number of placed controllers triplets. ECP-LL vs BARON

	N=20	N=40	N=60
K=2	(0.31,7.10,2), (622.24,15.37,4)	(0.52,13.24,2), (606.73,37.08,5)	(0.78,17.50,2), (627.56,585.03,55)
K=4	(0.48,8.65,7), (614.54,12.93,5)	(0.77,12.20,7), (610.03,28.41,8)	(1.08,16.82,7), (624.12,233.88,48)
K=6	(0.67,7.75,4), (605.20,13.70,6)	(1.20,15.19,4), (607.32,38.64,9)	(1.61,23.26,4), (638.92,95.92,13)
K=8	(0.87,6.53,4), (1630.94,9.39,6)	(1.34,18.66,4), (606.02,42.28,9)	(1.96,25.82,4), (618.61,390.73,54)
K=10	(1.05,6.81,2), (602.03,10.32,5)	(1.76,12.60,2), (617.59,27.35,10)	(2.49,19.46,4), (621.64,81.89,15)

Table 4.2: Duration and total communication delay as a function of size of dataset N and number of clusters K and $\gamma = 0.1$. Tuples show completion time (sec), objective value and number of placed controllers triplets. ECP-LB vs BARON

	N=20	N=40	N=60
K=2	(0.93,1.99,5), (2.11,6.21,4)	(1.63,3.68,5), (373.22,11.88,4)	(2.36,6.22,2), (906.84,15.24,3)
K=4	(1.49,2.89,6), (14.87,4.34,5)	(2.69,4.58,7), (267.35,6.09,6)	(4.58,6.79,7), (604.13,9.16,6)
K=6	(2.06,3.52,8), (63.59,4.88,3)	(3.71,8.14,5), (305.31,8.85,5)	(5.23,9.10,8), (612.36,12.78,5)
K=8	(2.81,2.71,11), (16.87,3.78,4)	(4.61,8.17,11), (458.29,10.56,5)	(7.06,11.73,11), (604.58,15.07,5)
K=10	(3.40,3.05,8), (68.99,4.90,4)	(7.10,5.04,9), (297.49,7.25,4)	(10.16,7.34,12), (607.64,12.27,5)

shows performance of RCP in comparison with ECP-LL algorithm. It can be observed that RCP can be consistently up to 25 times faster than the frame-by-frame method using ECP-LL, although the inference time for both algorithms grow linearly with the size of network. Figure 4.6 shows various properties of the RCP algorithm. Plot (a) shows the CDF [2] of inference time [3] for the RCP algorithm given various network sizes. Plots (b) and (c) show the quartiles of inference time for both RCP and ECP-LL algorithms using the box plot. Plot (d) shows the distance of RCP placements to the optimal controller locations. As can be seen this distance converges to zero over time. Plot (e) shows the evolution of total network delay for both ECP-LL and RCP algorithms. Plot (f) compares the inference time of RCP versus ECP-LL algorithms across various network sizes.

Table 4.3: Average inference time (Milliseconds).

Network size	ECP-LL		RCP	
	average	STD	average	STD
50	22.92	8.38	0.83	0.30
100	35.65	20.00	1.35	0.72
500	224.79	67.29	9.03	3.22
1000	370.30	167.19	14.27	6.56

In order to validate RCP+ we recruit multiple popular and publicly available mobility datasets. Table 4.4 shows the properties of each dataset. These datasets contain records of time and location pairs of moving people and cars on the streets of various locations on earth. The records are chronically ordered per data collection device, thus data can be reconstructed into trajectories. The end to end time span of these datasets vary a lot from weeks to years. For implementation purposes on datasets that have a wide temporal span, we

[2] Cumulative distribution function.

[3] Here inference time is the time it takes to compute optimal placements for a system snapshot.

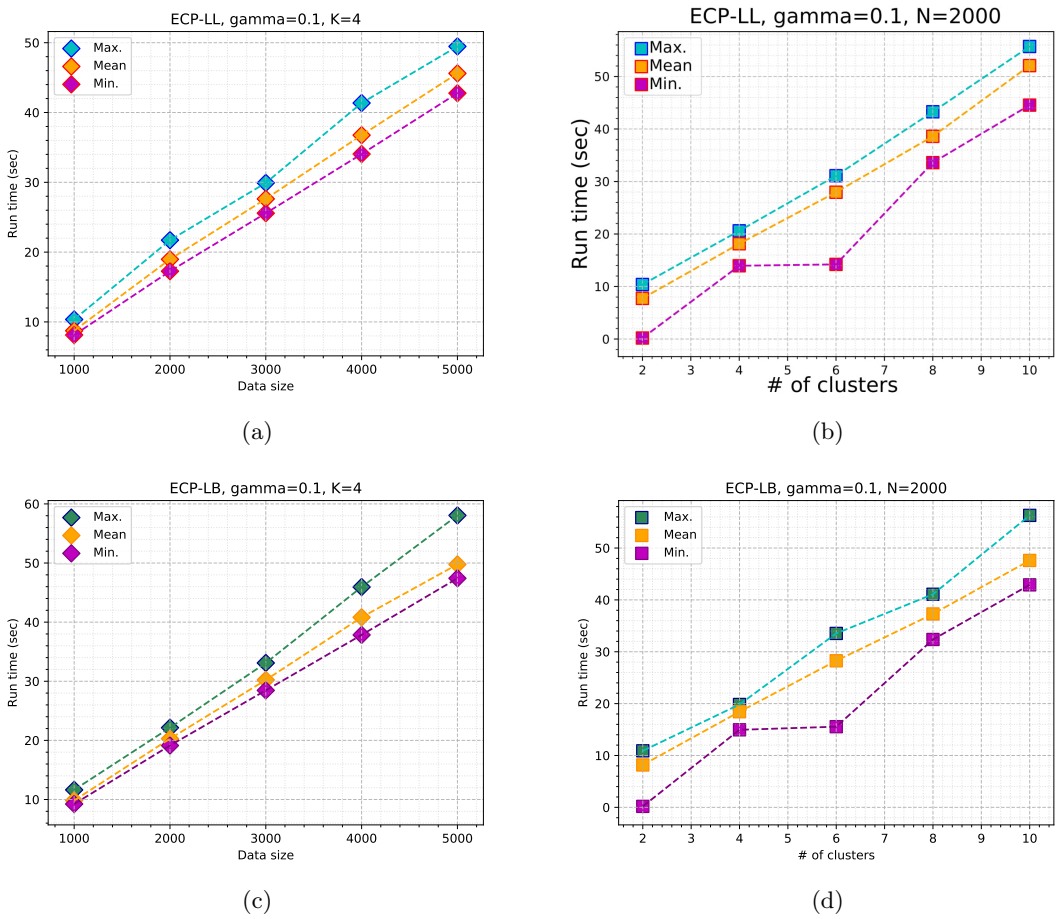


Figure 4.4: ECP algorithms run time vs. # of clusters and data size.

only experiment with an excerpt of the data, focusing the simulation to a single day of the year, preferably a busier day. The objective of this benchmarking study is to evaluate the performance of RCP+ in dynamic scenarios where the network topology and traffic patterns continuously change over time due to the movement of pedestrians and vehicles. By using real-world mobility datasets, we aim to simulate realistic and challenging network conditions that reflect the complexities of urban environments.

Figure 4.7 shows the heat map generated for these datasets. Since the plots include the whole dataset, the warmer colors shows congestion both temporally and spatially, most notably in the centers of business in the cities.

Figure 4.8 shows the placement controllers using RCP+ versus the optimal placements. *Optimal* as mentioned earlier corresponds to a locally optimal point specifically generated using the ECP-LL algorithm–to which RCP+ is the dynamic counterpart. This can be viewed as the solution that would be found if RCP+

Dataset	No. of records	trajectories per day		distance			time intervals		Time span	Location	Owner
		avg.	std	total	avg.	std	avg.	std			
Cabspotting [74]	11.2 millions	477	77	2 million km	237 m	225 m	58 secs	26 secs	24 days	San Francisco, USA	Dartmouth
T-drive [75]	17.7 millions	9,838	185	1.5 trillion km	146 m	252 m	42 secs	50 secs	6 days	Beijing, China	Microsoft
UCI [76]	18.1 thousands	2.2	1.7	804 km	45m	62 m	7.86 secs	8.1 secs	493 days	Aracaju, Brazil	UCI

Table 4.4: Public mobility datasets

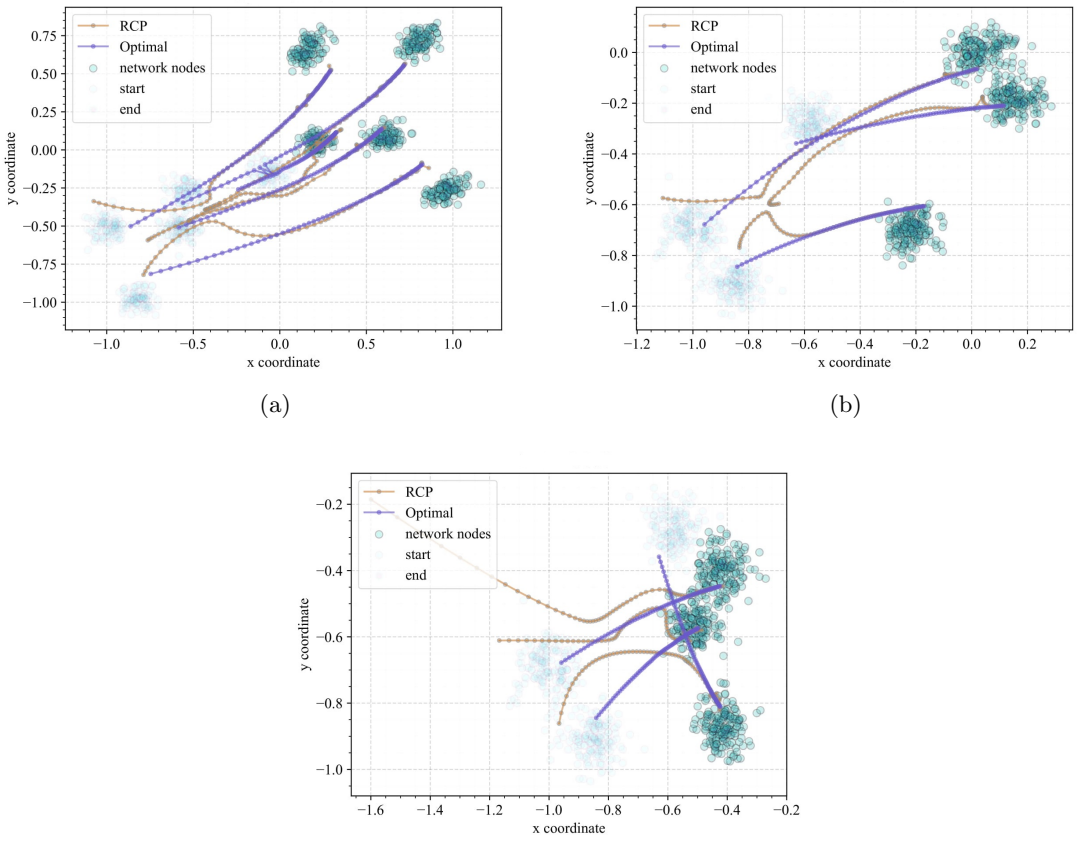


Figure 4.5: Performance of RCP for three network topologies and mobility models.

had infinite time at each time step to compute the placements.

Figure 4.9 shows sensitivity of the RCP+ trajectories to the hyperparameter K_0 that according to control law 3.53, it magnifies the velocity of controllers. It can be seen that for smaller values of K_0 stabilization at optimal placements happens at a slower rate. On the other hand for values of K_0 that is too high stabilization occurs at a faster rate however it is possible that it will distort the trajectories generating non-smooth paths, a behavior similar to gradient descent algorithm with a high learning rate. This can be alleviated as we mentioned earlier by annealing this hyperparameter; this fact can be observed from comparison of plots (c) and (d).

Algorithm	UCI		Cabspotting		T-drive	
	μ	σ	μ	σ	μ	σ
RCP+	0.31	0.14	2.13	1.01	14.31	6.21
RCP [14]	1.02	0.31	9.32	2.09	72.21	17.41
ECP-LL [57]	23.41	8.32	221.91	75.21	1531.92	426.6
Randomized Greedy [50]	41.25	1.93	500.05	83.19	2341.44	260.14

Table 4.5: Inference time (Milliseconds) of DCP algorithms for mobility datasets.

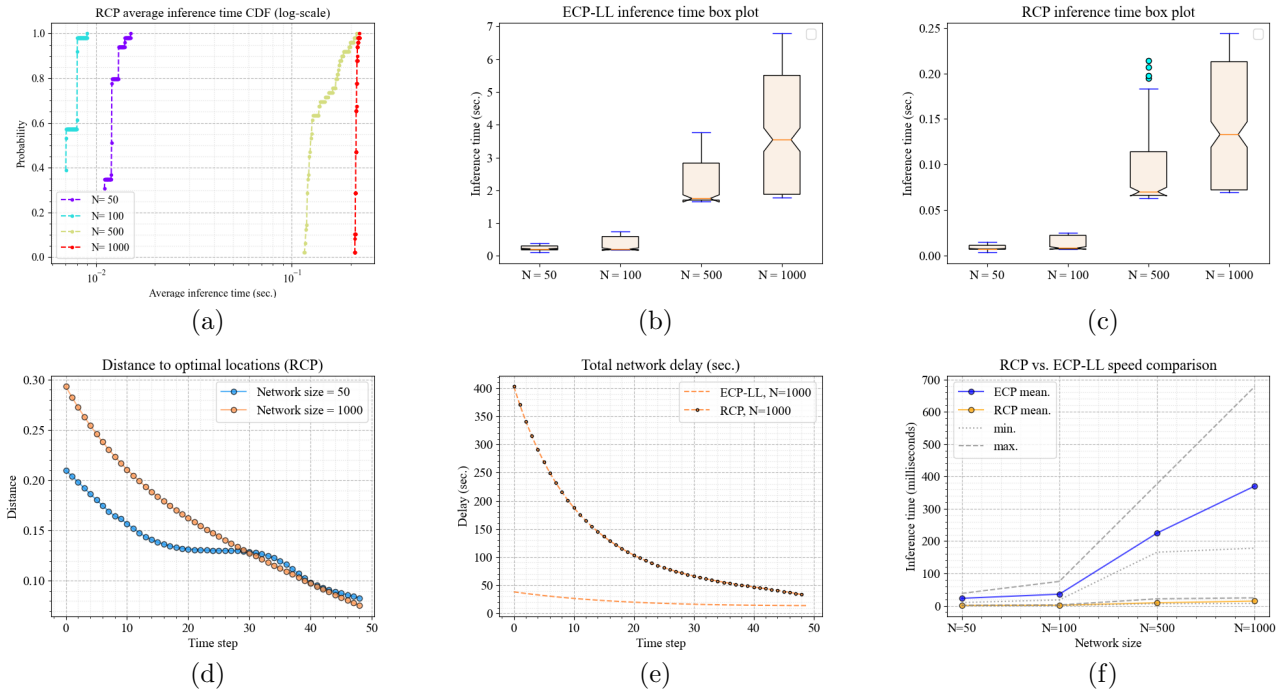
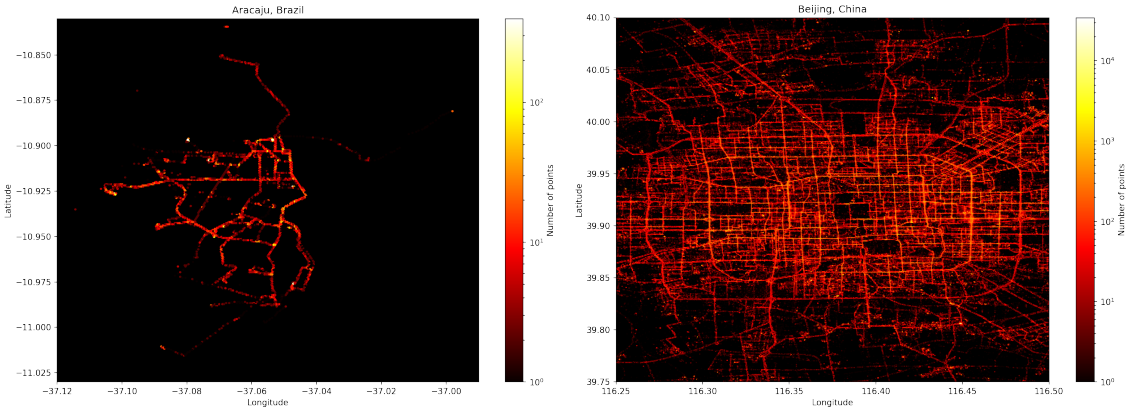


Figure 4.6: Performance of RCP in terms of delay and inference time.

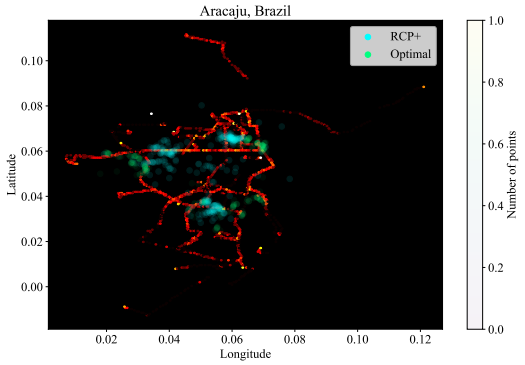


(a) UCI dataset, Aracaju, Brazil.

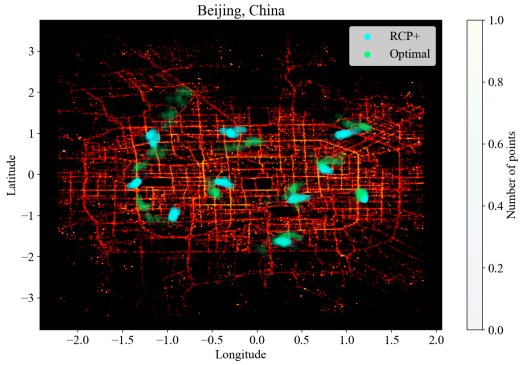
(b) T-drive dataset, Beijing.

(c) Cab-spotting dataset, San Francisco, USA.

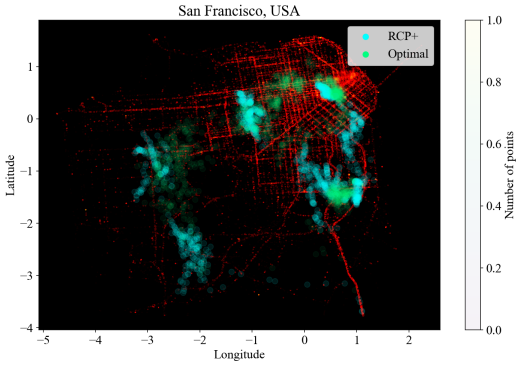
Figure 4.7: Heat map of records in public mobility datasets.



(a) Aracaju, Brazil, $N = 71, M = 3$

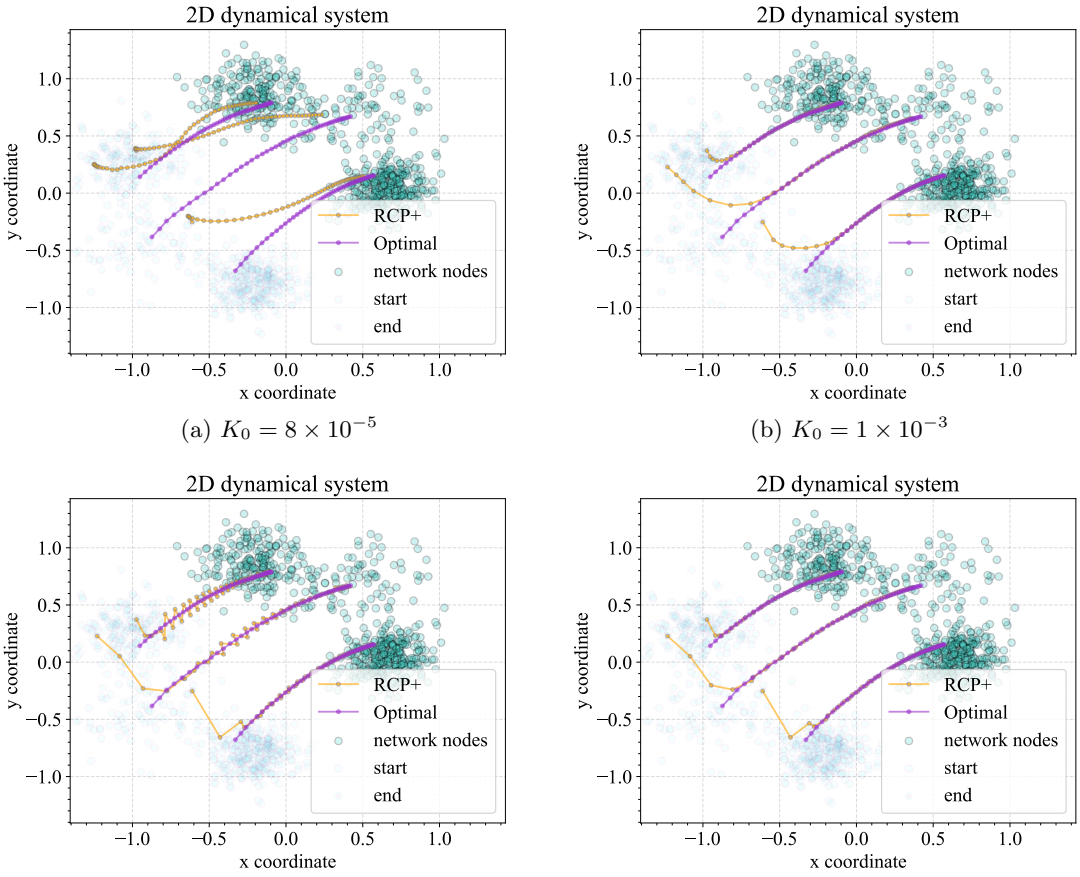


(b) Beijing, China, $N = 4398, M = 10$



(c) San Francisco, USA, $N = 474, M = 5$

Figure 4.8: UAV swarm placements for mobility datasets, using RCP+.



(a) $K_0 = 8 \times 10^{-5}$

(b) $K_0 = 1 \times 10^{-3}$

(c) $K_0 = 2 \times 10^{-3}$

(d) $K_0 = 2 \times 10^{-3}$, anneal rate $= 0.9$

Figure 4.9: Sensitivity analysis of hyperparameter K_0.

Chapter 5

Conclusion

In this thesis we introduced a family of maximum entropy based clustering algorithms for placement of controller devices in SDN networks in static and dynamic network scenarios. Our static algorithms ECP-LL and ECP-LB each address a different controller placement topology, and their design is inspired by a Mixed Integer Nonlinear Program. We show that our algorithms outperform state of art MINLP solver, BARON in both speed and accuracy. Total computational complexity for these algorithms is $O(\tau N K_{max})$ which is linear in data size, and maximum number of clusters. The main contributions of our static solutions in comparison with other ECP algorithms is quality of placement in terms of fewer controller devices placed and overall network delay. Typical ECP algorithms depend on multiple initialization and are collectively susceptible to getting trapped in poor local optima that abound the objective function of controller placement problem. ECP-LL and ECP-LB require no initialization and consistently yield placements that in practice has been shown to be very close to or exactly match the globally optimum placement of controllers.

Wired networks were the first to be affected by prevalence of softwarization paradigm, and as such SDN was implemented in data centers and ISP networks. More recently SDMN's were introduced as an analog in the context of wireless networks. Given high dynamicity of SDMN's, researchers have hypothesized about using mobile devices to host the so-called controller software, that leads to enabling instantaneous reaction to abrupt shifts in the network. In this light we introduced the RCP algorithm, a temporal clustering algorithm for real-time controller placement in mobile SDN systems. RCP can be viewed as a temporal extension of ECP-LL that was mainly designed to tackle the real-time aspect of CP without compromising the quality of placements. To the best of our knowledge, RCP is the first algorithm in DCP literature that exploits the temporal relationships of the network dynamics in order to efficiently adapt placement solutions in real-time. RCP leverages the principle of maximum entropy to avoid poor local optima that abound on the surface of our balanced cost function, and thus consistently provides high quality solutions. Unlike conventional methods that shrink the decision space into a discrete set, our algorithm allows use of the *open search* method for placement, which makes it unlikely to yield sub-optimal solutions. RCP has linear $\mathcal{O}(N)$ iteration computational complexity with respect to the network size and can be substantially faster than the conventional frame-by-frame approach. A major drawback of RCP is that it assumes network node dynamics is known, and this was the original motivation for introduction of RCP+ and creating a more practical solution for real-time controller placement problem.

RCP+ can be considered as a major step towards recruiting UAV swarms in order to implement the mobile controller concept in SDN's that offer continuous control over the performance of these networks.

RCP+, a successor to the RCP algorithm, used for real-time placement of controllers in software defined networks can be considered an upgrade that significantly improves the practicality of its predecessor on multiple fronts. In this light our work recruits various subroutines such as sparse network sampling, node prioritization, trajectory prediction using neural networks, and computation parallelization over GPU's that enables application to real world datasets and maintains the inference time of RCP+ within modern definitions of *real-time* computation. In this work we prove various properties of RCP+ including exponential stabilization to optimal placements, phase transition conditions, and convergence proof of our method. We benchmarked RCP+ against comparable state of art algorithms and show that it can by far outperform the conventional *static* approach both in terms of speed and overal network delay. We ackonwledge that performance of RCP+, in terms of cumulative delay can rely heavily on choice of hyperparameters. A future direction of this work can be to automate the process of hyperparameter selection, based on network size and its dynamics. Moreover, we propose extension of RCP+ to take into account node link quality, constrained size of clusters, constrained inter-controller distances, and application to resilience to UAV swarm jamming, in hostile environments.

References

[1] N. McKeown *et al.*, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008, ISSN: 0146-4833. DOI: 10.1145/1355734.1355746. [Online]. Available: https://doi.org/10.1145/1355734.1355746.

[2] A. Alshamrani, S. Guha, S. Pisharody, A. Chowdhary, and D. Huang, "Fault Tolerant Controller Placement in Distributed SDN Environments," in *2018 IEEE International Conference on Communications (ICC)*, IEEE, May 2018, pp. 1–7, ISBN: 978-1-5386-3180-5. DOI: 10.1109/ICC.2018.8422593. [Online]. Available: https://ieeexplore.ieee.org/document/8422593/.

[3] D. Dungay, *Software Defined Networking (SDN) Explained — Comms Business*, 2016. [Online]. Available: https://www.commsbusiness.co.uk/features/software-defined-networking-sdn-explained/.

[4] H. Kuang, Y. Qiu, R. Li, and X. Liu, "A Hierarchical K-Means Algorithm for Controller Placement in SDN-Based WAN Architecture," in *2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, IEEE, Feb. 2018, pp. 263–267, ISBN: 978-1-5386-5114-8. DOI: 10.1109/ICMTMA.2018.00070. [Online]. Available: https://ieeexplore.ieee.org/document/8337381/.

[5] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks - HotSDN '12*, New York, New York, USA: ACM Press, 2012, p. 7, ISBN: 9781450314770. DOI: 10.1145/2342441.2342444. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2342441.2342444.

[6] A. K. Singh and S. Srivastava, "A survey and classification of controller placement problem in SDN," *International Journal of Network Management*, vol. 28, no. 3, e2018, May 2018, ISSN: 10557148. DOI: 10.1002/nem.2018. [Online]. Available: http://doi.wiley.com/10.1002/nem.2018.

[7] Vigas G, *Software Defined Networking Market Size, Share — Forecast - 2027*, 2020. [Online]. Available: https://www.alliedmarketresearch.com/software-defined-networking-market.

[8] M. Alharthi, A. E. M. Taha, and H. S. Hassanein, "Dynamic controller placement in software defined drone networks," in *2019 IEEE Global Communications Conference, GLOBECOM 2019 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Dec. 2019, ISBN: 9781728109626. DOI: 10.1109/GLOBECOM38437.2019.9013799.

[9] M. A. Sayeed, R. Kumar, and V. Sharma, "Efficient data management and control over WSNs using SDN-enabled aerial networks," *International Journal of Communication Systems*, vol. 33, no. 1, e4170, Jan. 2020, ISSN: 10745351. DOI: 10.1002/dac.4170. [Online]. Available: http://doi.wiley.com/10.1002/dac.4170.

[10] M. Campion, P. Ranganathan, and S. Faruque, "UAV swarm communication and control architectures: a review," *Journal of Unmanned Vehicle Systems*, vol. 7, no. 2, pp. 93–106, 2019. DOI: 10.1139/juvs-2018-0009. [Online]. Available: https://doi.org/10.1139/juvs-2018-0009.

[11] C. Singhal and K. Rahul, "LB-UAVnet: Load Balancing Algorithm for UAV Based Network using SDN," in *International Symposium on Wireless Personal Multimedia Communications, WPMC*, vol. 2019-November, IEEE Computer Society, Nov. 2019, ISBN: 9781728154190. DOI: 10.1109/WPMC48795.2019.9096139.

[12] S. Ur Rahman, G. H. Kim, Y. Z. Cho, and A. Khan, "Deployment of an SDN-based UAV network: Controller placement and tradeoff between control overhead and delay," in *International Conference on Information and Communication Technology Convergence: ICT Convergence Technologies Leading the Fourth Industrial Revolution, ICTC 2017*, vol. 2017-December, Institute of Electrical and Electronics Engineers Inc., Dec. 2017, pp. 1290–1292, ISBN: 9781509040315. DOI: 10.1109/ICTC.2017.8190924.

[13] J. D. Day and H. Zimmermann, "The OSI reference model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983. DOI: 10.1109/PROC.1983.12775.

[14] R. Soleymanifar and C. Beck, "RCP: A Temporal Clustering Algorithm for Real-time Controller Placement in Mobile SDN Systems," Dec. 2021. [Online]. Available: https://arxiv.org/abs/2112.03037v1.

[15] M. Osama, A. A. Ateya, S. Ahmed Elsaid, and A. Muthanna, "Ultra-Reliable Low-Latency Communications: Unmanned Aerial Vehicles Assisted Systems," *Information*, vol. 13, no. 9, 2022, ISSN: 2078-2489. DOI: 10.3390/info13090430. [Online]. Available: https://www.mdpi.com/2078-2489/13/9/430.

[16] A. Abdelaziz *et al.*, "Distributed controller clustering in software defined networks," *PLOS ONE*, vol. 12, no. 4, C.-H. Huang, Ed., e0174715, Apr. 2017, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0174715. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0174715.

[17] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data Clustering: A Review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, Sep. 1999, ISSN: 0360-0300. DOI: 10.1145/331499.331504. [Online]. Available: https://doi.org/10.1145/331499.331504.

[18] T. K. Dey, A. Rossi, and A. Sidiropoulos, "Temporal clustering," *arXiv preprint arXiv:1704.05964*, 2017.

[19] P. Sharma, S. M. Salapaka, and C. L. Beck, "Entropy-based framework for dynamic coverage and clustering problems," *IEEE Transactions on Automatic Control*, vol. 57, no. 1, pp. 135–150, Jan. 2012, ISSN: 00189286. DOI: 10.1109/TAC.2011.2166713.

[20] H. Yang and M. Tate, "A descriptive literature review and classification of cloud computing research," *Communications of the Association for Information Systems*, vol. 31, no. 1, p. 2, 2012.

[21] F. Li and X. Xu, "A Discrete Cuckoo Search Algorithm for the Controller Placement Problem in Software Defined Networks," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, IEEE, Nov. 2018, pp. 292–296, ISBN: 978-1-5386-7266-2. DOI: 10.1109/IEMCON.2018.8614785. [Online]. Available: https://ieeexplore.ieee.org/document/8614785/.

[22] J. Lu, Z. Zhang, T. Hu, P. Yi, and J. Lan, "A Survey of Controller Placement Problem in Software-Defined Networking," *IEEE Access*, vol. 7, pp. 24 290–24 307, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2893283. [Online]. Available: https://ieeexplore.ieee.org/document/8618449/.

[23] P. Tao, C. Ying, Z. Sun, S. Tan, P. Wang, and Z. Sun, "The Controller Placement of Software-Defined Networks Based on Minimum Delay and Load Balancing," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, IEEE, Aug. 2018, pp. 310–313, ISBN: 978-1-5386-7518-2. DOI: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00059. [Online]. Available: https://ieeexplore.ieee.org/document/8511902/.

[24] A. Dvir, Y. Haddad, and A. Zilberman, "Wireless controller placement problem," in *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, Jan. 2018, pp. 1–4, ISBN: 978-1-5386-4790-5. DOI: 10.1109/CCNC.2018.8319228. [Online]. Available: http://ieeexplore.ieee.org/document/8319228/.

[25] A. Basta, A. Blenk, M. Hoffmann, H. J. Morper, K. Hoffmann, and W. Kellerer, "SDN and NFV dynamic operation of LTE EPC gateways for time-varying traffic patterns," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 141, Springer Verlag, 2015, pp. 63–76, ISBN: 9783319162911. DOI: 10.1007/978-3-319-16292-85.

[26] N. Mouawad, R. Naja, and S. Tohme, "Optimal and Dynamic SDN Controller Placement," in *2018 International Conference on Computer and Applications, ICCA 2018*, Institute of Electrical and Electronics Engineers Inc., Sep. 2018, pp. 413–418, ISBN: 9781538643716. DOI: 10.1109/COMAPP.2018.8460361.

[27] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli, "Large-Scale Dynamic Controller Placement," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 63–76, Mar. 2017, ISSN: 1932-4537. DOI: 10.1109/TNSM.2017.2651107. [Online]. Available: http://ieeexplore.ieee.org/document/7812749/.

[28] B. P. R. Killi, E. A. Reddy, and S. V. Rao, "Cooperative game theory based network partitioning for controller placement in SDN," in *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, IEEE, Jan. 2018, pp. 105–112, ISBN: 978-1-5386-1182-1. DOI: 10.1109/COMSNETS.2018.8328186. [Online]. Available: http://ieeexplore.ieee.org/document/8328186/.

[29] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, and T. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings," *Computer Networks*, vol. 112, pp. 24–35, Jan. 2017, ISSN: 1389-1286. DOI: 10.1016/J.COMNET.2016.10.014. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S1389128616303620.

[30] S. Wu, X. Chen, L. Yang, C. Fan, and Y. Zhao, "Dynamic and static controller placement in Software-Defined Satellite Networking," *Acta Astronautica*, vol. 152, pp. 49–58, Nov. 2018, ISSN: 0094-5765. DOI: 10.1016/J.ACTAASTRO.2018.07.017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576518308269.

[31] A. Papa, T. De Cola, P. Vizarreta, M. He, C. Mas Machuca, and W. Kellerer, "Dynamic SDN Controller Placement in a LEO Constellation Satellite Network," in *2018 IEEE Global Communications Conference (GLOBECOM)*, IEEE, Dec. 2018, pp. 206–212, ISBN: 978-1-5386-4727-1. DOI: 10.1109/GLOCOM.2018.8647843. [Online]. Available: https://ieeexplore.ieee.org/document/8647843/.

[32] T. Das and M. Gurusamy, "INCEPT: INcremental ControllEr PlacemenT in Software Defined Networks," in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, IEEE, Jul. 2018, pp. 1–6, ISBN: 978-1-5386-5156-8. DOI: 10.1109/ICCCN.2018.8487454. [Online]. Available: https://ieeexplore.ieee.org/document/8487454/.

[33] J. Liu, Y. Shi, L. Zhao, Y. Cao, W. Sun, and N. Kato, "Joint Placement of Controllers and Gateways in SDN-Enabled 5G-Satellite Integrated Network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 2, pp. 221–232, Feb. 2018, ISSN: 0733-8716. DOI: 10.1109/JSAC.2018.2804019. [Online]. Available: http://ieeexplore.ieee.org/document/8286925/.

[34] Zhiyang Su and M. Hamdi, "MDCP: Measurement-Aware Distributed Controller Placement for Software Defined Networks," in *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, IEEE, Dec. 2015, pp. 380–387, ISBN: 978-0-7695-5785-4. DOI: 10.1109/ICPADS.2015.55. [Online]. Available: http://ieeexplore.ieee.org/document/7384318/.

[35] A. Jalili, M. Keshtgari, R. Akbari, and R. Javidan, "Multi criteria analysis of Controller Placement Problem in Software Defined Networks," *Computer Communications*, vol. 133, pp. 115–128, Jan. 2019, ISSN: 0140-3664. DOI: 10.1016/J.COMCOM.2018.08.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366418300045.

[36] B. Zhang, X. Wang, and M. Huang, "Multi-objective optimization controller placement problem in internet-oriented software defined network," *Computer Communications*, vol. 123, pp. 24–35, Jun. 2018, ISSN: 0140-3664. DOI: 10.1016/J.COMCOM.2018.04.008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366416307241.

[37] K. Rose, "Deterministic Annealing for Clustering, Compression, Classification, Regression, and Related Optimization Problems," California Institute of Technology, Tech. Rep., 1998. [Online]. Available: https://pdfs.semanticscholar.org/4ce8/bc485df9ac987f18d99c7af1d95f9cbea6b2.pdf.

[38] S. Toufga, S. Abdellatif, H. T. Assouane, P. Owezarski, and T. Villemur, "Towards Dynamic Controller Placement in Software Defined Vehicular Networks," *Sensors*, vol. 20, no. 6, p. 1701, Mar. 2020, ISSN: 1424-8220. DOI: 10.3390/s20061701. [Online]. Available: https://www.mdpi.com/1424-8220/20/6/1701.

[39] S. Champagne, T. Makanju, C. Yao, N. Zincir-Heywood, and M. Heywood, "A genetic algorithm for dynamic controller placement in software defined networking," in *GECCO 2018 Companion - Proceedings of the 2018 Genetic and Evolutionary Computation Conference Companion*, New York, NY, USA: Association for Computing Machinery, Inc, Jul. 2018, pp. 1632–1639, ISBN: 9781450357647. DOI: 10.1145/3205651.3208244. [Online]. Available: https://dl.acm.org/doi/10.1145/3205651.3208244.

[40] M. He, A. Varasteh, and W. Kellerer, "Toward a Flexible Design of SDN Dynamic Control Plane: An Online Optimization Approach," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1694–1708, Dec. 2019, ISSN: 19324537. DOI: 10.1109/TNSM.2019.2935160.

[41] D. K. Luong, Y. F. Hu, J. P. Li, F. Benamrane, M. Ali, and K. Abdo, "Traffic-aware Dynamic Controller Placement using AI techniques in SDN-based aeronautical networks," in *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, vol. 2019-September, Institute of Electrical and Electronics Engineers Inc., Sep. 2019, ISBN: 9781728106496. DOI: 10.1109/DASC43569.2019.9081810.

[42] M. He, A. Basta, A. Blenk, and W. Kellerer, "How Flexible is Dynamic SDN Control Plane?" Technical University of Munich, Tech. Rep., 2018.

[43] M. He, A. Basta, A. Blenk, and W. Kellerer, "Modeling flow setup time for controller placement in SDN: Evaluation for dynamic flows," in *IEEE International Conference on Communications*, Institute of Electrical and Electronics Engineers Inc., Jul. 2017, ISBN: 9781467389990. DOI: 10.1109/ICC.2017.7996654.

[44] T. Wang, F. Liu, and H. Xu, "An Efficient Online Algorithm for Dynamic SDN Controller Assignment in Data Center Networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2788–2801, Oct. 2017, ISSN: 10636692. DOI: 10.1109/TNET.2017.2711641.

[45] Y. Liu, H. Gu, X. Yu, and J. Zhou, "Dynamic SDN Controller Placement in Elastic Optical Datacenter Networks," *Asia Communications and Photonics Conference, ACP*, vol. 2018-October, Dec. 2018, ISSN: 2162108X. DOI: 10.1109/ACP.2018.8596219.

[46] S. Hegde, R. Ajayghosh, S. G. Koolagudi, and S. Bhattacharya, "Dynamic controller placement in edge-core software defined networks," *IEEE Region 10 Annual International Conference, Proceedings/-TENCON*, vol. 2017-December, pp. 3153–3158, Dec. 2017, ISSN: 21593450. DOI: 10.1109/TENCON.2017.8228403.

[47] H. Sufiev, Y. Haddad, L. Barenboim, and J. Soler, "Dynamic SDN Controller Load Balancing," *Future Internet*, vol. 11, no. 3, 2019, ISSN: 1999-5903. DOI: 10.3390/fi11030075. [Online]. Available: https://www.mdpi.com/1999-5903/11/3/75.

[48] Z. Li, Y. Hu, T. Hu, and P. Wei, "Dynamic SDN Controller Association Mechanism Based on Flow Characteristics," *IEEE Access*, vol. 7, pp. 92 661–92 671, 2019. DOI: 10.1109/ACCESS.2019.2927173.

[49] Y. Wu, S. Zhou, Y. Wei, and S. Leng, "Deep Reinforcement Learning for Controller Placement in Software Defined Network," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2020, pp. 1254–1259. DOI: 10.1109/INFOCOMWKSHPS50562.2020.9162977.

[50] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas, "SDN Controller Placement at the Edge: Optimizing Delay and Overheads," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, IEEE, Apr. 2018, pp. 684–692, ISBN: 978-1-5386-4128-6. DOI: 10.1109/INFOCOM.2018.8485963. [Online]. Available: https://ieeexplore.ieee.org/document/8485963/.

[51] W. Zhang, "Branch-and-Bound Search Algorithms and Their Computational Complexity.," UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST, Tech. Rep., 1996.

[52] S. Kudrle, M. Proulx, P. Carrières, and M. Lopez, "Fingerprinting for solving A/V synchronization issues within broadcast environments," *SMPTE Motion Imaging Journal*, vol. 120, no. 5, pp. 36–46, 2011, ISSN: 15450279. DOI: 10.5594/J18059XY.

[53] M. Horani and M. O. Hasna, "Latency Analysis of UAV based Communication Networks," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, 2018, pp. 385–390. DOI: 10.1109/ICTC.2018.8539626.

[54] Alevizaki Victoria-Mariaand, Anastasopoulos Markosand, Tzanakaki Annaand, and Simeonidou Dimitra, "Joint Fronthaul Optimization and SDN Controller Placement in Dynamic 5G Networks," in *Optical Network Design and Modeling*, Tzanakaki Annaand Varvarigos, Manosand Muñoz Rauland, Nejabati Rezaand, Yoshikane Noboruand, Anastasopoulos Markosand, and Marquez-Barja Johann, Eds., Cham: Springer International Publishing, 2020, pp. 181–192, ISBN: 978-3-030-38085-4.

[55] D. Hock, M. Hartmann, S. Gebert, T. Zinner, and P. Tran-Gia, "POCO-PLC: Enabling dynamic pareto-optimal resilient controller placement in SDN networks," in *Proceedings - IEEE INFOCOM*, Institute of Electrical and Electronics Engineers Inc., 2014, pp. 115–116, ISBN: 9781479930883. DOI: 10.1109/INFCOMW.2014.6849182.

[56] Q. Qin, K. Poularakis, G. Iosifidis, S. Kompella, and L. Tassiulas, "SDN Controller Placement With Delay-Overhead Balancing in Wireless Edge Networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1446–1459, Dec. 2018, ISSN: 1932-4537. DOI: 10.1109/TNSM.2018.2876064. [Online]. Available: https://ieeexplore.ieee.org/document/8491378/.

[57] R. Soleymanifar, A. Srivastava, C. Beck, and S. Salapaka, "A Clustering Approach to Edge Controller Placement in Software-Defined Networks with Cost Balancing," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 2642–2647, Jan. 2020, ISSN: 2405-8963. DOI: 10.1016/J.IFACOL.2020.12.379.

[58] Y. Xu, S. M. Salapaka, and C. L. Beck, "Clustering and coverage control for systems with acceleration-driven dynamics," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1342–1347, 2014, ISSN: 00189286. DOI: 10.1109/TAC.2013.2292726.

[59] R. Sepulchre, M. Janković, and P. V. Kokotović, "Constructive Nonlinear Control," Communications and Control Engineering, 1997. DOI: 10.1007/978-1-4471-0967-9. [Online]. Available: http://link.springer.com/10.1007/978-1-4471-0967-9.

[60] E. D. Sontag, "A 'universal' construction of Artstein's theorem on nonlinear stabilization," *Systems and Control Letters*, vol. 13, no. 2, pp. 117–123, Aug. 1989, ISSN: 01676911. DOI: 10.1016/0167-6911(89)90028-5.

[61] K. Poularakis, Q. Qin, E. Nahum, M. Rio, and L. Tassiulas, "Bringing SDN to the mobile edge," in *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, 2017, pp. 1–6. DOI: 10.1109/UIC-ATC.2017.8397407.

[62] J. Lee *et al.*, "MeSDN: Mobile Extension of SDN," in *Proceedings of the Fifth International Workshop on Mobile Cloud Computing & Services*, ser. MCS '14, New York, NY, USA: Association for Computing Machinery, 2014, pp. 7–14, ISBN: 9781450328241. DOI: 10.1145/2609908.2609948. [Online]. Available: https://doi.org/10.1145/2609908.2609948.

[63] D. Syrivelis, G. Iosifidis, D. Delimpasis, K. Chounos, T. Korakis, and L. Tassiulas, "Bits and coins: Supporting collaborative consumption of mobile internet," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 2146–2154. DOI: 10.1109/INFOCOM.2015.7218600.

[64] A. Ahmadi-Javid, P. Seyedi, and S. S. Syam, "A survey of healthcare facility location," *Computers & Operations Research*, vol. 79, pp. 223–263, 2017.

[65] E. T. Jaynes, "Information Theory and Statistical Mechanics," *Physical Review*, vol. 106, no. 4, p. 620, May 1957, ISSN: 0031899X. DOI: 10.1103/PhysRev.106.620. [Online]. Available: https://journals.aps.org/pr/abstract/10.1103/PhysRev.106.620.

[66] C. E. Shannon, "A Mathematical Theory of Communication*," 1948.

[67] D. P. Bertsekas, "Convexification procedures and decomposition methods for nonconvex optimization problems," *Journal of Optimization Theory and Applications 1979 29:2*, vol. 29, no. 2, pp. 169–197, Oct. 1979, ISSN: 1573-2878. DOI: 10.1007/BF00937167. [Online]. Available: https://link.springer.com/article/10.1007/BF00937167.

[68] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[69] A. P. Dempster, ; N. M. Laird, and ; D. B. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[70] D. Arthur and S. Vassilvitskii, "K-Means++: The Advantages of Careful Seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '07, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035, ISBN: 9780898716245.

[71] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975, ISSN: 0001-0782. DOI: 10.1145/361002.361007. [Online]. Available: https://doi.org/10.1145/361002.361007.

[72] M. Ghashami, E. Liberty, J. M. Phillips, and D. P. Woodruff, "Frequent Directions : Simple and Deterministic Matrix Sketching," *CoRR*, vol. abs/1501.01711, 2015. [Online]. Available: http://arxiv.org/abs/1501.01711.

[73] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[74] D. Kotz, T. Henderson, I. Abyzov, J. Yeo, and R. Santos, *Cabspotting dataset*, Downloaded from https://crawdad.org/dartmouth/campus/20090909, Sep. 2009. DOI: 10.15783/C7F59T. [Online]. Available: http://www.lac.inpe.br/~rafael.santos/Docs/CAP394/Proj_Cabspotting.html#about.

[75] Y. Zheng, *T-Drive trajectory data sample*, Aug. 2011. [Online]. Available: https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/.

[76] M. O. Cruz, H. T. Macedo, R. Barreto, and A. P. GuimarÃ£es, *GPS Trajectories Data Set*, 2016. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/GPS+Trajectories.

[77] S. Yadav and S. Shukla, "Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification," *Proceedings - 6th International Advanced Computing Conference, IACC 2016*, pp. 78–83, Aug. 2016. DOI: 10.1109/IACC.2016.25.

[78] S. E. Tuller and A. C. Brett, "The goodness of fit of the weibull and rayleigh distributions to the distributions of observed wind speeds in a topographically diverse area," *Journal of Climatology*, vol. 5, no. 1, pp. 79–94, 1985. DOI: https://doi.org/10.1002/joc.3370050107. [Online]. Available: https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/joc.3370050107.

Appendix A

Codes

In this part we present the Python code used to implement the algorithms presented in this thesis. Python version 3.9 is used throughout the code base below.

Listing A.1: RCP Algorithm

```
1
2 class RCP:
3     """
4     Real-time Controller Placement (RCP) algorithm for mobile Software Defined
          Networks.
5     """
6
7     def __init__(self, t_series, y_0=None, m=None, seed=5,
8                  k_0=5e-5, k=1, alpha=0.9, gamma=0.1,
9                  rand_init=False, T=1, save_name='rcp_hist', ground_truth=True):
10         """
11         Initialize the RCP algorithm.
12
13         Parameters:
14             t_series (list): List of time-series data points, where each element
                  is an array of shape (n, 2).
15             y_0 (ndarray, optional): Initial set of m 2D points. If None, m must
                  be provided.
16             m (int, optional): Number of points in the initial set y_0. Required
                  if y_0 is None.
17             seed (int, optional): Seed for random number generation.
18             k_0 (float, optional): Initial value of k parameter.
19             k (float, optional): Scaling factor for the k parameter at each
                  iteration.
20             alpha (float, optional): Multiplicative factor to decrease T at each
                  iteration.
21             gamma (float, optional): Scaling factor for the synchronization cost
                  importance.
22             rand_init (bool, optional): Whether to add random noise to the
```

```
                initial set y_0.
            T (float, optional): Temperature parameter for probability
                calculations.
            save_name (str, optional): Name of the file to save intermediate RCP
                history.
            ground_truth (bool, optional): If True, use the next time step as
                ground truth for phi.

        Returns:
            None
        """
        self.t_series = t_series
        self.y_0 = y_0
        self.m = m
        self.seed = seed
        self.k_0 = k_0
        self.k = k
        self.alpha = alpha
        self.gamma = gamma
        self.rand_init = rand_init
        self.T = T
        self.save_name = save_name
        self.ground_truth = ground_truth
        self.dur_hist = []
        self.distortion_hist = []

    def fix_probs(self, p):
        """
        Automatically rebalances probabilities so that all probabilities have
            value greater than or equal to some minimum probability threshold.
            Used to avoid degenerate cases where a cluster size ratio is too small
            or zero.

        Parameters:
            p (ndarray): Array of probabilities.

        Returns:
            ndarray: Array of probabilities with fixed values.
        """
        m = len(p)
        min_prob = 1e-40
        mask_violations = p < min_prob

        mask_rest = np.logical_not(mask_violations)
        violation_locs = np.where(mask_violations)
        rest_locs = np.where(mask_rest)

```

```
        n_violations = len(violation_locs)
        n_rest = m - n_violations
        residual = (n_violations * min_prob) / n_rest

        for loc in violation_locs:
            p[loc] = min_prob

        for loc in rest_locs:
            p[loc] -= residual
        return p

    def rcp(self):
        """
        Run the RCP algorithm.

        Returns:
            tuple: A tuple containing RCP history, duration history, and
                distortion history.
        """
        if not np.all(self.y_0) and not self.m:
            raise Exception('y_0 and m cannot be both None')
        elif not np.all(self.y_0):
            x_0 = self.t_series[0]
            np.random.seed(self.seed)
            rnd_indices = np.random.choice(len(x_0), size=self.m, replace=False)
            self.y_0 = x_0[rnd_indices, :]

        self.m = len(self.y_0)

        if self.rand_init:
            np.random.seed(self.seed)
            eps = np.random.randn(self.m, 2) * 1e-1
            self.y_0 = self.y_0 + eps

        horizon = len(self.t_series)
        alpha = self.alpha
        k_0 = self.k_0
        n = self.t_series[0].shape[0]
        eta = self.gamma * (self.m - 1) + 1
        i2 = np.eye(2)

        theta = np.zeros((2 * self.m, 2 * self.m))
        for i in range(self.m):
            for j in range(self.m):
                i_start = i * 2
                i_end = i_start + 2
                j_start = j * 2
```

```
                j_end = j_start + 2
                if i == j:
                    theta[i_start:i_end, j_start:j_end] = np.kron(eta, i2)
                else:
                    theta[i_start:i_end, j_start:j_end] = np.kron(-self.gamma, i2)

        def d(x, y):
            return norm(x - y) ** 2

        y = self.y_0
        rcp_hist = []

        for t in range(horizon - 1):
            t0 = time()
            rcp_hist.append(y)
            x = self.t_series[t]
            if self.ground_truth or t == 0:
                phi = self.t_series[t + 1] - self.t_series[t]
            else:
                phi = self.t_series[t] - self.t_series[t - 1]

            phi_f = phi.flatten()
            D = np.array([[d(x_, y_) + self.gamma * sum(d(y_, _y_) for _y_ in y)
                for y_ in y] for x_ in x])
            temp = np.array([[-(d(x_, y_) + self.gamma * sum(d(y_, _y_) for _y_
                in y)) / self.T for y_ in y] for x_ in x])
            c = np.max(temp, axis=1).reshape((-1, 1))
            temp -= c
            p_yx = np.exp(temp)
            Z = np.sum(p_yx, axis=1).reshape((-1, 1))
            p_yx = p_yx / Z
            p_yx_ = np.kron(p_yx, i2)

            distortion = np.sum(np.multiply(D, p_yx))
            self.distortion_hist.append(distortion)

            p_x_temp = (np.ones(n) * (1 / n)).reshape((-1, 1))
            p_x = np.diag(np.ones(n) * (1 / n))
            _p_y = self.fix_probs(np.sum(p_yx * p_x_temp, axis=0))
            p_y = np.diag(_p_y)
            p_y_ = np.kron(p_y, i2)

            p_xy = p_x @ p_yx @ inv(p_y)
            p_xy_ = np.kron(p_xy, i2)
            x_f = x.flatten()
            y_f = y.flatten()
            y_ = n * theta @ (y_f - inv(theta) @ p_xy_.T @ x_f)
```

```
            term1 = x_f - y_f @ p_yx_.T
            term2 = y_ @ p_y_ @ y_
            u = -(k_0 + (term1 @ phi_f) / term2) * y_
            y_f += u
            y = y_f.reshape((self.m, 2))
            save_pickle(rcp_hist, f'temp_data/{self.save_name}.pickle')
            self.T *= alpha
            k_0 *= self.k

            self.dur_hist.append(time() - t0)

        return rcp_hist, self.dur_hist, self.distortion_hist
```

Listing A.2: Synthetic dataset generation

```
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt
import numpy as np
from tools.plots import plot_system
from tools.file_ops import save_pickle

class SyntheticDatasetGenerator:
    def __init__(self):
        pass

    def noise(self, rng: float = 5e-3, size: tuple = (1, 1), noisy: bool = False)
        -> np.array:
        """
        noise function
        :param rng: range of collected samples
        :param size: number of collected samples
        :param noisy: whether to add noise or not
        :return: numpy array
        """
        if noisy:
            return np.random.uniform(-rng, rng, size=size)
        else:
            return np.zeros(size)

    def exp_func(self, t: int, x_0: np.array, x_1: np.array, k: float, T: int,
        noisy: bool = False) -> np.array:
        """
        return location at time 't' according to first order linear dynamics
        :param t: time to return location
        :param x_0: starting location
        :param x_1: end location
        :param k: velocity parameters
```

```
        :param T: time horizon
        :param noisy: whether to add noise or not
        :return: numpy array
        """
        d = x_0.shape[0]
        x_t = (x_0 - x_1) * np.exp(-k * t / T) + x_1
        return x_t + self.noise(size=(d,), noisy=noisy)

    def lin_func(self, t: int, x_0: np.array, x_1: np.array, k: np.array, T: int,
        noisy: bool = False) -> np.array:
        """
        return location at time 't' according to a linear trajectory
        :param t: time to return location
        :param x_0: starting location
        :param x_1: end location
        :param k: velocity parameters
        :param T: time horizon
        :param noisy: whether to add noise or not
        :return: numpy array
        """
        d = x_0.shape[0]
        x_t = ((x_1 - x_0) / T) * t + x_0
        return x_t + self.noise(size=(d,), noisy=noisy)

    def loc_func(self, t: int, idx: int, noisy: bool = False) -> np.array:
        """
        calculates current location of a point given time 't'
        :param t: time to return location
        :param idx: index of point in dataset
        :param noisy: whether to add noise or not
        :return: numpy.array
        """
        result = self.calc_func(t, self.X[idx], self.X_[idx], self.K[idx],
            self.T, noisy=noisy)
        return result

    def move(self, t: int, noisy: bool = False) -> np.array:
        """
        creates a (n,2) matrix representing location of points at time t
        :param t: time to return location
        :param noisy: whether to add noise or not
        :return: numpy.array (n,2)
        """
        X_new = np.zeros((self.n, self.d))
        for idx in range(self.n):
            X_new[idx, :] = self.loc_func(t, idx, noisy=noisy)
        return X_new
```

```
76
77      def gen_dataset(self, seed1: int = 20, seed2: int = 22, n_samples: int = 300,
            centers: int = 3,
78                      cluster_std: int = 1, T: int = 100, case: str = 'exp', bound:
                            float = 1,
79                      compactness: float = 12, speed_var: float = 1e-3,
                            open_destination: bool = False,
80                      noisy: bool = False, save: bool = True, animate: bool = True)
                            -> dict:
81          """
82          generates synthetic dataset
83          :param seed1: random seed for starting clusters
84          :param seed2: random seed for end clusters
85          :param n_samples: number of sample points
86          :param centers: number of clusters in dataset
87          :param T: number of simulation steps
88          :param case: dynamics of points, either 'exp' or 'lin'
89          :param bound: system is approximately constrained in [-bound, bound] box
90          :param compactness: compactness of generated clusters
91          :param speed_var: adds variety to speed of points
92          :param open_destination: if points are matched with a corresponding
                cluster
93          :param noisy: add noise to point dynamics
94          :param save: save generated dataset
95          :param animate: plot synthesis progress
96          :return: {'args':args, 'history':history, 'labels':labels, 'K':K}
97          """
98          # args used to create the dataset. Returned for reproduction purposes
99          args = locals()
100
101         # orign clusters centers are scattered around (-bound, 0) [south-west]
102         self.X, labels = make_blobs(n_samples, centers=centers,
103                                     n_features=2, random_state=seed1,
104                                     center_box=(-bound, 0),
                                            cluster_std=np.array(cluster_std) /
                                            compactness)
105         # destination clusters centers are scattered around (bound, 0)
                [north-east]
106
107         self.X_, labels_ = make_blobs(n_samples, centers=centers,
108                                       n_features=2, random_state=seed2,
109                                       center_box=(0, bound),
                                              cluster_std=np.array(cluster_std) /
                                              compactness)
110
111         self.n, self.d = np.shape(self.X)
112         self.label_vals = np.unique(labels)
```

```
        # velocity parameters of each point
        self.K = np.zeros((self.n, self.d))

        # Generating stochastic dynamics parameters
        for label in self.label_vals:
            # stores indices of points of each cluster
            indices = np.array(np.argwhere(labels == label)).flatten()
            # cluster size
            size = len(indices)
            np.random.seed(seed1)
            # horizontal velocity parameters
            K1 = speed_var * np.random.rayleigh(0.5, size=(size, 1)) + 0.015
            np.random.seed(seed2)
            # vertical velocity parameters
            K2 = 2 * np.random.uniform(0.7, 1) * K1
            K_ = np.concatenate((K1, K2), axis=1)
            self.K[indices] = K_.copy()

        # indices of labels for starting and end clusters
        self.indices = {label: labels == label for label in self.label_vals}
        self.indices_ = {label: labels_ == label for label in self.label_vals}

        # Set the calculation function based on the case
        if case == 'lin':
            self.calc_func = self.lin_func
        elif case == 'exp':
            self.calc_func = self.exp_func

        # a list of size 'T' with each element being a (n,2) matrix
        history = []

        if open_destination is False:
            # re-arranges rows of X_ so that matching labels are in the same rows
                with X
            temp = self.X_.copy()

            for label in self.label_vals:
                self.X_[self.indices[label]] = temp[self.indices_[label]]

        # moving through time horizon
        for t in range(0, T):
            X_new = self.move(t, noisy=noisy)
            history.append(X_new)
            time_series = {'args': args, 'history': history, 'labels': labels,
                'K': self.K}
            if save == True:
                save_pickle(time_series, 'temp_data/time_series.pickle')
```

```
            if animate == True:
                plot_system(self.X, X_new, self.X_, t, T)
                plt.clf()
        return time_series
```

Listing A.3: Plotting facilities

```

import matplotlib.pyplot as plt
import matplotlib
import numpy as np
import random
from sklearn.neighbors import KDTree
from typing import List, Union, Optional, Tuple

plt.rcParams.update({'font.size': 16})
matplotlib.rcParams['font.sans-serif'] = "Times New Roman"
matplotlib.rcParams['font.family'] = "sans-serif"

def plot_system(X_0: np.ndarray, X_new: np.ndarray, X_: np.ndarray, t: int, T:
    int) -> None:
    """
    Plot the 2D dynamical system.

    Args:
        X_0 (np.ndarray): Initial data points.
        X_new (np.ndarray): New data points at time t.
        X_ (np.ndarray): Final data points.
        t (int): Current time step.
        T (int): Total time steps.
    """
    x = X_new[:, 0]
    y = X_new[:, 1]
    x_0 = X_0[:, 0]
    y_0 = X_0[:, 1]
    x_1 = X_[:, 0]
    y_1 = X_[:, 1]

    plt.xlabel('x coordinate')
    plt.ylabel('y coordinate')
    plt.title('2D dynamical system (' + str(t) + '/' + str(T) + ')')
    plt.scatter(x, y, color='turquoise', alpha=0.25, edgecolors='black')
    plt.scatter(x_0, y_0, color='cyan', alpha=0.02, edgecolors='purple')
    plt.scatter(x_1, y_1, color='purple', alpha=0.03, edgecolors='cyan')
    plt.minorticks_on()
    plt.grid(b=True, which='major', color='black', linestyle='--', alpha=0.15)
    plt.grid(b=True, which='minor', color='black', linestyle='-', alpha=0.01)
```

```
    plt.pause(1e-3)

def animate_system(t_series: List[np.ndarray]) -> None:
    """
    Animate the 2D dynamical system over time.

    Args:
        t_series (List[np.ndarray]): List of data points over time.
    """
    X_0 = t_series[0]
    X_ = t_series[-1]
    t = 0
    for X in t_series:
        t += 1
        plot_system(X_0, X, X_, t, len(t_series))
        plt.clf()

def plot_nodes(data: np.ndarray, label: str, color: str, alpha: float,
    edgecolors: str, zorder: int = 2) -> None:
    """
    Plot network nodes.

    Args:
        data (np.ndarray): Data points.
        label (str): Label for the legend.
        color (str): Node color.
        alpha (float): Node transparency.
        edgecolors (str): Edge color.
        zorder (int, optional): Z-order for plotting. Defaults to 2.
    """
    x = data[:, 0]
    y = data[:, 1]
    plt.scatter(x, y, color=color, alpha=alpha,
                edgecolors=edgecolors, zorder=zorder, label=label)

def fix_collisions(indices: List[List[int]]) -> None:
    """
    Fix collisions in indices.

    Args:
        indices (List[List[int]]): List of indices.
    """
    len_ = len(indices)
    indices_unique = {item[0] for item in indices}
    buffer = set(range(len_))
    un_assigned = buffer - indices_unique
    for item in indices:
```

```
            n = item[0]
            if n in buffer:
                buffer.remove(n)
            else:
                n_ = random.choice(list(un_assigned))
                item[0] = n_
                un_assigned.remove(n_)
    return None

def realign_paths(paths: np.ndarray) -> None:
    """
    Realign paths to fix collisions.

    Args:
        paths (np.ndarray): Array of paths.
    """
    paths_ = list()
    steps = len(paths)
    for step in range(steps):
        points = paths[step]
        if step > 0:
            indices = points_tree.query(points, k=1, return_distance=False)
            fix_collisions(indices)
            paths[step] = points[indices.flatten()]
        points_tree = KDTree(points)

def plot_paths(paths: np.ndarray, label: str, color: str = 'orange', alpha: float
    = 0.6,
               edge: Optional[str] = None, realign: bool = False, scatter: bool =
                   False,
               s: int = 20) -> None:
    """
    Plot paths.

    Args:
        paths (np.ndarray): Array of paths.
        label (str): Label for the legend.
        color (str, optional): Path color. Defaults to 'orange'.
        alpha (float, optional): Path transparency. Defaults to 0.6.
        edge (Optional[str], optional): Edge color. Defaults to None.
        realign (bool, optional): Whether to realign paths. Defaults to False.
        scatter (bool, optional): Whether to use scatter plot. Defaults to False.
        s (int, optional): Marker size for scatter plot. Defaults to 20.
    """
    if realign:
        realign_paths(paths)
    m = paths.shape[1]
```

```
    paths_ = []
    for path_idx in range(m):
        path = paths[:, path_idx, :]
        paths_.append(path)

    ctr = 0
    for path in paths_:
        x = path[:, 0]
        y = path[:, 1]

        if ctr == 0:
            if scatter:
                plt.scatter(x, y, s=s, marker='.', c=color, edgecolors=edge,
                    alpha=alpha, label
```

Listing A.4: Trajectory prediction

```

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import numpy as np

class LSTMWithFC(nn.Module):
    def __init__(self, input_size: int, hidden_size: int, n_layers: int,
                 fc_hidden_sizes: Tuple[int, ...], output_size: int):
        """
        LSTM model with fully connected layers.

        Parameters:
            input_size (int): Size of the input features for each time step.
            hidden_size (int): Size of the hidden state of LSTM cells.
            n_layers (int): Number of stacked LSTM layers.
            fc_hidden_sizes (Tuple[int, ...]): Tuple of integers representing the
                number of neurons in each fully connected layer.
            output_size (int): Size of the output of the fully connected layers.
        """
        super(LSTMWithFC, self).__init__()
        self.n_layers = n_layers

        # LSTM layers
        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
            num_layers=n_layers, batch_first=True)

        # Fully connected layers
        self.fc_hidden_layers = nn.ModuleList()
```

```
        last_layer_size = hidden_size
        for hidden_layer_size in fc_hidden_sizes:
            self.fc_hidden_layers.append(nn.Linear(last_layer_size,
                hidden_layer_size))
            last_layer_size = hidden_layer_size

        self.fc_output = nn.Linear(last_layer_size, output_size)

    def forward(self, input_data: torch.Tensor) -> torch.Tensor:
        """
        Forward pass of the LSTM model with fully connected layers.

        Parameters:
            input_data (torch.Tensor): Input data as a tensor of shape
                (batch_size, sequence_length, input_size).

        Returns:
            torch.Tensor: Output of the fully connected layers as a tensor of
                shape (batch_size, output_size).
        """
        # LSTM forward pass
        lstm_output, _ = self.lstm(input_data)

        # Extract the last time step's output from the LSTM
        last_output = lstm_output[:, -1, :]

        # Fully connected layers forward pass
        fc_hidden_outputs = last_output
        for fc_layer in self.fc_hidden_layers:
            fc_hidden_outputs = torch.relu(fc_layer(fc_hidden_outputs))

        output = self.fc_output(fc_hidden_outputs)
        return output

def train_model(train_loader, model, criterion, optimizer, device):
    model.train()
    total_loss = 0.0
    for inputs, targets in train_loader:
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()

        total_loss += loss.item() * inputs.size(0)
```

```
73
74      return total_loss / len(train_loader.dataset)
75
76  def validate_model(val_loader, model, criterion, device):
77      model.eval()
78      total_loss = 0.0
79      with torch.no_grad():
80          for inputs, targets in val_loader:
81              inputs, targets = inputs.to(device), targets.to(device)
82
83              outputs = model(inputs)
84              loss = criterion(outputs, targets)
85
86              total_loss += loss.item() * inputs.size(0)
87
88      return total_loss / len(val_loader.dataset)
```