Um algoritmo de força bruta para avaliação de desempenho da minimização de handovers intra-VPON em VCRAN

Lucas Marchesoti Franco

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2018

Lucas Marchesoti Franco

Um algoritmo de força bruta para avaliação de desempenho da minimização de handovers intra-VPON em VCRAN

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Lásaro Jonas Camargos
Coorientador: Gustavo Bittencourt Figueiredo

Uberlândia
2018

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

F825a
2018

Franco, Lucas Marchesoti, 1988-
Um algoritmo de força bruta para avaliação de desempenho da minimização de handovers intra-VPON em VCRAN [recurso eletrônico] / Lucas Marchesoti Franco. - 2018.

Orientador: Lásaro Jonas Camargos.
Coorientador: Gustavo Bittencourt Figueiredo.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://dx.doi.org/10.14393/ufu.di.2018.1224
Inclui bibliografia.
Inclui ilustrações.

1. Computação. 2. Teoria dos grafos. 3. Algoritmos. 4. Otimização combinatória. I. Camargos,, Lásaro Jonas (Orient.). II. Figueiredo, Gustavo Bittencourt (Coorient.). III. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

CDU: 681.3

Maria Salete de Freitas Pinheiro - CRB6/1262

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ata da defesa de DISSERTAÇÃO DE MESTRADO junto ao Programa de Pós-graduação em Ciência da Computação da Faculdade de Computação da Universidade Federal de Uberlândia.

Defesa de Dissertação de Mestrado Acadêmico: PPGCO -09/2018

Data: 29 de agosto de 2018 Hora de início: 10 h 03 min

Discente: Lucas Marchesoti Franco Matrícula: 11612CCP011

Título do Trabalho: Um Algoritmo De Força Bruta Para Avaliação De Desempenho Da Minimização De Handovers Intra-VPON Em VCRAN

Área de concentração: Ciência da Computação

Linha de pesquisa: Sistemas de Computação

Reuniu-se na sala 1B132, Bloco 1B, Campus Santa Mônica da Universidade Federal de Uberlândia, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-Graduação em Ciência da Computação assim composta: Professores doutores: Igor Santos Peretta – FEELT/UFU, Márcia Aparecida Fernandes – FACOM/UFU, Rodolfo da Silva Villaça – DTI/UFES, e Lásaro Jonas Camargos – FACOM/UFU, orientador do candidato.

Ressalta-se que o Prof. Dr. Rodolfo da Silva Villaça, participou da defesa por meio de vídeo conferência desde a cidade de Vitória - ES (Brasil). Os demais membros da banca e o aluno participaram *in loco*.

Iniciando os trabalhos o presidente da mesa Prof. Dr. Lásaro Jonas Camargos apresentou a Banca Examinadora e o candidato, agradeceu a presença do público e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu os conceitos finais.

Em face do resultado obtido, a Banca Examinadora considerou o candidato **aprovado**.

Esta defesa de Dissertação de Mestrado Acadêmico é parte dos requisitos necessários à obtenção do título de Mestre. O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, legislação e regulamentação interna da Universidade Federal de Uberlândia.

Nada mais havendo a tratar foram encerrados os trabalhos às 12 horas e 40 minutos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora

Prof. Dr. Igor Santos Peretta
FEELT/UFU

Participou por meio de vídeo conferência

Prof. Dr. Rodolfo da Silva Villaça
DTI/UFES

Prof.ª Dr.ª Márcia Aparecida Fernandes
FACOM/UFU

Prof. Dr. Lásaro Jonas Camargos
FACOM/UFU (Orientador)

Dedico este trabalho a todos os alunos de cursos de pós graduação que sofrem com depressão ano após ano. A todos que desejam criar um produto acadêmico de qualidade, mas sofrem com a incapacidade, própria e de seus pares, de traçar métricas claras e ancoradas na realidade. Ser relevante é uma arte. Sem a percepção de utilidade é impossível retirar satisfação do trabalho.

Agradecimentos

Agradeço primeiro aos meus orientadores, Lásaro Jonas Camargos e Gustavo Bittencourt Figueiredo, que me apresentaram este instigante tema e me conduziram no desenvolvimento deste projeto. Agradeço à Prof. Márcia Aparecida Fernandes, que acompanhou de perto meu trabalho e contribuiu imensamente com o desenvolvimento teórico. Agradeço ao Prof. Luís Fernando Faina pelos excelentes insights e direcionamentos. Agradeço também ao Prof. Rafael Pasquini, que foi de imensa ajuda sempre que solicitado.

Agradeço às coordenações da Faculdade de Computação (FACOM) da Universidade Federal de Uberlândia, campi Uberlândia e Monte Carmelo, sempre prestativas e compreensivas com a dualidade de conduzir um mestrado enquanto se trabalha. Em especial gostaria de citar o coordenador da FACOM em Monte Carmelo à época em que fiz parte do corpo docente, Kil Jin Brandini Park, e o diretor da FACOM no mesmo período, Ilmério Reis da Silva.

Agradeço ao meu amigo Tuanir França Rezende, com o qual muito discuti sobre a vida acadêmica e o curso de mestrado, e com quem troquei muitas idéias sobre projetos, ferramentas e técnicas. Agradeço ao falecido Fábio Ferreira de Moura, excelente pessoa com a qual desenvolvi meu primeiro trabalho acadêmico, sob a orientação da Prof. Márcia Aparecida Fernandes mencionada acima.

Enfim agradeço à minha esposa, Taynara Tuanne Lima Pereira, que me acompanhou durante a longa jornada para a obtenção do título de mestre e me apoiou estóicamente, sendo tolerante com minhas indecisões e compreensiva com as minhas escolhas.

"Nada do que vivemos tem sentido, se não tocarmos o coração das pessoas."

Cora Coralina

Resumo

Arquitetura de redes 5G utilizando Acesso de Rádio em Nuvem Virtual permite movimento suave entre *base stations* utilizando transmissão conjunta, mas transições ainda podem acontecer dependendo da configuração da rede e dos recursos alocados. Os dois problemas de alocação de recursos nesta arquitetura são formação de Redes Ópticas Passivas Virtuais e Alocação de Banda de Recursos, para os quais os trabalhos recentes se focam em resolver com técnicas de Programação Linear Inteira. Nós formulamos os problemas em termos de Teoria de Grafos Temporais e propomos um algoritmo força-bruta e um heurístico para resolver o problema de Alocação de Banda de Recursos, ambos os quais investigam apenas soluções válidas. Nosso algoritmo pode ser utilizado para avaliar o desempenho de outros algoritmos cujo objetivo é decidir as atribuições em tempo de execução. Nós também investigamos os algoritmos em cenários realistas, mas cenários com tamanho e complexidade suficiente para serem interessantes se provaram demasiadamente grandes tanto para nossos algoritmos quanto para os de trabalhos relacionados.

Palavras-chave: Teoria dos Grafos, Redes 5G, VCRAN, Teoria de Grafos Temporais, Otimização.

A brute-force algorithm to evaluate the performance of VCRAN intra-VPON handover minimization

Lucas Marchesoti Franco

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA – UFU
FACULDADE DE COMPUTAÇÃO – FACOM
PROGRAMA DE PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO – PPGCO

The undersigned hereby certify they have read and recommend to the PPGCO for acceptance the dissertation entitled **Um algoritmo de força bruta para avaliação de desempenho da minimização de handovers intra-VPON em VCRAN** submitted by **Lucas Marchesoti Franco** as part of the requirements for obtaining the **Master's degree in Computer Science**.

Uberlândia, 29 de Agosto de 2018

Supervisor: _____
Prof. Dr. Lásaro Jonas Camargos
Universidade Federal de Uberlândia

Cosupervisor: _____
Prof. Dr. Gustavo Bittencourt Figueiredo
Universidade Federal de Uberlândia

Examining Committee Members:

_____
Prof. Dr. Igor Santos Peretta
Universidade Federal de Uberlândia

_____
Prof. Dr. Rodolfo da Silva Villaça
Universidade Federal do Espírito Santo

_____
Profa. Dra. Márcia Aparecida Fernandes
Universidade Federal de Uberlândia

Abstract

5G Network architecture using Virtual Cloud Radio Access enables seamless movement across *base stations* via Joint Transmission, but Handovers still occur depending on the network configuration and allocated resources. In VCRAN, two resource allocation problems are present, Virtual Passive Optical Network Formation and Resource-Block Assignment, which existing works focus on solving with Integer Linear Programming techniques. We have formulated the problems in terms of Temporal Graph Theory and propose a brute-force and a heuristic algorithm for solving the Resource-Block Assignment problem, both of which explore only valid solutions. Our algorithm can be used to evaluate the performance of other algorithms whose aim is deciding assignments on the fly. We have also investigated realistic scenarios, but ones with sufficient complexity and size to be of interest proved to be large for both our approach and that of related works.

Keywords: Graph-Theory, 5G Networks, VCRAN, Temporal Graph Theory, Optimization.

List of Figures

List of Tables

Acronyms list

AL Adjacency List

BS Base Station

CRAN Cloud Radio Access Network

DRAN Dynamic Radio Access Network

DU Digital Unit

HO Hand Over

ILP Integer Linear Programming

JT Joint-Transmission

LP Linear Programming

LTE Long Term Evolution Architecture

MT Mobile Terminal

OFDM-PON Orthogonal Frequency Division Multiplexing - Passive Optical Network

RAN Radio Access Network

RB Resource Band

RB-A Resource Band Assignment

SD ScenarioDescriptor

SDI ScenarioDescriptor Index

TVG Time-Varying Graph

TGT Temporal Graph Theory

TO Transmission Overhead

VCRAN Virtual Cloud Radio Access Network

VPON Virtual Passive Optical Network

VF VPON Formation

List of Algorithms

Contents

I hereby certify that I have obtained all legal permissions from the owner(s) of each third-party copyrighted matter included in my thesis, and that their permissions allow availability such as being deposited in public digital libraries.

Lucas Marchesoti Franco

Chapter 1

Introduction

4G networks incur service disruption when a MT moves between Base Stations (BSs). This event is called a Handover and it occurs because no two BSs exchange messages with the same device at the same time using the same radio frequency. It follows the break before make principle, that is, the connection gets broken before the next BS takes over transmission to the mobile device. Virtual Cloud Radio Access Network (VCRAN) is one prominent proposal for 5G network architectures that aims to improve this by allowing joint transmission to a single MT, but the technique cannot guarantee that service will always be seamless. It centralizes signal processing from BS in the cloud by using so-called digital units. Any single digital unit and its affiliated base stations constitute a Virtual Passive Optical Network (VPON). Due to architecture limitations, transitioning between VPONs Inside the same VPON, however, these might be avoided if the communication resource, also known as Resource Block (Resource Band (RB)) being used by the MT is not being used by another MT in the vicinity of any newly transmitting BS.

Current methods used to solve these problems are generic and slow, as they are based on Linear Programming LP (WANG et al., 2016a). In (ZHANG et al., 2017) the authors devise a minimum-cut algorithm to reconfigure the network and reduce inter-VPON traffic. In (TININI et al., 2017) the authors use an Integer Linear Programming (ILP) algorithm to accommodate processing in VPONs and turn functions on or off in an energy-efficient way. The authors of (BASU et al., 2010) also try to allocate VPON resources in an energy-efficient way, but they use graph partitioning and rejoining techniques. Other related works focus on proposing architectures to improve communication efficiency and reducing overhead. In (YANG; CHAN, 2017) the authors propose a switching architecture to protect remote radio heads against being overwhelmed. In (RIVA et al., 2018) the authors propose an OFDM-PON architecture with VCRAN to improve bandwidth utilization when the channel is bigger than the demand. In (YU et al., 2018) the authors propose an architecture to improve utilization of the network's optical transceivers and an algorithm to improve energy utilization.

Existing works focus on scenarios with set boundary-crossing rates in order to generate

high complexity situations in small setups, ignoring the possibility of generating realistic movement patterns by using random walk and manhattan grid, for example. In order to make our work comparable, we will focus on the former approach. We think that graph and temporal graph theory have better prospects of producing an acceptable algorithm compared to previous LP formulations, and that further progress is possible beyond classic graph algorithms.

In this work we explore the problem of benchmarking handover minimization in VCRAN architectures. We aim to establish a baseline for evaluation of algorithms that will be used in a production system. For this end we formulate the problem in terms of Time-Varying Graphs (TVG) and develop a brute-force algorithm that identifies assignments with minimum number of handovers. We use Temporal Graph Theory (TGT) for deriving the algorithm, where a first approach using hypergraphs is presented, but due to time complexity constraints we also develop a simpler one. We devise and implement a brute-force algorithm and run it against generated scenarios, comparing it with a random pruning strategy, to find the optimal RB allocation in order to reduce handovers. The brute-force algorithm is capable of evaluating only valid assignments. We judge them based on time complexity, execution time and number of optimal solutions found.

This work is divided as follows. We first present the theoretical fundamentals, then move on to define the network problem in terms of TGT. Based on these definitions we formalize the problem both in temporal hypergraphs and temporal graphs, and move on to derive a brute force algorithm along with its complexity analysis. We describe the implementation and test it in generated scenarios along with a random pruning approach. We then analyze the results and present the conclusions.

Chapter 2

Fundamentals

2.1 5G Networks

5G Networks need to improve over current Radio Access Networks (RAN) to support orders of magnitude more traffic, along with unique services and devices, specific quality of experience requirements and latency constraints. As the number of connected devices explodes the network is expected to become overloaded.

The solution to this problem is usually to increase network Base Station density. Current Distributed RAN (DRAN) fails to scale in this respect because the shrunk area cells would cause excessive handovers (HO) as mobile terminals (MTs) move. To make matters worse, the break before make nature of these handovers asks that connection be dropped before it is established with the new base station, resulting in more overhead.

To avoid network thrashing we need a new architecture that enables MT movement across base stations with seamless handovers. The development of such a network is possible due to recent advances in cloud computing and virtualization.

2.1.1 Virtual Cloud Radio Access Network

In traditional Distributed RAN, baseband processing is done at each BS, thus each time an MT moves from one area to another there is a HO delay to move baseband processing from one BS to another. In Cloud RAN (CRAN) the baseband processing is moved from the BS to the Digital Unit (DU). Furthermore, if this setting is centralized in commodity hardware and scaled through virtualization, we have a Virtual CRAN (VCRAN). This way we can perform baseband processing in a centralized machine and coordinate handovers locally. Now when an MT moves from one area to another, traffic for that MT just needs to be routed through a different BS.

VCRAN architecture is shown in Figure 1. We can see the BSs which communicate with MTs being aggregated in VPONs, or virtual base-stations (V-BS), by their respective DUs, which process MT packets in the DU Cloud. DUs communicate between themselves

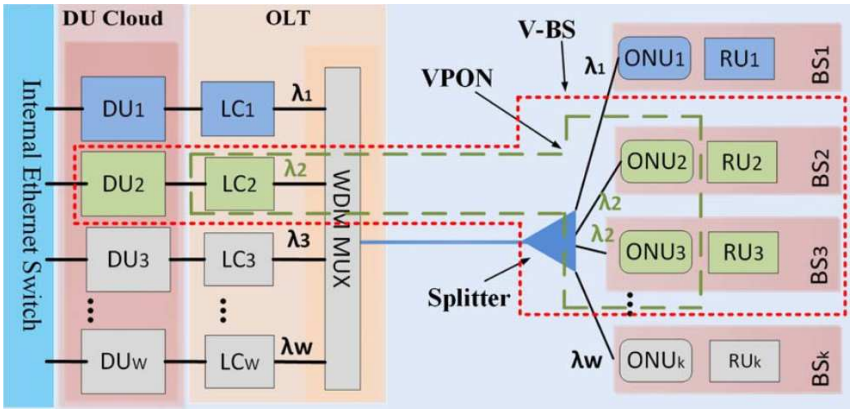

Figure 1 – VCRAN architecture (WANG et al., 2016b).

through an internal ethernet switch. Each time a handover occurs between base stations of different DUs, the MT information must be handled from one digital unit to another, thus a handover cannot be avoided in this situation. Conversely, when an MT moves between two BSs that report to the same DU, then the handover will only occur on the MT side if necessary. BSs will be able to communicate seamlessly with the MT without additional overhead.

There are two other essential pieces to support this architecture: the FrontHaul, a high bandwidth, low latency and strict jitter network to communicate data between BSs and DUs; and the ability to Joint-Transmission (JT), a method to coordinate signal redirection from one RU to another, without the need to break connection.

2.1.2 FrontHaul

The FrontHaul can be built in a way that satisfies our constraints by using Time-wavelength division multiplexing in a passive optical network. By aggregating certain fibers under the same Virtual Passive Optical Network we can support baseband processing for a certain network area under the same digital unit. Thus if we coordinate transmission between different base stations to a moving MT in a way that service isn't disrupted as it changes area cells, also known as Coordinated Multi-Point, we avoid HOs altogether inside VPONs (WANG et al., 2016a).

2.1.3 Joint Transmission

Joint-Transmission is a coordinated multi-point technique that works by enabling multiple base-stations to transmit together to the same MT (WANG et al., 2016a). Signal is duplicated and sent to both base-stations, which coordinate their scheduled transmissions in order to strengthen the signal for that MT in an area that would be ill-served by either BS alone.

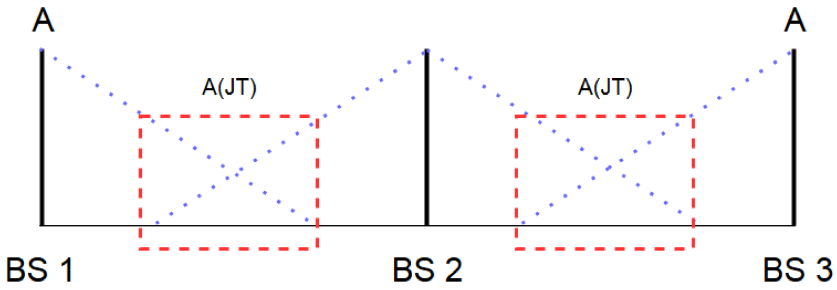

Figure 2 – Joint Transmission in action: BS 2 tries to communicate with MTs in both BS 1 and BS 3.

Figure 2 shows Joint Transmission in action. It depicts how BS 2 would communicate with MTs in BS 1, if any, with the same RB used by its native BS. The same would occur with BS 3. Any MT in the highlighted box would receive its signal from both its surrounding BSs. Notice that if both BS 1 and BS 3 are using the same RB, then joint transmission cannot occur, as BS 2 would be communicating with 2 different MTs using the same RB.

On the one hand, the technique enables seamless movement of any MT on the network, on the other hand it might incur transmission overhead or another form of handover. The transmission overhead is caused by performing JT to an MT when using BSs from more than one VPON. Since digital units are in the cloud and connected by an internal ethernet switch, it is possible to perform this operation, but it entails additional communication costs, which we would like to minimize.

The new form of handover has to do with the freedom an MT now has to walk among BSs using the same frequency assigned to it when it first connected. Unless we have an unlimited number of available frequencies, this is bound to generate some conflicts. When two MTs attempt to communicate through the same BS using the same Resource Block (RB), the JT signal becomes interference and, therefore, we need to perform a handover in one of the MTs to avoid this clash. This operation also incurs additional overhead on the network. Note that the RB assigned to the MT is independent of the VPON of its transmitting BSs, and this is a resource allocation problem to be solved globally. In this case we want to minimize RB allocation changes among moving MTs. These problems have been tackled using Integer Linear Programming (ILP) in a static setting and Heuristics in a dynamic setting (WANG et al., 2016a; YU et al., 2013).

2.1.4 Resource Allocation Problems

There are two prominent problems of resource allocation using the aforementioned architecture. The first one concerns the attribution of Base Stations to Digital Units. Each digital unit has a limited processing capacity, and there must be a HandOver every time an MT crosses borders between BSs from different digital units, called an Inter-

VPON HandOver. This topological assignment, called VPON Formation (VF) can heavily impact network performance. The second problem concerns the attribution of Resource Blocks to individual MTs inside each VPON, where a RB is the time-wavelength slot used to communicate with the MT. We want to minimize RB conflicts upon MT movement, each of which would cause an Intra-VPON HandOver. This problem is called Resource Block Assignment (RB-A). The number of available RBs is dependent upon the specific equipment being used by the network.

2.2 Graph Theory

We base our initial investigations in classic graph algorithms. We will assume planarity of the graph topology due to the network layout.

2.2.1 Hypergraphs

Hypergraphs are an established branch of graph theory relating to graphs that have edges linking more than 2 nodes at once. Common hypergraph problems related to ours are the k-way hypergraph cut problem (CHEKURI; LI, 2015), colouring planar mixed hypergraphs (KUNDGEN; MENDELSOHN; VOLOSHIN, 2001) and total colouring of hypergraphs (COWLING, 1995). In these works, it has been shown to be hard approximation problems for which algorithms are still immature. We have not found any papers that attack the specific problem restrictions we run into.

In terms of algorithms, (KAZNACHEY; JAGOTA; DAS, 2003) enumerates some greedy algorithms for graph coloring and provides a neural network model based on Hopfield networks and an adaptation of the primal-dual method for training. Among the presented heuristics, GR2 seems especially fit for our objectives. It is a uniform-coloring algorithm based on always picking the least-used color. It can be easily adapted for multi-coloring, and the order of picking elements can also be adapted to satisfy full-coloring, by using the chromatic number of each element instead of using edges as pivots to choose which vertex to color.

2.2.2 Temporal Graph Theory (TGT)

Temporal graphs appear in literature as a novel way to understand dynamic networks. Static graphs have been used to model static networks, but the dynamic nature of some applications prohibits the system from being completely understood in a static vision. Adding a temporal dimension to graphs opens new opportunities to understand such systems, and at the same time poses a new challenge.

The idea of temporal graphs has been applied independently in various works. Most closely related are fault tolerant networks, where a connection might be present or not at

a given moment. Another close theme is opportunistic mobile networks, where a message travels according to available connections, which appear in function of agent movement. In the following paragraphs we borrow from (CASTEIGTS et al., 2010) to describe Time-Varying Graphs (TVG) and related concepts.

A Time-Varying Graph is defined as $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta, \mathcal{L})$, where V is a set of nodes, $E : V \times V$ is a set of edges (or relationships between nodes), $\mathcal{T}$ is the system lifetime, $\rho : E \times \mathcal{T} \rightarrow \{0, 1\}$ is a binary function relating the presence of each edge with time, and $\zeta : E \times \mathcal{T} \rightarrow \mathcal{T}$ is a latency function, indicating the cost of each edge at each moment in time. $\mathcal{L}$ is the label set, which tags each element on the hypergraph with domain-specific properties.

Just as in Static Graph Theory, TGT establishes some base-concepts that are useful when working with TVGs. For example, the *base graph* of a TVG is the graph representing all nodes and all edges that appeared at some moment in the lifetime of a TVG. Some works refer to it as the *smashed* representation(BASU et al., 2010).

The *evolution* of a TVG may be seen from 3 points of view: edges, nodes and the graph. Edge evolution happens as edges appear and disappear from the graph. The union of all these appearance and disappearance dates forms the set of *characteristic dates* of the TVG. Node evolution happens as the neighborhood of each node changes. Graph evolution is a sequence of states, or *snapshots*, where each state is different from the previous one. Note that for discrete-time applications we might establish that the only variation is time, making it possible for two adjacent snapshots to be equal. In the same fashion, a TVG *subgraph* might consist of a subset of nodes, edges and time.

The analogous of a *path* in TGT is a *journey*. A journey is a temporal path, that is, a path that is satisfied over the lifetime of a TVG. Crossing an edge is subject to it being available over a period that overcomes its latency. We can then view a journey as happening both *topologically* and *temporally*. The topological distance of a journey is the number of hops to reach a destination, while the temporal distance is the interval between departure and arrival. If there is a journey from u to v, we say that u *reaches* v. The set of all nodes reachable from u is the *horizon* of u.

According to the concept of a journey we can also see the distance between two nodes topologically and temporally. In terms of shortest paths we could then have three metrics: The *shortest* path is the one with least topological distance; the *fastest* path is the one with least temporal distance; and the *foremost* path is the one with closest arrival date. Note that all of these are taken from a base time t, so there is a set of such paths for every pair of nodes at every time t.

The basic concepts in TVG can be analyzed to create classes, each of which guarantees certain properties, that form a hierarchy. For example, two desirable characteristics are universal temporal source and universal temporal sink. These establish, respectively, a node that can send a (temporal) message to any other node and that can receive a

(temporal) message from any other node. These two characteristics together lead to the presence of connectivity over time, a third property that establishes that any node can reach any other node. Strengthening this a bit, we get to establishing temporal cyclic connectivity, that is, every node can send a message and receive an answer from the same node, to any node in the graph.

The class of a TVG can be recognized by analyzing alternative views of the TVG, such as connected components over time and the transitive closure. In the same way, classic properties can be analyzed in other ways when seen through a formal temporal viewpoint.

Non-determinism can be introduced on a TVG by making the presence function return the probability that an edge exists at time t, that is, $\rho : E \times \mathcal{T} \rightarrow [0, 1]$. Non-determinism in discrete TVGs is modeled as a markovian process, while in continuous graphs it is modeled as a graph where edge appearance follows a Poisson process.

Research opportunities in TVG abound, especially when related to distributed algorithms, self-stabilization techniques, design and optimization of TVG topology, complexity analysis of algorithms in TVGs, pattern detection and TVG visualization (CASTEIGTS et al., 2010).

Chapter 3

Problem Formulation

We start by defining a mathematical model of the system's inner workings in terms of TGT. We investigated Temporal Graph Theory and modeled the VF and RB-A problems using it, contributing to Temporal Graph Theory in any respect it lacks in modeling our problem. We also developed novel algorithms based on TGT and built prototypes to validate our solutions against simulated network data.

Our initial formulation is based on hypergraphs. We proceed to work on a simple graph model due to complexity constraints, but still present the first formulation here, as it is valid and might be further explored in future works.

3.1 Base concepts

3.1.1 System model

We begin the definition of a system state by first delimiting the time-domain where the system will function. Then we properly detail the information the system needs to store at each state and how it is done.

3.1.1.1 Time-domain

Mobile Terminals move in the physical world with a continuous time domain. Even then, the network system works in cycles and uses a discrete time domain. We address this conflict by considering the system clock to be fast enough to detect MT movements between adjacent signal areas any time they happen.

In other words, we can assume a system window time and a minimum time to cross a signal area (i.e., the time required to traverse two edges to reach a non-adjacent cell) such that $system\_window\_time \leq time\_to\_cross$. We define $time\_to\_cross$ in function of the signal area traversal distance and the max MT speed (e.g. $10m/s$) as

$$time\_to\_cross = \frac{signal\_area\_traversal\_distance}{max\_mt\_speed}$$

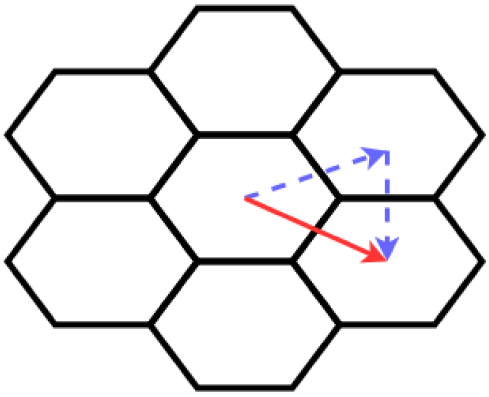

Figure 3 – MT movement scenario illustrating how MT movement is discretized in a time-frame. The blue arrows show the actual MT movement, the red arrow shows the MT movement as detected by the network in a single time frame transition.

for the general formulation:
$system\_window\_time \times max\_mt\_speed \leq signal\_area\_traversal\_distance$. Note that when the MT moves twice (or more) between different adjacent signal areas, then only one movement is captured. This last case is depicted in Figure 3.

Let's work through an example to understand that we can work with signal areas instead of the full network cell. We can do some math and show that considering a reasonable max MT speed, we also have a reasonable size signal area that cannot be crossed twice between time steps. Suppose we have an MT traveling at $10m/s$, and our $time\_to\_cross$ is $10ms$, then by substituting these values into our formula, and considering the safe values as the ones where the time to cross is more than the available transmission frame, we have:

$$\frac{signal\_area\_traversal\_distance}{10m/s} > 10 * 10^{-3}s$$

$$signal\_area\_traversal\_distance > 10^{-2}s * 10m/s$$

$$signal\_area\_traversal\_distance > 0.1m$$

Thus our signal area needs to have a diameter of more than $100cm$, which is clearly the norm.

These bounds will be useful when considering MT movement, thus not allowing them to "leap", which presumably makes movement prediction considerably harder.

3.1.1.2 System information

Internally we assign wavelengths to sets of cells, which constitutes a VPON. These wavelengths are used to establish communication between the BS and the network back-end. We also assign a Resource Block to each MT, which will be used when transmitting to it by all BSs in range. Each MT is also associated with a Home BS, its closest Base Station, which determines its VPON.

VPON formation is restricted by wavelength capacity, as all information from the BS must fit into the bandwidth of the wavelength used for transmission.

Non-adjacent groups of cells are considered to be different VPONs even though they might share the same wavelength. This is done to improve resource use while reducing system complexity: a detached VPON is the same as two separate VPONs, but this does not create wavelength interference. In practice both groups might be treated by the network as the same VPON, as signal processing is split by wavelength. A complete system state is then characterized by the definition of two functions: one that maps cells to wavelengths and another one that maps resource bands to MTs inside each VPON.

3.1.1.3 Handover Triggering and Transmission Overhead

In order to coordinate JT between base stations we have two situations that cause overhead (and potential denial of service) in the network, which are desired to be avoided. We shall call them Transmission Overhead (TO) and Handover (HO). We would like to minimize the total overhead incurred by these situations.

Joint Transmission poses a single requirement upon the network: we cannot transmit to different MTs using the same RB where it would cause interference. This situation arises when two BSs are close enough that their cover area overlaps, and they try transmitting to different MTs using the same RB. Handovers are triggered when the movement of MTs using the same resource block would result in interference. In this case we need to change one of the MTs' RB to a free one in order to avoid interference.

Transmission Overhead is caused by two BSs in different VPONs transmitting to the same MT. While possible, this incurs communication overhead in the central office, because the DUs of both VPONs must coordinate and transmit the baseband signal through the DU cloud's internal ethernet switch. This overhead is present at each time step there is a transmission in this situation.

3.1.1.4 Time-Varying Graph Model

We provide the following TVG model for this problem. Consider a base TVG $\mathcal{B} = (V, E, \mathcal{T}, \mathcal{L})$ where V is the set of cell signal areas and E is the set of edges $\{u, v\}$, where u and v are adjacent signal areas. A signal area is any delimited spot on the map for which the set of RBs that transmit to that spot is distinct from all adjacent areas. $\mathcal{L}$ is the Label set, which contains the base-stations that serve each signal area. A signal area serviced by more than one BS is considered an interference area. Also $\mathcal{T}$ is the time domain. We do not define a presence function ρ nor a latency function ζ, as we use the TVG only to observe MT journeys over time. The underlying structure is otherwise static, unless in case of failure, which is not considered here. An example is shown in Figure 4.

We can represent an MT journey on this graph as a journey spanning several such signal areas. The journey starts when the MT connects to the network and ends when it

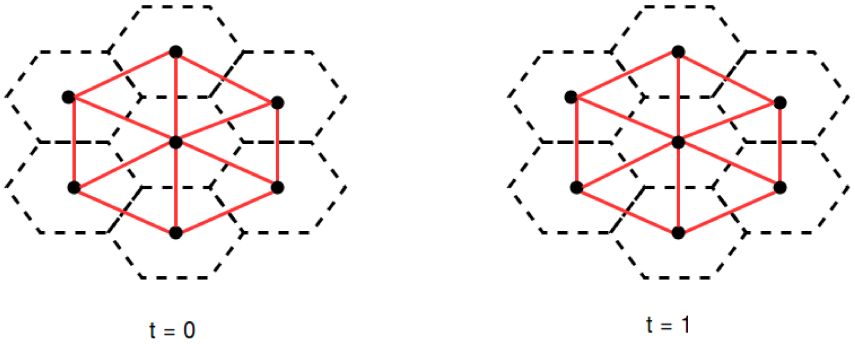

Figure 4 – Example system as a TVG, both in state and snapshot representation.

disconnects. We consider each snapshot to show what happened between samplings. As such we can see that the MT has a new position at each time-step, which might or might not be equal to its last position. We say that if the current time is t, then MT position at time t is its current position. We also consider M to be the set of all MTs in our network throughout its lifetime, individually represented as a journey which can be referenced as M_i. The position of M_i at time t can be referenced as M_i^t. Figure 5 exhibits a sample MT on our network.

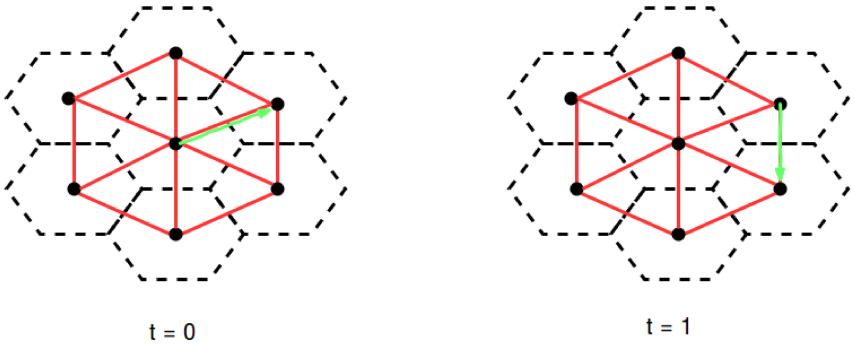

Figure 5 – MT Journey on the network in green.

We dismiss the alternative view of the problem as a bipartite graph, with a set of MTs and a set of BSs, where edges are associations, as it creates a difficult structure to verify adjacency and represent movement in Time-Varying Graphs (TVGs).

3.1.2 Problem

For each of the optimization problems we derive a new TVG $\mathcal{G}$ from $\mathcal{B}$ by removing all edges, and for each interference area, we create a multi-edge joining all its servicing BSs with weight equal to the number of MTs on a journey passing through that area at that time step. We also remove the interference areas. For signal areas with a single servicing BS, we can consider a self-loop, or assign the weight to the node. We favor the first approach for consistency in always dealing with edges. Such a conversion is exemplified in Fig. 6.

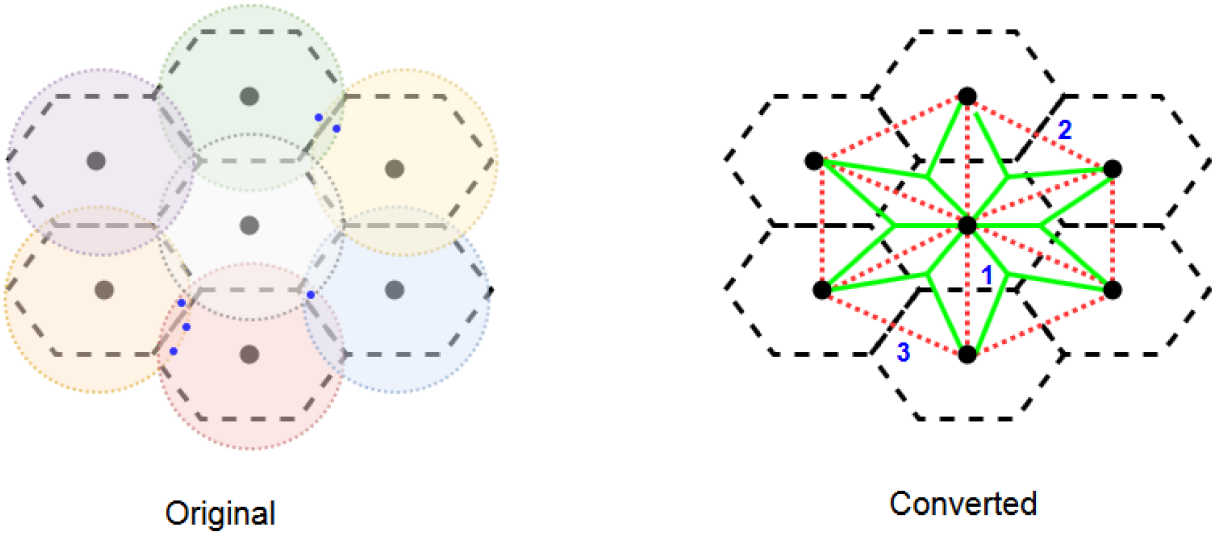

Figure 6 – Example conversion from interference areas to Hypergraph. Green hyperedges connect 3 base stations, while red hyperedges connect two base stations. They are represented differently to improve visualization. Blue MTs on the original network are converted to weights indicated by blue numbers.

Given this graph, in the Resource Block Assignment (RB-A) problem, we consider each element's chromatic number to be equal to its weight, and perform a total multi-coloring. In the VPON Formation (VF) problem we perform a minimum cut using the established weights.

3.1.2.1 RB-A

In order to find an RB Assignment that minimizes handovers we model this problem as a *total multi-coloring* over the TVG $\mathcal{G}$, and solve it for the minimum color changes between time-steps. Total coloring is an operation that colors both vertices and edges of a graph. When used without any qualification, a total coloring is always assumed to be proper in the sense that no adjacent vertices, no adjacent edges, and no edge and its end-vertices are assigned the same color. The total chromatic number $\chi"(G)$ of a graph G is the least number of colors needed in any total coloring of G. Since we might have multiple MTs communicating with the same combination of base-stations, we actually have to deal with a bag of colors in each element. Say we have C_i as the set of colors in node/multi-edge i.

> Like the graph coloring problem, the multi-coloring problem can model a number of applications. It is used in scheduling where each node represents a job, edges represent jobs that cannot be done simultaneously, and the colors represent time units. Each job requires multiple time units (the required number of colors at the node), and can be scheduled preemptively. The minimum num-

> ber of colors then represents the makespan of the instance. Multi-colorings also arise in telecommunication channel assignment where the nodes represent transmitters, edges represent interference, and the transmitters send out signals on multiple wavelengths (the colors). It is due to this application in telecommunications that multicoloring, as well as generalizations that further restrict feasible colorings, dates back to the 1960s. Aardal et al. provide an excellent survey on these problems.
>
> The multi-coloring problem can be reduced to graph coloring by re placing each node by a clique of size equal to the required number of colors. Edges are then replaced with complete bipartite graphs between the corresponding cliques. Such a transformation both increases the size of the graph and embeds an unwanted symmetry into the problem. It is therefore useful to develop specialized algorithms that attack the multi-coloring problem directly.
>
> (MEHROTRA; TRICK, 2007)

Regardless of the algorithm used to solve our problem, a hypergraph total multi-coloring is the most natural representation. It can then be converted to simpler problems with more efficient algorithms for solving.

3.1.2.2 VF Problem

In order to minimize the transmission overhead during the network lifetime, we formulate it as a minimum k-cut problem, where k is unknown and is a function of the total network load. Since we deal with a TVG, a cut might disappear at some point and come to reappear later. Also, since the cost of changing a VPON is very low we can assume the optimal solution is the same as calculating the optimal cut at each time-step. Flexibility in regards to running time can be achieved by m-stacking the graph (YU et al., 2013), but approximation quality is not uniform as different time-steps might delineate different change factors.

3.1.2.3 Practical Considerations

We immediately stumble upon the controversy of needing journey information for MTs that can move freely within the VCRAN. Thus we resort to movement prediction, as we can only perform as well as we can predict future MT movement. At this time we assume an oracle that can correctly predict every movement from the start to the end of each journey.

As computing time is bound to be a bottleneck, per the computational nature of the problem, the *system_window_time* becomes an algorithm choice driver.

The LTE specification establishes a 10ms system window, dictated by the transmission frame time-span (COMPANY, 2009), but a handover can take up to 1 second (IRMER

et al., 2011). In practice, we will probably need to arrange a compromise to cover for this gap, as more details are needed such as, if a handover can take up to 1 second, how long after it started do we need to know the appropriate RB? As such we abstract this notion in benchmarking classical algorithms and work with the network as a perfectly synchronous mechanism, keeping in mind the time-order of execution time.

3.1.3 Method

We focus on solving the RB-A problem through regular coloring algorithms, which provides a good and justifiable approximation to this problem. We choose the RB-A problem because it cannot be avoided, whereas the VF problem can be partially solved offline by analyzing network load and movement statistics. Besides, related work is already tackling the VF problem.

Even though our hypergraph formulation offers the most granular solution, we opt for a simplified model for two reasons: first, it only opens up coloring opportunities from close-by interference areas, which we deem too small; second, it constrains our time requirements even more. Thus we model the problem as a graph node multi-coloring, where each node is a BS and the chromatic number of each is the number of MTs attached to it.

Consider, for example, the situation illustrated in Figure 7. In this scenario, consider that BS1 is transmitting to an MT within area A using some RB. By modeling this situation with a regular graph, BS2 cannot reuse the same RB, although it could. Our hypergraph model captures this situation and would yield a more granular solution.

Even though we devised this model we have decided to implement a simpler approach for a first attempt because of two reasons: first, we already have to deal with other time-constrainers, such as dealing with multiple timesteps; second, these are very close areas, and stochastic probability of the MT moving to an adjacent area that would cause a handover in a close time-step is high. Still, gains are possible and should only be evaluated if possible.

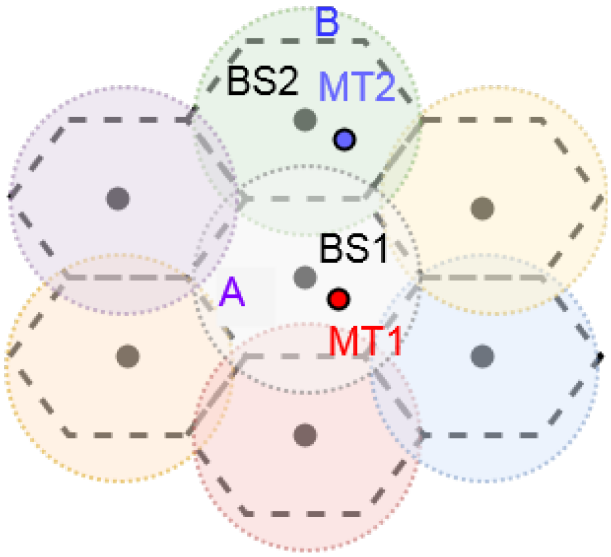

Figure 7 – Two MTs that are not adjacent in the hypergraph formulation, but are adjacent in the Graph formulation. In the hypergraph formulation they would be separated by the interference area between BS1 and BS2. Remember that this formulation only considers as adjacent MTs that are in the same interference area.

Chapter 4

Combinatorial Optimization

We will focus recoloring by considering a couple of adaptations on classic graph coloring algorithms. Suppose, for example, that we are using a standard brute-force approach in which we need to enumerate all possible k colors for all possible vertexes. This gives us $|V|^k$ possible solutions. If we wish to naively check for conflicts, we multiply this by an additional factor of $2|E|$, which is the sum of degrees on the graph.

We can adapt this algorithm to work with multi-coloring by stating that instead of considering each color for each node once, we consider each color for each color slot of each node. In other words, we substitute a node by its chromatic number in the equation, yielding a chromatic number of $\sum_{i=1}^{|V|} \chi_i$ for the graph. Checking conflicts is complicated by checking once against each position in the same node and each position in each of the neighbors. We can do it naively with cost $\chi_i + \sum_{j=1}^{|\eta_i|} \chi_j$ for each vertex i, where η_j is the set of neighbors of j, but it can be sped up by using better data structures.

For partial multi-coloring we keep the same conflict-checking function and assume some of the nodes (or positions) have already been colored. So instead of re-rolling all coloring for each color spot, we re-run it only for the un-colored spots, which reduces our search space to $\sum_{i=1}^{|V|}(\chi_i - \chi'_i)$, where χ'_i is the number of fixed colors in node i. Alternatively, we can denote this as $\sum_{i=1}^{|V|} \chi''_i$, where χ''_i is the number of conflicting colors for node i.

We can apply the same principle to a common heuristic in graph coloring: for each node in turn, pick the smallest non-conflicting color. At first sight, this still yields the same upper bound of $|V|^k$, but a good implementation, using a neighbor list and a color-bit array associated with each vertex, can bring the running time down to $O(|V|^2)$ (KAZNACHEY; JAGOTA; DAS, 2003). Multi-coloring adaptation would result in $O(\sum_{i=1}^{|V|} \chi_i^2)$, and partial multi-coloring cost would be $O(\sum_{i=1}^{|V|} \chi_i''^2)$.

These algorithms form the base for working with a TVG to minimize the number of total handovers over the network service time, expressed by

$$min \sum_{t=1}^{|T|} H_t$$

, where $|H_t|$ expresses the set of handovers occurring at time-step t.

We ran experiments to confirm these theoretical bounds in practice while considering various factors such as data structures that guarantee these theoretical bounds, hidden constant factors that might render them unusable in practice and their applicability given the asynchronous nature of handovers. Our first model will consider only the Manhattan distance, which is the distance between two points by traveling only in parallel to the axes (CHANG et al., 2009).

4.1 Problem Statement

Let S be a bi-dimensional array consisting of the assigned color to each MT at each timestep, and $M = [m_1, m_2, \ldots, m_{|M|}]$ be the set of MTs. Denote S_i^t as the color assignment to m_i at timestep t. We can then define

$$H_t = \sum_{i=0}^{|M|} (S_i^t \neq S_i^{t-1})$$

. Thus, the problem we want to optimize is, in terms of T and M, given by Equation 1.

$$\min \sum_{t=1}^{|T|} \sum_{i=0}^{|M|} (S_i^t \neq S_i^{t-1}) \tag{1}$$

4.2 Naive Brute-Force Optimization

This is a theoretical explanation on optimizing the implementation of the full scale optimization, together with complexity analysis. We highlight that this algorithm is meant to evaluate the scenario. If we wanted the best coloring to be used in a running network, we would be interested in the timestep coloring with the best prospect of reducing future Handovers, repeating and re-evaluating at each timestep.

As a first algorithm we can lay all MTs at each timestep in a single large array, and try each possible combination of color assignments, yielding

$$O\left(\chi^{|M|}\right)$$

possibilities for each timestep, where χ is the number of colors and $|M|$ is the number of MTs on the system. We also have to validate each possibility regarding color clashes inside each BS and among its adjacency. Intra-BS clashes can be calculated by creating a color array for each BS, initialized with zeros, and summing one for each time the color appears in the base station's MTs. It has cost $O(|M|)$, as each MT is only located at one BS and we need only inspect each MT once. Clashes with adjacent BSs can be calculated by applying a logic AND between adjacent base stations' color arrays, for which the result

should be 0 in case of a conflict-free coloring. This step has cost $O(|B|)$, where $|B|$ is the number of base stations.

We need to repeat this for each timestep, so in total the naive algorithm will have cost $O(T) * O(\chi^{|M|}) * (O(|M|) + O(|B|)) = O(T(|M| + |B|)\chi^{|M|})$ for a given χ. If we want to test for multiple χ, we must apply this cost multiple times (i.e. from 1 to our desired maximum χ (C), yielding

$$O\left(\sum_{c=1}^{C} T(|M| + |B|)c^{|M|}\right)$$

.

We can reduce the number of colors to test by proving a lower bound of colors. We can also eliminate the intrinsic BS coloring validation by using a data structure to generate only valid color permutations and incrementing them as needed. For this we will need to shift focus a bit and see the problem on a timestep by timestep basis. We can also eliminate adjacency color validation by using an availability index for each BS. As we generate the colors in order, we will guarantee that the colors available to any base station only depend on the colors already used by the other base stations.

Let's start by looking at optimizing the generated colorings, which present both greater challenge and potential. Other opportunities for optimization include: identifying and ignoring symmetric colorings, for example by hashing already visited states and remembering them in a bloom filter; doing a random walk in the coloring search space.

4.3 Optimizing the combinatorial explosion

4.3.1 Eliminating intrinsic BS coloring validation

Let us start by generating all possible valid colorings on the first timestep. Figure 8 illustrates how we can visualize a single timestep as split by its base stations and contained MTs. This representation will be useful to avoid color assignments that clash inside the same base station.

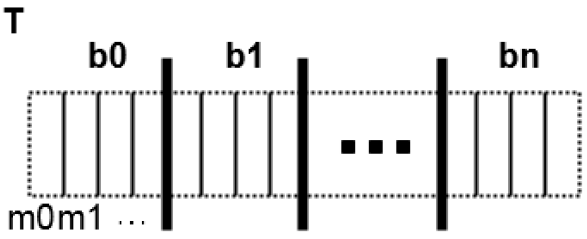

Figure 8 – A single timestep split by its composing base stations, each with its respective MTs.

Let $B = [b_1, b_2, \ldots, b_{|B|}]$ be the list of BSs, and $b_i = [m \in M \mid m \in b_i]$ the set of MTs at base station i. We can generate all non-clashing color assignments for base station i

by generating all color permutations of size $|b_i|$, the number of MTs at base station i. If we do this for each BS, we will need to generate

$$O\left(\prod_{i=1}^{|B|}\frac{\chi!}{|b_i|!}\right)$$

color assignments for this first timestep, where $|B|$ is the number of BSs. This can be shown to be less than the full $O(\chi^{|M|})$ colorings from the naive algorithm. The idea is that instead of each MT considering all possible colors, it considers only the available colors in its BS given the previous assignments. So instead of $\chi * \chi * \ldots * \chi$ we have $\chi * (\chi - 1) * \ldots * (\chi - |b_i|)$ inside the base station.

Moving forward to the next timesteps, we do not need to repeat this whole calculation. It is both a network constraint and a mathematical advantage that MTs that do not move do not need to be reassigned a RB. If we do it, we will spuriously cause spontaneous handovers on the network, a situation undesirable both from the user's experience point of view and from the mathematical optimization one. Delaying possible handovers offers no drawbacks.

To see that this is true, suppose that we have an MT m at a given timestep t. Also, suppose that m is going to conflict colors with some other MT in a posterior timestep. If we proactively do a handover on m to avoid the future conflict, we will incur the same cost as doing it later. There is no gain, then, in doing preemptive handovers, yet there is the risk that we might miss our prediction and even cause an additional one. Now suppose more than one handover could be evicted by doing this preemptive handover. We would also gain the same advantage by doing it on the first handover in the sequence, so avoiding multiple handovers in conflicts with the same MT in a row becomes more of a choice of who keeps their color than a matter of doing preemptive handovers.

This said, we can optimize the calculation of the next timesteps with a bound on the number of handovers, much better than the total number of MTs. We can basically use the same trick used on the first timestep, but using only the MTs that moved between base stations *and* need a handover: for the base-stations containing a clash, separate the MTs who moved in into a new array and use it to generate all permutations of available colors inside that BS. Doing it for all base stations, we have a total cost of

$$O\left(\prod_{1}^{|B|}\frac{(\chi - (|b_i| - |h_i|))!}{|h_i|!}\right)$$

, where $|h_i|$ is the number of handovers on base station i. What this means is that we keep the assignments of all MTs that stayed put or moved without a clash $(|b_i| - |h_i|)$, and try all free colors for each of the remaining ones $(\chi - (|b_i| - |h_i|))$.

Putting together all the parts, we have a cost of

$$O\left(\sum_{c=1}^{C}\left(\prod_{i=1}^{|B|}\frac{c!}{|b_i|!} + \prod_{i=1}^{|B|}\frac{(c - (|b_i| - |h_i|))!}{|h_i|!}\right)\right)$$

4.3.2 Eliminating extrinsic BS coloring validation

For this step we will use an availability color index for each base station. For it to work we will make use of the order of color assignment in generating a new candidate scenario. Consider that we are evaluating the possible assignments for a given timestep (any timestep, initial or otherwise). We have already divided it into a sequence of intra-BS permutations on all colors. We will change this slightly to permutate only on the available colors, given the assignments of the previous BSs. The used colors of a BS's adjacency can be calculated by taking the color arrays of each adjacent base station, as well as its own, and applying a logical OR. If we invert this logical array we get the available colors, and permutate on them.

The term for the number of available colors in base station b, A_b, is presented in 2, where ADJ_i is the set of adjacent base stations of BS i, $|ADJ_i|$ is the size of the adjacency, and ADJ_{ij} is the j'th adjacent base station in i's adjacency. Notice that A_b will not change for different color assignments as it is only dependent on the number of MTs in the near vicinity and the chromatic number. We can substitute $c!$ for it in the previous cost equation to express the full cost of the algorithm as:

$$O\left(\sum_{c=1}^{C}\left(\prod_{i=1}^{|B|}\frac{A_b}{|b_i|!}+\prod_{i=1}^{|B|}\frac{(A_b-(|b_i|-|h_i|))!}{|h_i|!}\right)\right)$$

$$A_i = c - |b_i| - \sum_{j=1}^{|ADJ_i|} |b_{ADJ_{ij}}| \tag{2}$$

4.4 Giving a lower bound to the necessary number of colors

For this step let us assume that the MTs are equally distributed across the network. This presents the least number of MTs at any single BS and thus the potential for the smallest C. We have another constraint, which is that a color cannot be used at the same time at a base station and any of its adjacent ones.

Suppose an hexagonal topology with b base stations; we need to split our network along a intertwined groups, where a is the maximum adjacency size among base stations. Figure 9 illustrates this intertwined map. Notice how we cannot use the same color in any BS of different groups, but we can repeat a color in any two BSs of the same group without potential clashes.

Given this situation, we need

$$C \geq \frac{|M|}{|B|} * a$$

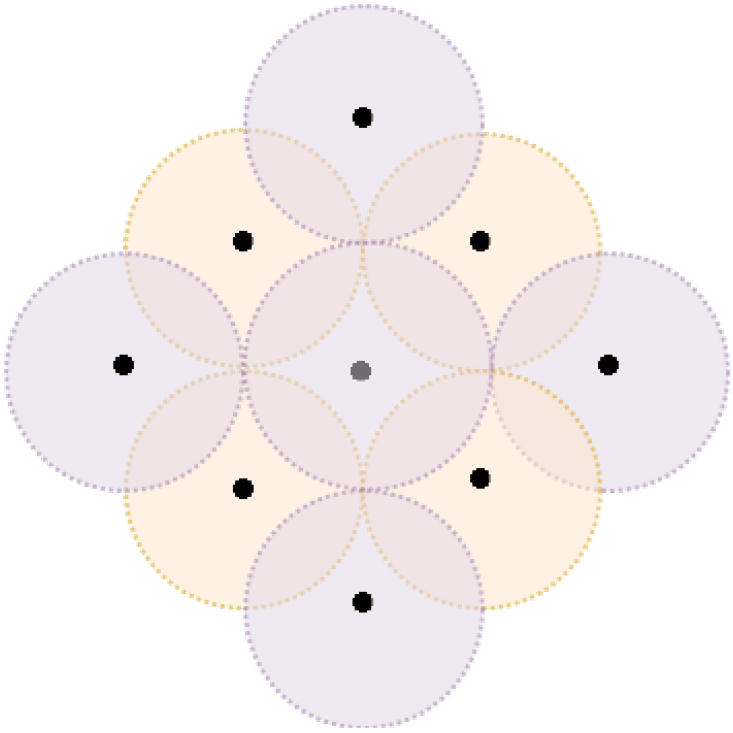

Figure 9 – A scenario with intertwined base-stations groups, each with their color-set.

colors. This represents that we will need at least the amount of MTs equally split among the base stations, with the caveat that the same colors cannot be used by adjacent groups, so we need to multiply by the adjacency size a.

For the upper bound we do not need $C \geq |M|$, as we will have more colors than MTs and can freely assign one to each.

The final cost of the algorithm can be expressed by

$$O\left(\sum_{c=\frac{|M|}{|B|}*a}^{C}\left(\prod_{i=1}^{|B|}\frac{A_b!}{|b_i|!}+\prod_{i=1}^{|B|}\frac{(A_b-(|b_i|-|h_i|))!}{|h_i|!}\right)\right)$$

.

4.5 Algorithms

Algorithm 4.1 presents the steps of the naive brute-force algorithm. We take as inputs the network trace, the adjacency list and C.

The core of the algorithm is the color generator. To implement it we will make use of two data structures, the ScenarioDescriptor (SD) and the Adjacency List (AL), which we will describe. The generator itself is divided into three levels: coloring, timestep, and base station. The first two levels share an implementation that differ only in determining the problem size and creation of its constituent generators, which we will also describe. In the end, we have a composite structure for the generator, as depicted in Figure 10.

Algorithm 4.1 Naive Brute-force

```
1:  Task NaiveBF(T, A, C)
2:      best_score = inf
3:      coloring = Coloring.from_trace(T, C)
4:      best = coloring.values()
5:      gen = ColoringGenerator(A, coloring)
6:      for _ in gen do
7:          c = coloring
8:          score_c = score(c)
9:          if score_c < best_score then
10:             best = coloring.value()
11:             best_score = score_c
12:     return best
```

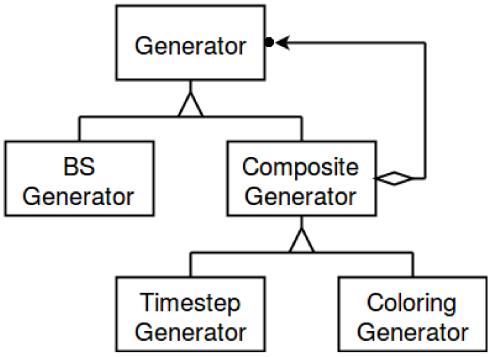

Figure 10 – Composite structure of the Generator data structure.

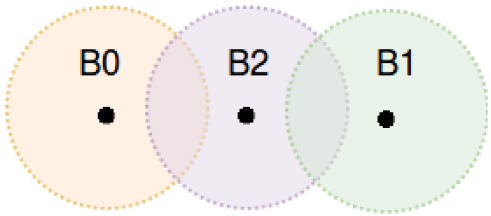

Figure 11 – Example scenario.

For this section we will use the example from Figure 11 to illustrate how the data structures are used.

4.5.1 Underlying data-structures: Scenario Descriptor and Adjacency List

The simplest of these two data structures is the adjacency list. It is composed of an index by BS id which points to an adjacency list which includes the base station itself. This is convenient because the calculations which use the adjacency list will also need to include the referenced base station.

The Scenario Descriptor is also a simple data structure per se. It is composed of a

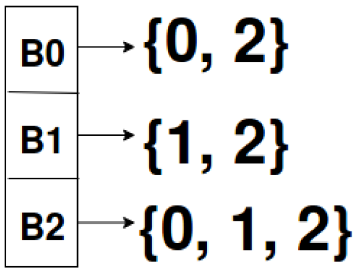

Figure 12 – Adjacency List.

	m0	m1	m2	m3
t0	0	0	1	1
t1	2	0	1	2

Figure 13 – Trace.

	m0	m1	m2	m3
t0	0	2	0	2
t1	1	2	0	3

Figure 14 – Possible coloring.

$[T \times MT]$ integer matrix, which will contain either BS ids, for the trace, or color numbers, for the coloring. It also has a companion data structure, the Scenario Descriptor Index (SDI), used to index the data for each base station. This way we can access the base station's information at any timestep for comparison. The SDI contains which MTs are part of which BS at each timestep. The data structures, as they would be applied to our example scenario, are presented in Figures 12, 13 and 14. The Trace and Coloring data structures use the SD and SDI to represent their internal state.

4.5.2 Generator

The Generator's job is to supply complete valid colorings to the brute force algorithm. It is composed of four classes: the CompositeGenerator, the ColoringGenerator, the TimestepGenerator, and the BSGenerator. Of these, the CompositeGenerator is an abstract class that implements the updating mechanism on any generator that conforms to the interface. It is used to implement the ColoringGenerator as a composition of TimestepGenerators, and TimestepGenerator as a composition of BSGenerators. The Coloring object is external to the iterators, and will be updated to reflect the most recently generated coloring. All the generators will reset their sections when exhausted, but

only the BSGenerator will fill the Coloring object with color values.

4.5.2.1 BSGenerator

Let us look first at the BSGenerator, which is more concrete and thus easier to grasp. It is initialized with a set of MTs, a set of colors, and a Coloring object, and will generate all color permutations for the available colors of size equal to the cardinality of the set of MTs. It should be empty in case the number of colors is less than the number of MTs, meaning that no coloring can be made. It should also be worth a single empty pass in case the number of MTs is 0. This is important for the higher level iterations, as failing this empty iteration when in the middle of the generator array will block the recursive progress and impossibilitate generation of all colorings. In the general case it will return the next available permutation and override the coloring section corresponding to the set of MTs. Notice that the BSGenerator receives only the coloring slice that corresponds to its timestep. When it is exhausted, it will clear its section (filling with color number -1). The class initialization, iterator initialization and iteration algorithms are presented in Algorithms 4.2 and 4.3 respectively. The python itertools library was used to generate permutations.

Algorithm 4.2 BSGenerator.___iter___

```
1: Task BSGenerator.___iter___
2:     if len(self.colors) < len(self.mts) then
3:         return iter([])
4:     if len(self.mts) == 0 then
5:         return iter([None])
6:     return self
```

Algorithm 4.3 BSGenerator.___next___

```
1: Task BSGenerator.___next___
2:     if has permutations then
3:         self.coloring[self.mts] = next(self.permutations)
4:     else
5:         self.coloring[self.mts] = -1
6:         raise StopIteration
```

4.5.2.2 CompositeGenerator

The CompositeGenerator is initialized with an adjacency matrix and a coloring object, and creates a Generator array with length equal to the problem size (defined by subclasses). In order to process the recursive iteration properly it has three phases: generating the first coloring, updating it to the next coloring, and exhaustion.

In the first time it is called, it will initialize all of its generators in ascending order, in a way that if it hits a dead end when trying to initialize any of the generators, it will backtrack to the previous generators until it can break through. This is accomplished by having a method *update_generator*(i), which takes care of iteration on i and previous generators if needed, until it can generate a next color for generator i or exhausts all possibilities, stopping the iteration. This method will ensure that we create a first valid coloring, or none in case it is impossible. This method is illustrated in Algorithm 4.4. Control of when it is first called is done by a flag. This step needs to be done on the first iteration, rather than on the iterator initialization, or we will lose the first coloring once the *next* method is called.

Algorithm 4.4 CompositeGenerator.first_color

```
1: Task CompositeGenerator.first_color
2:     for i in range(0, len(self.generators)) do
3:         self.generators[i] = iter(self.create_generator(i))
4:         self.update_gen(i)
```

In subsequent iterations, we only need to call the *update_generator* method on the last generator, as it will take care of generating a complete valid coloring recursively. Upon exhaustion, it calls a *cleanup* method, to be implemented by subclasses.

The *update_generator* method has three versions: recursive, semi-recursive, and iterative. We will start with the recursive version, as it is easier to grasp, and remove recursion step by step in the other versions.

The recursive version tries to increment the generator by the argument index. If it fails, then it calls itself on the previous index, re-initializes the current generator, and calls update on itself again. It should stop when the argument index is less than zero. This algorithm is presented in Algorithm 4.5.

Algorithm 4.5 CompositeGenerator.update_generator_recursive

```
1: Task CompositeGenerator.update_generator_recursive(idx)
2:     if idx < 0 then
3:         raise StopIteration
4:     if has next then
5:         next(self.generators[idx])
6:     else
7:         self.update(idx-1)
8:         self.generators[idx] = self.create_generator(idx)
9:         self.update_generator(idx)
```

To remove the innermost recursion we will use a $finished$ flag. We will retry updating the generator on the argument index while the flag is false, going through the same steps when it fails, except for the recursive call on the same index, and setting the flag to true

once we can successfully update the generator. The stopping criterion is the same. This algorithm is presented in Algorithm 4.6.

Algorithm 4.6 CompositeGenerator.update_generator_semi_recursive

```
1: Task CompositeGenerator.update_generator_semi_recursive(idx)
2:     if idx < 0 then
3:         raise StopIteration
4:     finished = False
5:     while not finished do
6:         if has next then
7:             next(self.generators[idx])
8:             finished = True
9:         else
10:            self.update(idx-1)
11:            self.generators[idx] = self.create_generator(idx)
```

The final, iterative version, of the algorithm uses an iteration index and a creation flag, to track the generator being currently updated and necessity for re-initializing the next iterator. It starts by setting the iteration index i to the argument index idx, and the creation flag to False, and iterates on i while $0 \leq i \leq idx$. The loop follows a two-case structure: if it can increment the generator at i, then it moves on to the next index and needs to re-initialize the generator at that position (set creation flag to True); if it cannot increment the generator at i, then it retries on the previous generator. At the start of any iteration, if the creation flag is True (meaning it comes from an index increment) then it will re-initialize the generator at this new i and reset the creation flag to False. If it ends by exceeding idx, then it returns with a new coloring, but if the iteration ends with $i < 0$, then all possibilities where exhausted and the iteration stops. This iterative version is presented in Algorithm 4.7.

4.5.2.3 CompositeGenerator

The CompositeGenerator subclasses need to implement 3 methods: *create_generator*, *problem_size*, and *cleanup*. The ColoringGenerator is simpler: its problem size is the number of timesteps, it creates a TimestepGenerator by passing t along with the other parameters, and on cleanup it should set the whole coloring object to -1.

The TimestepGenerator has trivial problem size, equal to the number of base stations (length of adjacency list), and trivial cleanup (set whole timestep to -1, but the create generator step is a bit tricky. First, as long as it is not the first timestep, it needs to carry over the used colors for the MTs who do not suffer a handover. It literally copies the colors from the previous timestep. This is illustrated in Algorithm 4.8. Second, it must pass to the BSGenerator a list containing the MTs that need recoloring, a list of available colors, and its coloring slice. These mt-list and available color calculations will be presented further, and this rather high level algorithm is presented in Algorithm 4.9.

Algorithm 4.7 CompositeGenerator.update_generator_iterative

```
1: Task CompositeGenerator.update_generator_iterative(idx)
2:     i = idx
3:     create = False
4:     while 0 ≤ i ≤ idx do
5:         if create is True then
6:             self.generators[i] = iter(self.create_generator(i))
7:             create = False
8:         if has next then
9:             next(self.generators[i])
10:            i += 1
11:            create = True
12:        else
13:            i -= 1
14:    if i < 0 then
15:        raise StopIteration
```

Algorithm 4.8 TimestepGenerator.do_carryover

```
1: Task TimestepGenerator.do_carryover
2:     if self.t > 0 then
3:         co_mts = self.coloring.carryover_mts(self.t, bs, self.adjacency[bs])
4:         self.coloring[self.t, co_mts] = self.coloring[self.t - 1, co_mts]
```

Algorithm 4.9 TimestepGenerator.create_generator

```
1: Task TimestepGenerator.create_generator
2:     self._do_carryover(bs)
3:     return BSGenerator(self.coloring.ho_mts(self.t, bs, self.adjacency[bs]),
   self.coloring.available_colors(self.t, self.adjacency[bs]), self.coloring.timestep(self.t))
```

4.5.3 In-Coloring calculations

With all the other pieces in place, we need the coloring object to answer these three questions for any given BS and timestep:

- ❏ which MTs need recoloring?
- ❏ which MTs do not need recoloring?
- ❏ what are the available colors?

For this the Coloring will be represented by a ScenarioDescriptor and two extra components: a base-station index (SDI) and χ. It will also need a companion data structure, the ColorTracker, which will index (mt, color) pairs to answer the following questions:

- ❏ What MTs are conflicting colors?
- ❏ What colors are free?

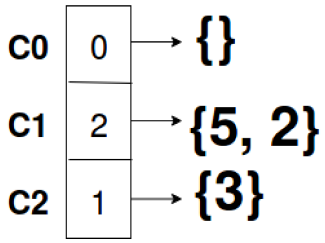

Figure 15 – ColorTracker example.

❏ What colors are unavailable?

The questions about MTs that do and do not need recoloring are better understood declaratively using set operations. There are four sets of MTs to be composed in calculating these answers: MTs that are new to the base station N, MTs that were already in this base station O, conflicting MTs C and MTs with unavailable colors U. N and O are trivial to calculate: just compare the MTs on this base station at this timestep with the ones from the previous timestep. This is easily accomplished with our base-station index.

The C set is composed of the new MTs that use the same color, in which case both need to be recolored. It is calculated by creating a ColorTracker with the new MTs and checking for the colors that contain more than one MT.

The U set is composed of MTs whose colors are present in the adjacency. This means that they were already used to color some MT and cannot be used again. It is calculated by creating a ColorTracker for the adjacency at the previous timestep and discounting the colors being used by these moving-in MTs.

We can then define the MTs that suffer handover as: None, if it is the first timestep, or $N \wedge (C \vee U)$ otherwise. We can also define the MTs that do not suffer handover (a.k.a. carryover MTs) as: None, in the first timestep, or $O \vee ((N \setminus C) \setminus U)$ otherwise. This means that all new MTs or MTs that conflict or have unavailable colors must suffer handover, and that all old MTs or new MTs that do not conflict nor have unavailable colors do not need a handover. The set of available colors is trivially calculated by creating a ColorTracker for the adjacency and returning the free colors.

The ColorTracker is a simple hash-like index: it stores a counter array to keep how many MTs have each color, and stores all MTs containing that color in a list to be referenced by the color number. Each time we add a new (mt, color) pair, we increase the counter and store the MT in the list. We know the free colors as they have a counter equal to 0, and we know the unavailable colors (optionally discounting for a passed-in color array) as they will have a counter higher than 1. We can also know the conflicting MTs by joining the lists for the positions with a counter higher than 1. Figure 15 illustrates the data structure.

Chapter 5

Experiments

This chapter starts with a discussion of the scenarios used in testing, and proceeds to discuss how each algorithm performed using different parameters. We measure both scalability and solution quality.

5.1 Scenarios

We ran tests against the scenario presented at (WANG et al., 2016a), which is comprised of an hexagonal 7-cell architecture. Movement patterns were considered by forcing the movement of each MT at each timestep, during a fixed number of timesteps. In practice the BS of each MT at the next timestep is drawn from the adjacent base stations. We used the following parameters:

- Number of MTs: [1-10]
- Number of colors: [1 ... Number of MTs]
- Number of timesteps: 5

Attempting to generate realistic scenarios with BonnMotion (ASCHENBRUCK et al., 2010), using ManhattanGrid models of differing sizes, resulted in sparse scenarios that took long to run and had trivial solutions, comprised of one color for each MT with no possibility of reuse. Such maps are often too vast for any significant MT movement in such a small amount of timesteps and did not yield interesting movement patterns.

To give a sense of scale, even cases with one block of $100m \times 100m$, with one base station on each corner, proved to yield few HOs in a realistic timestep scale. The movement of the MTs is too slow to cause transitions in 5 seconds that would result in handovers. In a scenario with 4 base stations, 10 MTs and 5 seconds, using a random walk, we could barely get one or two transitions. This conveys two important points: first, that recoloring scenarios will not be so complex, so the algorithm in practice does not need to be

a prediction powerhouse; second, that it is probably overkill to compute the colorings over a dense network representation. Working over a data structure that allows efficient access to transitions at each timestep and MT journey might yield more efficient algorithms without losing on coloring quality.

We nevertheless took the dense brute-force approach as other works have. To this end we implemented a simple generator that calculates the next position of each MT based on its adjacency for the required number of timesteps. The results are presented below.

5.2 Results

We ran each brute-force scenario a single time, since its execution is deterministic. Each random-pruning scenario was executed 10 times.

5.2.1 Valid-only brute-force

We present the raw experiment data in Table 1. We can see that for most tests the execution time remains constant and yields a best-case coloring (0 HandOvers). For the tests that start taking longer the number of equivalent solutions rises fast, along with the time of experiment. This reflects the many equivalent solutions for a complex scenario. A first way to explore a faster search in this case is to randomly prune the search branches, which is what we tried next. We also plot the time progression for the scenarios consisting of each separate number of MTs as we increase the color count. This is shown in Figure 16.

5.2.2 Using random pruning

We present in Table 2 the experiment data for the tests with random pruning. For these we randomly stop the evaluation of a search branch with a fixed probability. This chance is evaluated at every time we generate a coloring permutation for a base station. Each reported value is the average of results for 50 runs of each experiment.

The first round used 50% chance when pruning. We see that for many cases it was unable to find even one valid solution, even when given enough colors. Running times show a slight upward trend at a much lower time scale, less than 1 second for any investigated problem size. The time curves are presented in Figure 17.

#MTs	#Colors	Time (s)	# equivalent solutions	#HOs
1	1	0.003002	0.00	0.00
2	1	0.001346	0.00	0.00
	2	0.001747	0.00	0.00
3	1	0.001321	0.00	0.00

	2	0.001793	0.00	0.00
	3	0.004390	0.16	0.08
4	1	0.001307	0.00	0.00
	2	0.001666	0.00	0.00
	3	0.009404	0.36	0.06
	4	0.007229	0.34	0.06
5	1	0.001274	0.00	0.00
	2	0.001787	0.00	0.00
	3	0.007070	0.36	0.08
	4	0.009016	0.50	0.12
	5	0.004766	0.32	0.06
6	1	0.001328	0.00	0.00
	2	0.001755	0.00	0.00
	3	0.007763	0.32	0.08
	4	0.005513	0.34	0.10
	5	0.008478	0.50	0.06
	6	0.012240	0.72	0.10
7	1	0.001403	0.00	0.00
	2	0.001629	0.00	0.00
	3	0.006841	0.28	0.04
	4	0.005719	0.30	0.04
	5	0.008236	0.34	0.08
	6	0.015348	0.76	0.08
	7	0.012795	0.78	0.12
8	1	0.001311	0.00	0.00
	2	0.001858	0.00	0.00
	3	0.005560	0.30	0.08
	4	0.006952	0.36	0.08
	5	0.014184	1.06	0.24
	6	0.008304	0.52	0.10
	7	0.012029	0.54	0.10
	8	0.011204	0.46	0.04
9	1	0.001519	0.00	0.00
	2	0.002123	0.00	0.00
	3	0.004073	0.14	0.04
	4	0.010130	0.58	0.14
	5	0.008747	0.48	0.06
	6	0.007135	0.40	0.04
	7	0.007826	0.44	0.04

	8	0.010582	0.66	0.08
	9	0.010624	0.66	0.04
10	1	0.001315	0.00	0.00
	2	0.001798	0.00	0.00
	3	0.008030	0.36	0.02
	4	0.009048	0.50	0.12
	5	0.031816	0.88	0.14
	6	0.014948	0.60	0.06
	7	0.014034	0.68	0.12
	8	0.009977	0.52	0.10
	9	0.012159	0.70	0.12
	10	0.007953	0.34	0.02

Table 2 – Test results for random pruning with 50% chance. Average over 50 runs for each combination of #MTs, #Colors.

The second round used 25% chance for pruning. We present in Table 3 the experiment data for these tests. We see that it was still unable to find valid solutions for many cases, but in this regard it scored much better than random pruning with 50% chance. Running times show an exponential upward trend, but at a much lower time scale than the brute force at less than 1 second for any investigated problem size. The time curves are presented in Figure 18.

#MTs	#Colors	Time (s)	# equivalent solutions	#HOs
1	1	0.001432	0.00	0.00
2	1	0.001383	0.00	0.00
	2	0.003544	0.00	0.00
3	1	0.002204	0.00	0.00
	2	0.003886	0.00	0.00
	3	0.046645	1.54	0.06
4	1	0.001402	0.00	0.00
	2	0.003644	0.00	0.00
	3	0.041009	1.20	0.08
	4	0.118902	3.92	0.06
5	1	0.001394	0.00	0.00
	2	0.003704	0.00	0.00
	3	0.047929	1.70	0.08
	4	0.082922	3.30	0.12
	5	0.171783	6.26	0.08
6	1	0.001432	0.00	0.00

	2	0.003381	0.00	0.00
	3	0.043942	1.42	0.06
	4	0.073953	2.66	0.04
	5	0.134522	4.84	0.04
	6	0.188622	7.94	0.08
7	1	0.002120	0.00	0.00
	2	0.003881	0.00	0.00
	3	0.039560	1.30	0.06
	4	0.095298	3.44	0.08
	5	0.162225	6.92	0.04
	6	0.195013	7.24	0.06
	7	0.236402	10.52	0.02
8	1	0.001573	0.00	0.00
	2	0.003607	0.00	0.00
	3	0.041753	1.38	0.08
	4	0.084951	3.40	0.08
	5	0.169339	6.98	0.10
	6	0.237853	9.98	0.08
	7	0.223513	10.98	0.04
	8	0.289968	12.90	0.04
9	1	0.001382	0.00	0.00
	2	0.003646	0.00	0.00
	3	0.039135	1.18	0.06
	4	0.096627	3.32	0.04
	5	0.140969	5.28	0.08
	6	0.205040	9.48	0.06
	7	0.212568	12.50	0.10
	8	0.175220	11.50	0.04
	9	0.277696	17.46	0.06
10	1	0.001431	0.00	0.00
	2	0.003219	0.00	0.00
	3	0.033790	1.16	0.06
	4	0.109213	3.32	0.02
	5	0.185368	6.68	0.04
	6	0.239807	9.28	0.10
	7	0.195067	8.86	0.06
	8	0.247531	11.86	0.06
	9	0.334270	16.58	0.04
	10	0.291506	16.52	0.00

Table 3 – Test results for random pruning with 25% chance. Average over 50 runs for each combination of #MTs, #Colors.

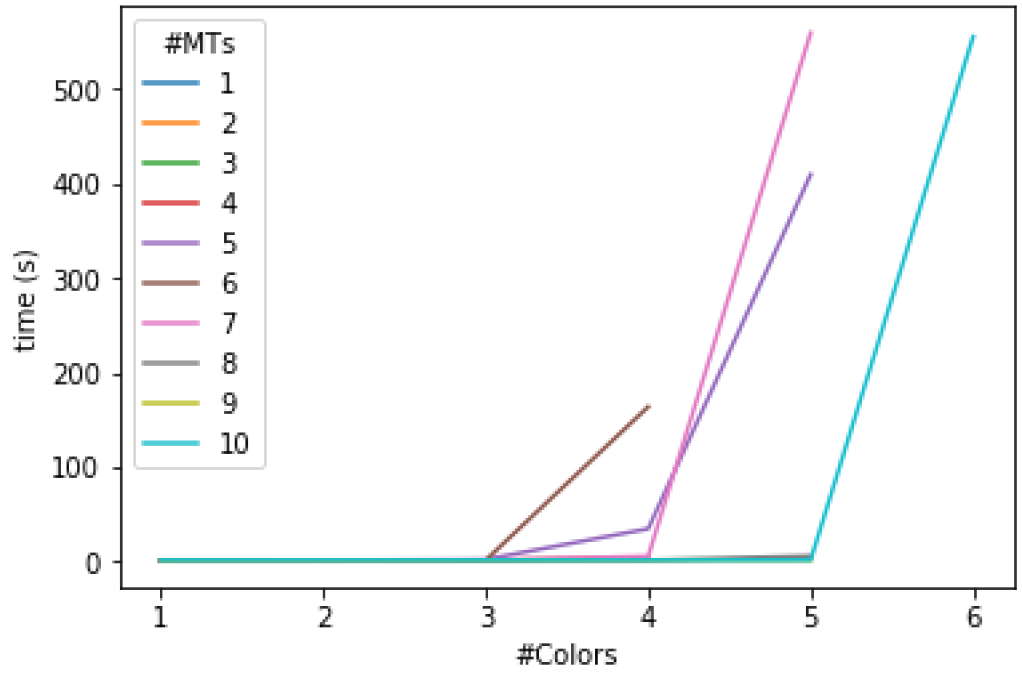

Figure 16 – Running times of brute-force algorithm as the number of colors grow. Each curve depicts a scenario with a different number of MTs.

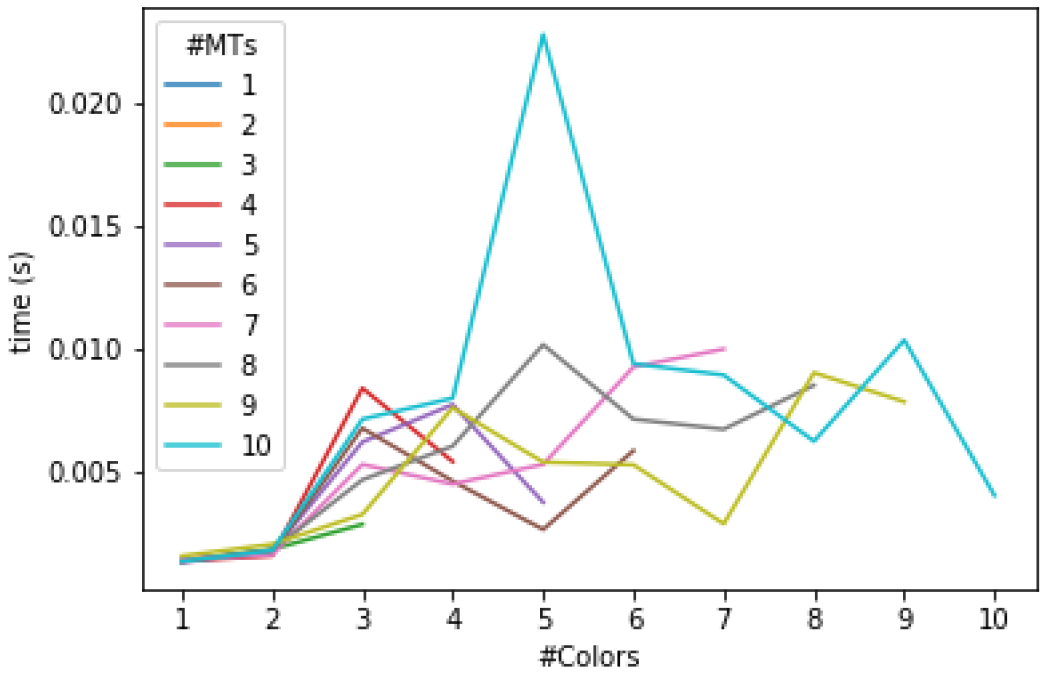

Figure 17 – Execution times for random pruning with 50% chance.

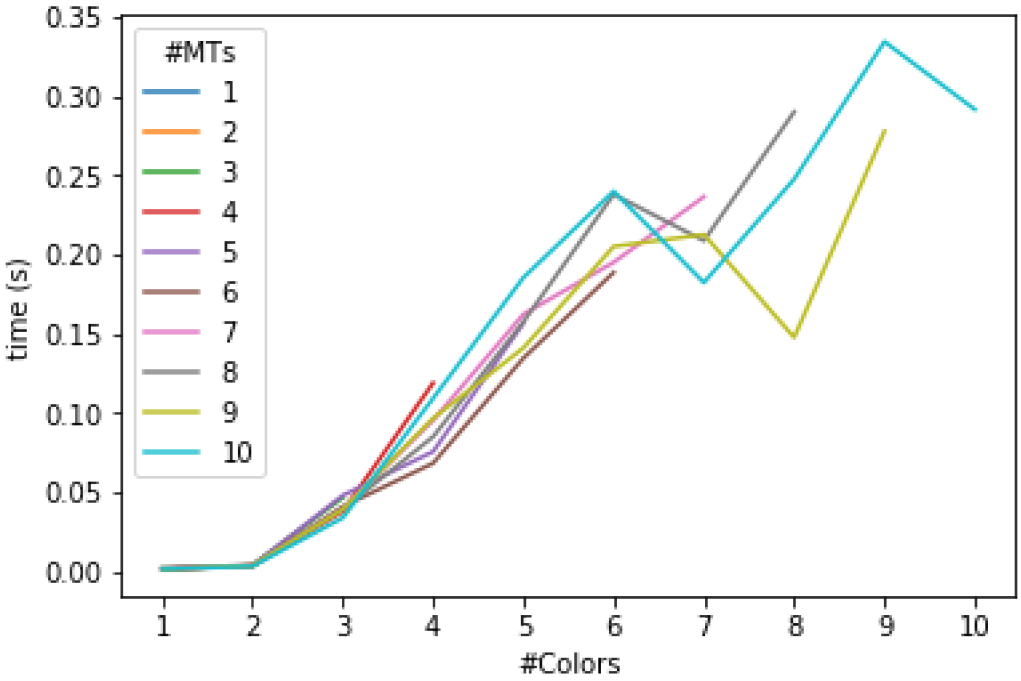

Figure 18 – Execution times for random pruning with 25% chance.

#MTs	#Colors	Time (s)	# equivalent solutions	#HOs
1	1	1.5	1	0
2	1	1.5	0	0
2	2	1.6	2	0
3	1	1.5	0	0
3	2	1.5	0	0
3	3	1.8	6	0
4	1	1.5	0	0
4	2	1.5	0	0
4	3	1.6	0	0
4	4	4	24	0
5	1	1.5	0	0
5	2	1.5	0	0
5	3	2.3	6	4
5	4	34.8	24	0
5	5	410	240	0
6	1	1.5	0	0
6	2	1.5	0	0
6	3	1.5	0	0
6	4	164	168	2
7	1	1.5	0	0
7	2	1.5	0	0
7	3	1.5	0	0
7	4	5.7	0	0
7	5	560	3	3
8	1	1.5	0	0
8	2	1.5	0	0
8	3	1.5	0	0
8	4	1.6	0	0
8	5	6	0	0
9	1	1.3	0	0
9	2	1.4	0	0
9	3	1.5	0	0
9	4	1.6	0	0
9	5	1.8	0	0
10	1	1.5	0	0
10	2	1.5	0	0
10	3	1.5	0	0
10	4	1.5	0	0
10	5	1.7	0	0
10	6	556	0	0

Table 1 – Test results for brute-force coloring. Tests cases that took longer than 600 seconds are omitted from the table.

Chapter 6

Conclusion

In this work we have studied the problem of Resource Band Assignment (RB-A) in 5G networks. We have modeled the network in terms of Hypergraphs, but decided to run tests with a simpler model due to computational constraints.

We have advanced the theoretical formulations towards a solid algorithm for calculating optimal colorings. The problem formalization is solid and seems to reflect its practical nature. We have presented an algorithm capable of exploring only the valid instances, going through each of them exactly once, and a faster algorithm using random pruning.

Our attempt to tackle practical instances was not successful due to the scenario size needed for complex coloring choices. We found out that other work in this area uses condensed cases, and that the practical instances we generated through simulation were not causing as many handovers as we would expect for interesting cases, at least not in a scale we could calculate solutions for.

We have attained satisfactory advancements in evaluation of handover minimization performance, although we have not focused on exploring production algorithms. The random pruning algorithm we tested is just a small example of what can be compared against our benchmark. The opportunity for production-grade algorithms in this problem lies in approximation algorithms. Maybe artificial intelligence techniques such as genetic programming can yield good results. Its combinatorial nature could be put to good use, for example, by crossing solutions on points where a color would be exchanged between MTs. Future work could explore more specific formulations.

We have also calculated the complexity of the optimized brute force algorithm. The complexity of a benchmark tool is always expected to be poorer than a production-grade algorithm, but its complexity is too high for large scenarios. We could probably settle for some approximation algorithms on such cases. Its formulation serves as a base, however, and can be used to judge further developments of these benchmark tools by comparing evaluation performance on the cases it can handle.

The software developed for this work can be found on under the author's GitHub profile (FRANCO, 2018).

Bibliography

ASCHENBRUCK, N. et al. Bonnmotion: a mobility scenario generation and analysis tool. In: ICST (INSTITUTE FOR COMPUTER SCIENCES, SOCIAL-INFORMATICS AND TELECOMMUNICATIONS ENGINEERING). **Proceedings of the 3rd international ICST conference on simulation tools and techniques**. 2010. p. 51. Disponível em: <https://doi.org/10.4108/ICST.SIMUTOOLS2010.8684>.

BASU, P. et al. Modeling and analysis of time-varying graphs. **arXiv preprint arXiv:1012.0260**, 2010.

CASTEIGTS, A. et al. Time-varying graphs and dynamic networks. **arXiv preprint arXiv:1012.0009**, 2010.

CHANG, D.-J. et al. Compute pairwise manhattan distance and pearson correlation coefficient of data points with gpu. In: IEEE. **Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 2009. SNPD'09. 10th ACIS International Conference on**. 2009. p. 501–506. Disponível em: <https://doi.org/10.1109/SNPD.2009.34>.

CHEKURI, C.; LI, S. A note on the hardness of approximating the k-way hypergraph cut problem. 2015.

COMPANY, A. **LTE Resource Guide**. 2009. Disponível em: <http://web.cecs.pdx.edu/~fli/class/LTE_Reource_Guide.pdf>.

COWLING, P. Total colouring of hypergraphs. **Journal of Combinatorial Mathematics and Combinatorial Computing**, Charles Babbage Research Centre, v. 19, p. 151–160, 1995.

FRANCO, L. M. **VCRAN Handover Optimization Benchmark**. 2018. <https://github.com/lmarchesoti/vcran-ho-optimization-benchmark>.

IRMER, R. et al. Coordinated multipoint: Concepts, performance, and field trial results. **IEEE Communications Magazine**, IEEE, v. 49, n. 2, p. 102–111, 2011. Disponível em: <https://doi.org/10.1109/MCOM.2011.5706317>.

KAZNACHEY, D.; JAGOTA, A.; DAS, S. Neural network-based heuristic algorithms for hypergraph coloring problems with applications. **Journal of Parallel and Distributed Computing**, Elsevier, v. 63, n. 9, p. 786–800, 2003. Disponível em: <https://doi.org/10.1016/S0743-7315(03)00064-9>.

KUNDGEN, A.; MENDELSOHN, E.; VOLOSHIN, V. Colouring planar mixed hypergraphs. **JOURNAL OF COMBINATORICS**, v. 7, n. 2, p. R60–R60, 2001.

MEHROTRA, A.; TRICK, M. A. A branch-and-price approach for graph multi-coloring. In: **Extending the horizons: Advances in computing, optimization, and decision technologies**. Springer, 2007. p. 15–29. Disponível em: <https://doi.org/10.1007/978-0-387-48793-9_2>.

RIVA, M. et al. An elastic optical network-based architecture for the 5g fronthaul. In: **Simpósio Brasileiro de Redes de Computadores (SBRC)**. [S.l.: s.n.], 2018. v. 36.

TININI, R. I. et al. Optimal placement of virtualized bbu processing in hybrid cloud-fog ran over twdm-pon. In: IEEE. **GLOBECOM 2017-2017 IEEE Global Communications Conference**. 2017. p. 1–6. Disponível em: <https://doi.org/10.1109/GLOCOM.2017.8254770>.

WANG, X. et al. Handover reduction in virtualized cloud radio access networks using an optical fronthaul. 2016. Under evaluation. Disponível em: <https://doi.org/10.1364/JOCN.8.00B124>.

_____. Handover reduction in virtualized cloud radio access networks using twdm-pon fronthaul. **Journal of Optical Communications and Networking**, Optical Society of America, v. 8, n. 12, p. B124–B134, 2016. Disponível em: <https://doi.org/10.1364/JOCN.8.00B124>.

YANG, Q.; CHAN, C. C.-K. A switching architecture for remote radio head protection in cloud radio access networks. In: IEEE. **Microwave Photonics (MWP), 2017 International Topical Meeting on**. 2017. p. 1–3. Disponível em: <https://doi.org/10.1109/MWP.2017.8168666>.

YU, F. et al. Algorithms for channel assignment in mobile wireless networks using temporal coloring. In: ACM. **Proceedings of the 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems**. 2013. p. 49–58. Disponível em: <https://doi.org/10.1145/2507924.2507965>.

YU, H. et al. Energy-efficient dynamic lightpath adjustment in a decomposed awgr-based passive wdm fronthaul. **Journal of Optical Communications and Networking**, Optical Society of America, v. 10, n. 9, p. 749–759, 2018. Disponível em: <https://doi.org/10.1364/JOCN.10.000749>.

ZHANG, J. et al. Reconfigurable optical mobile fronthaul networks for coordinated multipoint transmission and reception in 5g. **IEEE/OSA Journal of Optical Communications and Networking**, IEEE, v. 9, n. 6, p. 489–497, 2017. Disponível em: <https://doi.org/10.1364/JOCN.9.000489>.